Gait Analysis and Rehabilitation Using Web-Based Pose Estimation

Dominic Mark Richmond

Master of Science (by research)

University of York
Physics, Engineering and Technology
January 2023

Abstract

Gait abnormalities are one of the most common health conditions in the elderly population, with almost one in three people over 60 experiencing symptoms that disrupt their movement [1]. These symptoms can cause disability [2] and present an increased fall risk [3] [4]. Detecting these abnormalities early is, therefore, crucial as it reduces the likelihood of injuries and accidents.

Current treatments for gait abnormalities depend on the condition, but many treatment plans commonly incorporate some form of physiotherapy. Clinicians typically deliver physiotherapy in the form of gait assessments and targeted exercises or therapies. Recent research has also shown that virtual reality (VR) treadmill walking, using motion capture technology, can be an effective method of treating certain gait abnormalities [5] [6] [7].

This thesis covers the development of a web-based VR treadmill walking system to make VR physiotherapy cheaper and more accessible. The system uses convolutional neural networks to assess the patient's gait from an RGB webcam feed and provides them with live feedback on their body position within a VR environment. The system's gait assessment capabilities are validated by comparing it to a gold standard – the OptiTrack motion capture system.

The results demonstrate that the system's percentage error ($\tilde{\epsilon}_{\%}$) was much less for temporal gait metrics ($0.24 < \tilde{\epsilon}_{\%} < 12.40$) than it was for spatial ones ($70.90 < \tilde{\epsilon}_{\%} < 79.72$). Four out of five spatial metrics also had a "very strong correlation" ($0.74 < r < 0.86$) when compared to the OptiTrack's metrics, meaning the accuracy could be increased using a gain factor. These findings establish the basis for a similar study with a larger sample size. They also raise the possibility that this system could analyse gait in the clinic and the home without specialist motion capture equipment or facilities.

Acknowledgements

I would like to thank the people who supported and championed me during this project. Firstly, my supervisor Dr. Adar Pelah and my second supervisors: Doctor Eugene Avrutin and Prof. Zion Tse. Next, I would like to thank my partner Maisie Hardy, her family, and my family: Rachelle Richmond, Mark Richmond, Lucy Richmond, Molly Richmond, Bertie Richmond, Julie Hardy, Keith Hardy, and Ned Hardy. I would also like to express my gratitude for the resources and support provided during the course of this research by my employer, Asuuta Ltd. Finally, I would like to thank my friends from LIVE, Vignesh Radhakrishnan, Cai Xin Chen and Zadok Storkey.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, university. The work makes use of resources kindly made available by Asuuta Ltd's StepSense Clinic system [8] [9] and StroMoHab Android system [10]. All sources are acknowledged as references.

Contents

List of Figures

List of Tables

Overview

This thesis covers the design, implementation, testing and validation of a web-based gait analysis and biofeedback system. Physiotherapists could use this system as a cost-effective tool to diagnose and rehabilitate conditions that affect the motor system, such as Parkinson's Disease, Long COVID, and Musculoskeletal disorders.

Chapter 1 introduces the problem, describing what a movement disorder is and giving examples of common disorders. It then explains why diagnosing and treating these disorders is important, establishing a motivation for the research. Finally, the chapter describes some of the solutions already available for the diagnosis and treatment of these disorders.

Chapter 2 describes the proposed solution and introduces the different technologies that could form parts of the system's tech stack. It lists the advantages and disadvantages of each technology and decides upon the final tech stack. Finally, the chapter details the project management strategy this thesis will use and describes the aims and objectives that will measure the thesis's success.

Chapters 3 - 5 detail the design, implementation and testing process for each version of the system. Chapter 3 includes a list of user stories and a specification for each version of the system. It also explains the system's design in more detail with flow diagrams of the architecture and UI designs. Chapter 4 explains how key features of the system work at a programming level using code snippets. Chapter 5 describes how the system's code is tested, providing a list of unit tests, visual tests, and the testing results.

Chapter 6 documents the process of validating the web app with respect to the gold standard. It describes the validation protocol and discusses the problems encountered during data collection. Finally, the chapter explains the data processing methods in the validation step and discusses the validation results.

Chapter 7, evaluates the system and proposes next steps for a study that will further validate it for clinical use. The chapter also provides ideas for future system features.

Finally, chapter 8 reiterates what the thesis has covered and describes how the aims and objectives have been fulfilled.

1 Introduction

1.1 Gait Disorders and Treatments

Gait refers to the locomotive movements of a living thing (a human in the context of this thesis). According to a 2013 study [1], almost one in three people over 60 experience abnormalities in their gait. These abnormalities are well known to correlate with decreased mood [35], chronic pain [36] and disability [2] along with an array of other symptoms. Neurological gait problems also present a significant fall risk [3] [4] which increases the risk of mortality [37]. Because of these reasons, it is crucial to identify gait problems early on in order to prevent serious accidents.

One such gait disorder is Parkinson's Disease. Parkinson's disease is a neurological disorder that affects how neurons in the brain fire, causing symptoms such as "slow movement, tremor, rigidity and imbalance" [38], all symptoms that affect gait quality. Some common gait-specific symptoms of Parkinson's Disease include reduced stride length resulting in increased cadence [39] (see section 1.2), freezing of gait (inability to start or continue walking) [40], and Festinations (increasingly smaller steps sometimes occurring after freezing of gait) [41]. It is a degenerative condition [38] and, according to the WHO, is increasing global disability and death "faster than for any other neurological disorder" [38]. Parkinson's UK lists three main treatments for the disease [42]:

- Drugs such as levodopa, dopamine agonists, MAO-B inhibitors, COMT inhibitors, amantadine, and anticholinergics.
- Physical activity.
- Conventional therapies such as physiotherapy, speech and language therapy, and occupational therapy.

Papers often cite levodopa as the main treatment option for Parkinson's disease [43]. Physical activity/therapy is usually prescribed alongside levodopa, as it also has a positive effect [44].

Musculoskeletal conditions can also affect gait [45]. The term "Musculoskeletal (MSK) conditions" covers a wide range of problems associated with muscles, bones, joints and multiple body areas or systems [46]. There are three categories of MSK conditions [47]:

- Inflammatory conditions e.g. rheumatoid arthritis.
- Conditions causing MSK pain e.g. osteoarthritis and back pain.
- Osteoporosis and fragility fractures.

The gait parameters affected by MSK conditions vary widely and depend on the specific diagnosis. Examples of deviations include decreased joint moments in Osteoarthritis [48] and Rheumatoid Arthritis [49]. According to the NHS, patients with musculoskeletal disorders account for 30% of GP consultations in England [45]. The treatment for MSK disorders varies based on the specific issue, but care providers often use physiotherapy as an initial treatment.

Another condition known to affect gait is Multiple Sclerosis (MS). MS is a neurological condition affecting the brain, spinal cord and optic nerves [50]. Whilst the cause of MS is unknown, scientists believe it is caused by the immune system attacking the central nervous system [50]. Some of the most common symptoms are fatigue, cognitive changes, and spasticity (stiff limbs) [51]. Some commonly reported deviations in gait include reduced walking speed [52][53][54], reduced step length [52][53], heightened variability in joint angles [54], and changes in temporal parameters such as double support [52]. Medical professionals usually treat MS using a comprehensive approach [55] involving disease-modifying medications, steroids (during flare-ups), and rehabilitation (both mental and physical) [55].

Cerebral palsy (CP) can also cause abnormal gait. CP happens when the movement centres of the brain do not develop as they should [56]. This causes symptoms such as stiff muscles , spasticity, uncontrollable movements, poor balance and co-ordination and seizures [56]. Spasticity usually causes the gait deviations observed in CP. Commonly observed gait deviations include decreased range of motion at the ankle, knee and hip [57]. Increased hyperextension at the knee and hip is also common [57]. Hyperextension involves the joint extending past its normal maximum extension. CP usually appears at a young age and is prevalent throughout the person's life [56]. Whilst there is no cure for CP, it can still be managed using medication (for seizures, muscle control and pain management), surgery, and therapy (physical, occupational and mental) [58].

More recently, the COVID-19 virus has been reported to cause gait abnormalities. COVID-19 is a respiratory illness that first appeared on the 31st of December 2019 in Wuhan, China [59]. It received the status of "global pandemic" on the 11th March 2020 [60] and continues to affect billions of people.

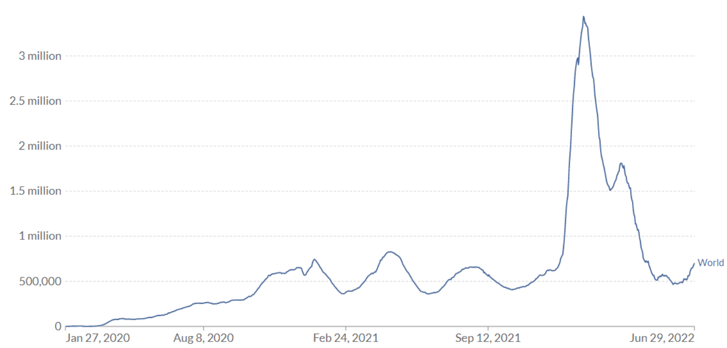

Figure 1.1: Graph of world COVID cases per day over the last two years (from [11]).

The most common symptoms of COVID-19 the World Health Organization (WHO) describes are fever, cough, tiredness, and loss of taste or smell [61]. These symptoms usually take 5-6 days to show [61], and recovery usually takes one to two weeks [62]. The severity of COVID-19 varies greatly depending on the individual. The age and general health of the person infected are known to affect this. According to the WHO, COVID-19 can also cause long-term complications such as fatigue, shortness of breath, cognitive dysfunction [63] and joint pain [63]. Some studies have observed the potential of these long-term complications to affect gait [64], particularly symmetry of temporal measures such as double and single support [65]. Treatments for the immediate symptoms of COVID-19 come in the form of antiviral medicines (Paxlovid, Remdesivir and Molnupiravir) and neutralising monoclonal antibody (nMAb) treatments (Sotrovimab) [66]. Treatments for the long-term symptoms can come in many different forms and are handled on a case-by-case basis as the spectrum of symptoms is so large [67]. Common treatments include pulmonary rehabilitation, physiotherapy, occupational therapy and modifying diet [67].

Serious cases of COVID-19 also put people at higher risk of PICS. PICS is a disability that can arise from the physical and mental trauma associated with ICU admissions. Generalized weakness, fatigue, decreased mobility, anxious or depressed mood, sexual dysfunction, sleep disturbances, and cognitive issues [68] are all symptoms that can occur. The risk factor relating COVID-19 to PICS is respiratory failure requiring prolonged mechanical ventilation [68]. Much like the long-term symptoms of COVID-19, the treatments are wide-ranging, typically involving a combination of physiotherapy, mental health support/medication, medication to treat injuries and speech therapy.

1.2 Diagnosing Gait Disorders

Clinicians measure gait using different metrics they calculate from their observation of the patient walking. The clinician compares the calculated metrics to the expected healthy values to detect the presence of abnormal gait. This comparison can help to identify the presence of different conditions affecting movement. Access to gait analysis services is important, as early detection of abnormalities prevents falls occurring and allows for early intervention, which can "reduce the frequency of nursing home admissions" [69].

Gait is measured and classified using a variety of metrics (some metric definitions taken and adapted from "Terminology of Human Walking" [70]):

- Step count - Number of steps taken.
- Velocity - Change in distance walked divided by change in time elapsed.
- Cadence - Step count divided by time elapsed.
- Heel strike - When the heel first touches the floor.
- Toe-off - When the toe first leaves the floor.
- Stride length - Distance between toe-off and heel strike on the same leg.
- Swing time - Time the foot is in the air (begins at toe-off and ends at heel-strike). The swing time occupies 39% of the gait cycle on average.
- Stance time - Time the foot is in contact with the ground (begins at heel-strike and ends at toe-off). The stance time occupies 62% of the gait cycle on average.

- Swing/stance ratio - Ratio of average swing time to average stance time.
- Double support - Time that both feet are on the ground (begins when a heel-strike occurs on the foot in question and ends when a toe-off occurs on the opposite foot).
- Single support - Time that one foot is on the ground (same as swing time of the opposite leg).
- Double/single support ratio - Ratio of average double support time to average single support time.

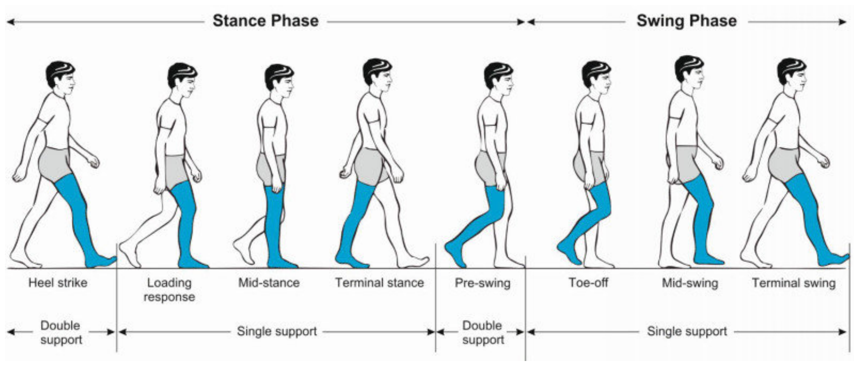

Figure 1.2: Phases of the human gait cycle and important gait events (from [12]).

More advanced gait analysis and biomechanics focuses on Kinetics and Kinematics. Kinematics is primarily concerned with joint positions (measured in metres (m)), velocities (measured in metres per second (m/s)) and accelerations (measured in metres per second squared (m/s^2)) [71]. Joint rotations, rotational velocities and rotational accelerations can also be calculated using the positional information and inverse kinematics [72]. Kinematics measurements are traditionally taken using marker-based motion capture systems (detailed later in this section). The most common measures of kinematics used by clinicians are joint rotations, which usually fall under one of five categories (Fig.1.3):

- Flexion - a decrease of the angle between two connected segments (a segment is a section of the body (i.e. the forearm)) [71].
- Extension - an increase of the angle between two connected segments [71].
- Abduction - a movement away from the centre of the body or the centre of a segment [71].
- Adduction - a movement towards the centre of the body or the centre of a segment [71].
- Rotation - a movement about the axis running through a segment (can be internal or external [71].

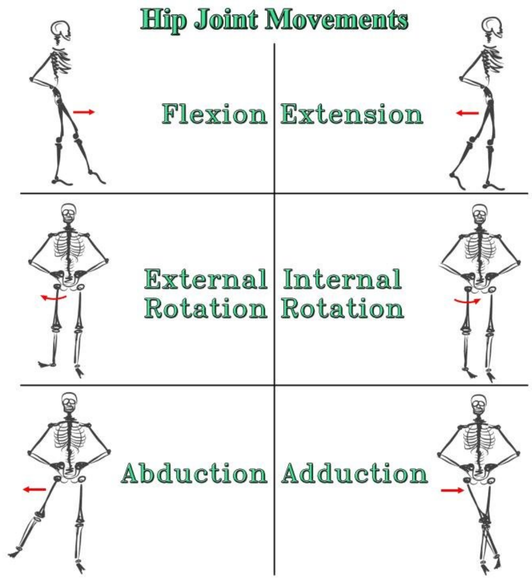

Figure 1.3: The five common types of joint rotation (from [13]).

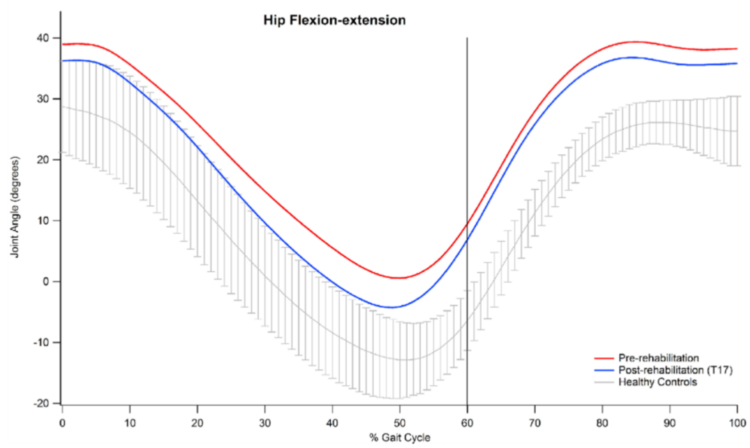

Figure 1.4: Graph of average hip flexion/extension over the gait cycle (from [14]).

Kinetics focuses on the forces applied to the skeletal system by muscles [71]. There are two types of kinetics: Linear Kinetics and Angular Kinetics. Linear kinetics focuses on force (expressed in Newtons) and the direction it is acting in. Linear kinetics during gait is usually measured using force plates (fig.1.5). The force plate measures the direction and intensity of the force using piezoelectric sensors under the plate. Meanwhile, angular kinetics focuses on torque (or moments) about a joint (expressed in Newton-metres (Nm)) [71]. Angular Kinetics information is usually derived using inverse dynamics [73].

Figure 1.5: AMTI force plates used to measure kinetics [15].

Clinicians determine gait metrics using one of two methods: observational analysis or technology-aided analysis. A clinician performs observational gait analysis without a computer (Fig.1.6). During an assessment, the patient walks in front of the clinician on a treadmill or a large section of the floor. The clinician observes irregularities in the patient's movement, often using a gait scoring system as an impartial assessment method. There are many scoring systems for different diseases and movement disorders. Some notable examples include:

- GALS (Gait, Arms, Legs, Spine) system for detecting MSK disorders [74].
  - The gait assessment section of the GALS system assesses gait symmetry, gait smoothness, gait quality across the gait cycle, stride length, and the ability to rapidly change position. The clinician assesses the patient's gait by asking them to walk a few steps, turn around and walk back [74]. GALS also assesses any potential causes of abnormal gait such as structural deformities or swelling [74].
- Tinetti test for gait and balance in older adults [75].
  - The gait assessment section of the Tinetti test assesses hesitancy in starting walking, gait symmetry, step length, step height, step continuity, walking path, walking time, and the straightness of the trunk. The clinician assesses the patient's gait by asking them to walk a few metres at a standard walking speed, turn around and walk faster on the way back.[75].
- Get up and go test for balance in older adults [76].
  - The gait assessment section of the get up and go test measures postural stability, gait, stride length, and sway [76]. The clinician assesses the patient's gait by asking them to get up from a seated position in the chair, walk 3 metres at a normal pace, turn around, walk back at a normal pace, and sit back down.

Figure 1.6: Observational gait analysis (from [16]).

Technology-aided gait analysis (Fig.1.7) can take many forms, but all share a general method:

1. Sensor - captures the patient's movement in a form that the analysis algorithm can process.

2. Additional input (optional) - in some analysis methods, the software requires clinicians to manually label parts of the data to assist the analysis algorithm.

3. Analysis algorithm - takes in the sensor data and any additional input, processing it, and outputting the required gait metrics.

Figure 1.7: Technology-aided gait analysis (from [17]).

There are different categories of technology-aided gait analysis:

- **Marker-based systems** involve the clinician placing markers on the patient. These markers track the patient's joints and calculate the specified gait metrics. Marker-based systems are extremely accurate and time-efficient, but are costly as a result.

- **Markerless systems** use pose-estimation algorithms to determine joint positions from RGB or RGB-D images and calculate relevant gait metrics. These systems are usually less accurate than marker-based systems, but are still very time-efficient and more cost-effective than marker-based systems.

- **Manual marker placement systems** prompt the clinician to label joint positions frame-by-frame and calculate relevant gait metrics using these positions. These systems have around the same accuracy and price as markerless systems but are less time-efficient.

Technology-aided gait analysis is much more accurate and time-efficient than observational gait analysis. This increases the likelihood that medical clinicians can detect the gait disorder at an early stage.

1.3 Physiotherapy as a Treatment

There are many treatments for gait and movement disorders ranging from different types of medication to surgery. One more holistic option is physiotherapy. Physiotherapy helps to restore movement and function [77] and is a treatment option for most gait disorders. Unlike other treatment options, physiotherapy is advantageous, as can have less side effects than other treatment options.

There are two types of physiotherapy: passive and active. Passive physiotherapy is performed by the medical clinician and does not require the patient's muscles to be active [78]. It is usually used where the patient's pain is too extreme for them to participate in active physiotherapy [78]. Acupuncture and manual therapy (manipulation of the muscles with the hands) are the main two forms of passive physiotherapy [77]. Active physiotherapy is where the patient performs exercises to increase their strength and mobility[77]. The patient usually performs these exercises free, with weights, or with resistance bands. Active physiotherapy for gait disorders often uses treadmill walking to build strength and improve physical capacity.

More recently, researchers have used virtual reality (VR) treadmill walking to administer active physiotherapy. VR is a method of displaying animated 3D content that is more immersive than 2D displays (Fig.1.8). According to Britannica, VR is "the use of computer modelling and simulation that enables a person to interact with an artificial three-dimensional (3-D) visual or other sensory environment" [79].

Figure 1.8: Screen showing the user's in-game view during a VR game (from [18]).

VR headsets are the most common method of displaying VR content. A VR headset has one display for each eye, giving the user the illusion of being immersed in a virtual 3D environment. Here some of the most popular VR headsets currently on the market:

- **VIVE pro (Fig.1.9b)** - Developed by the HTC Corporation. It runs by connecting to a PC and costs £1,299.00 [80].

- **Meta Quest 2 (Fig.1.9a)** - Owned by the company Meta. It costs £399.00 and does not need to be connected to a PC [81].
- **Valve Index (Fig.1.9c)** - Developed by the online games platform Steam. Similar to the VIVE pro, the Valve Index requires a PC to work. It costs £459.00 [82].
- **PlayStation VR (Fig.1.9d)** - Developed by the games console company PlayStation. It requires a PlayStation 5 console to work and costs £299.00 [83].

(a) Meta Quest 2 (from [81]). (b) HTC VIVE (from [80]). (c) Valve Index (from [82]). (d) PlayStation VR (from [83]).

Figure 1.9: Some of the most well-known VR devices.

Although VR treadmill walking improves physiotherapy outcomes for certain conditions [5] [6] [7], it is only accessible to a small fraction of the population. This is due to the high cost of motion capture equipment and the need for specialist facilities. This thesis will aim to address this issue by providing a portable and cost-effective version of the protocol.

2 Proposed Solution

2.1 Solution Outline

This thesis describes the implementation of a gait analysis and physiotherapy application. The application functions as a web-based first-response tool that is more accessible than current diagnosis and treatment solutions and provides a reliable quantitative diagnosis, unlike the scoring systems described in section 1.2 which use qualitative observations. The application performs physiotherapy within a VR world, as VR is significantly better at improving gait quality [84] and pain [85] when compared with traditional physiotherapy approaches. The main priorities for development were:

1. Cost-effectiveness.
2. Time efficiency.
3. Accuracy.

Prioritizing cost effectiveness whilst still maintaining an acceptable level of accuracy allowed a wide selection of clinics with limited facilities to use the product.

With these priorities in mind, it was decided a markerless motion capture (MoCap) system would be used for the web application. The technology-aided nature of this solution would provide higher accuracy than observational analysis and the markerless format would make the system more cost effective and time efficient than other technology-aided solutions.

2.2 Gait Analysis Protocol

The proposed solution uses a markerless motion capture system to analyse the subject's gait. To understand markerless systems, it is important to have a basic grasp of convolutional neural networks. Convolutional neural networks (CNNs) are one of the most common methods of estimating pose. According to Encyclopedia Britannica, a neural network is "a computer program that operates in a manner inspired by the natural neural network in the brain" [86]. A neural network's output is called an inference. Inside a neural network are layers of linked mathematical operators. Each mathematical operator has a weight which dictates how much it factors into the network's inference. The network improves its weights by performing inference on training data and calculating the error (loss) compared to the correct answer.

A CNN is a neural network often tasked with solving computer vision problems. A typical CNN has convolutional layers, pooling layers, and fully connected layers. A convolutional layer transforms the image by convolving it with a kernel (transformation matrix). The convolution process involves stepping (or shifting) the kernel across the image pixels, multiplying the selected pixels by the kernel each time (Fig.2.1) and producing an output matrix. Convolutions can pick particular features out of an image which is why they are popular for computer vision tasks.

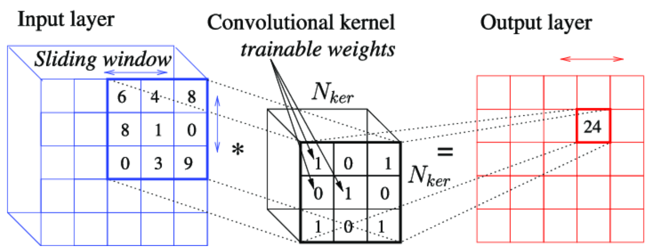

Figure 2.1: Convolving an input matrix with a kernel to produce an output matrix (from [19]).

A pooling layer uses an algorithm to reduce the size of an image whilst trying to preserve as much information as possible (Fig.2.2). This method can reduce the computing power required in subsequent network layers. An example of a pooling algorithm is max pooling (Fig.2.2). Max pooling takes the maximum values of small regions of the image and uses them to construct a smaller image. A similar example is average pooling (Fig.2.2), where the new image is composed of the mean values of each region.

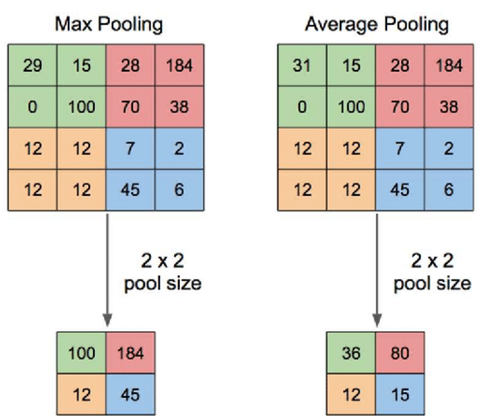

Figure 2.2: Two common pooling algorithms: max pooling and average pooling (from [20]). In both cases, the image is split into four 2x2 regions which are then condensed into four 1x1 regions.

The fully connected (FC) layers are at the end of a convolutional neural network. FC layers have connections to every node of the previous layer (Fig.2.3) and are used to formulate an inference based on the convolution and pooling in previous layers.

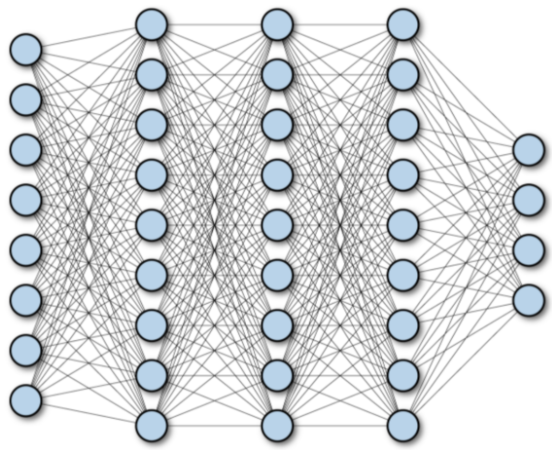

Figure 2.3: A fully connected neural network layer (from [21]).

In the context of markerless systems, CNNs are used to perform the task of pose estimation. Pose estimation detects the location of a person's joint positions from RGB (or RBG-D) video (Fig.2.4). Pose estimators usually predict joint positions using CNNs and express them as 2D or 3D Cartesian coordinates. Some examples of pose estimation technology are described in more detail below.

Figure 2.4: 2D pose estimation of a single RGB frame (from [22]).

OpenPose [23] is a pose estimator developed by the Carnegie Mellon University's Perceptual Computing Lab. It detects 2D joint locations of multiple subjects from a single RGB image (Fig.2.5) in real-time. OpenPose achieves joint prediction using two stages, both performed by a CNN. First, the CNN creates a confidence map for each joint. The confidence map shows the most likely positions for the joint in a frame. Second, OpenPose predicts part affinity fields (PAFs) for the subject's joints. PAFs are 2D vectors that describe the most likely orientation of a joint. OpenPose then combines these two sources of information and outputs 2D coordinates for each joint, each with a confidence value.

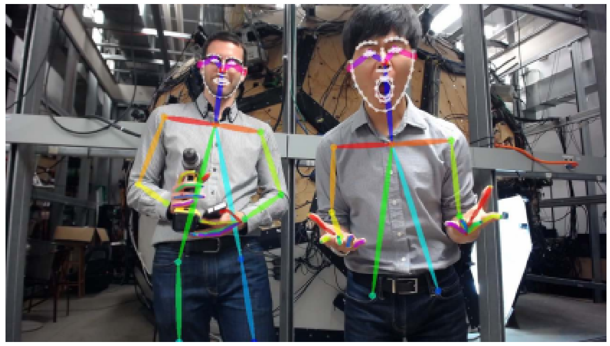

Figure 2.5: 2D pose estimation using OpenPose (from [23]).

MediaPipe [87] is a framework developed by Google to combine perception-based machine learning algorithms. There are several example projects within the MediaPipe page [88] including face mesh detection, object detection and iris detection. One MediaPipe example relevant to the field of pose estimation is MediaPipe Pose [89]. MediaPipe Pose is a pose estimator that uses the BlazePose CNN [90] to predict 3D joint positions from a single RGB image of the subject in real-time (Fig.2.6a). MediaPipe Pose provides the joint positions in a normalised format and in metres. MediaPipe Holistic [91] (Fig.2.6b) is another pose estimator from the MediaPipe project. It combines MediaPipe Pose with MediaPipe's hand and face keypoint detection algorithms, producing a more versatile pose estimator.

(a) Pose estimation with MediaPipe Pose (from [89]).

(b) Pose estimation with MediaPipe Holistic (from [91]).

Figure 2.6: MediaPipe Pose and MediaPipe Holistic.

The Azure Kinect [92] (Fig.2.7) is an RGB-D (colour and depth) machine learning solution developed by Microsoft. It performs 3D pose estimation and speech recognition in real-time. The pose estimation functionality is comprised of a software component - the Azure Kinect Sensor SDK, and a hardware component - the Azure Kinect RGB-D camera. The pose estimation component uses a CNN to detect joint positions in 2D and then uses the depth stream data to convert them into 3D positions [93]. The 3D pose data can be used to analyse movement or power 3D avatars within games [94].

Figure 2.7: Azure Kinect RGB-D camera (from [24]).

XNect is a pose estimator developed by the Max Planck Institute for Informatics, EFPL, Saarland Informatics Campus, and The University of British Colombia. It detects 3D joint locations from an RGB image in real-time using a CNN to detect 3D pose and skeleton fitting to ensure the body pose is valid when compared to a biomechanical model of the human body [25]. XNect is capable of detecting multiple people in a scene and can be used for 3D character control [25] (Fig.2.8).

Figure 2.8: Multi-person character control with XNect (from [25]).

DensePose is a pose estimator developed by Facebook Research. It calculates 2D UV co-ordinates for the body of each person in the scene. The UV co-ordinates form a map of the surface of the human body, allowing 3D meshes to be fitted to the 2D image (Fig.2.10) [26]. This method of pose recognition is referred to as dense, as it is not true 3D pose estimation. To detect the mesh co-ordinates, DensePose analyses each pixel of the image to determine which body part it belongs to and its precise location within that body part (Fig.2.9) [26].

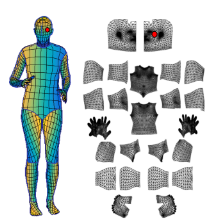

Figure 2.9: Detecting UV co-ordinates using DensePose. The red dot is the location of a pixel in the image on a mesh representation of the person's body (from [26]).

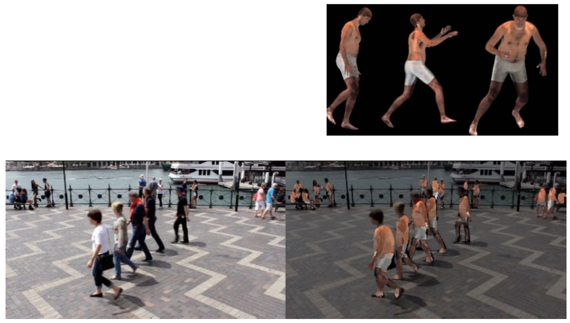

Figure 2.10: Fitting a 3D mesh to a 2D image using DensePose (from [26]).

Below is a table comparing the features of the markerless solutions this section has mentioned.

Pose Estimator	Input	2 or 3D?	Multi-person?	Web-compatible?	Price
OpenPose	RGB Video	2D	Yes	No	£0.00
MediaPipe Pose	RGB Video	3D	No	Yes	£0.00
MediaPipe Holistic	RGB Video	3D	No	Yes	£0.00
Azure Kinect	RGB-D Video	3D	Yes	No	£355.00
XNect	RGB Video	3D	Yes	No	£0.00
DensePose	RGB Video	2D (dense)	Yes	No	£0.00

Table 2.1: Table comparing markerless motion capture solutions

This thesis explores MediaPipe and OpenPose as options for motion capture. XNect was excluded because the code was not easily accessible and DensePose was excluded as it could not be implemented on a windows machine. The Azure Kinect was excluded, as it is expensive compared to the other options which would limit accessibility of the system.

2.3 Physiotherapy Protocol

The physiotherapy component of the system uses VR technology to deliver the treatment. Most VR physiotherapy options use some form of biofeedback as a method of improving gait.

According to Encyclopedia Britannica, biofeedback is “information supplied instantaneously about an individual’s own physiological processes” [95]. Some examples include:

- **Heart rate [95]** - a live heart rate display on a smart watch.
- **Brain signals [95]** - a live readout of brain signals from an electroencephalogram (EEG).
- **Blood pressure** - a blood pressure monitor [95].

Figure 2.11: Heart rate based biofeedback on the Apple Watch (from [27]).

The biofeedback paradigm in VR physiotherapy utilizes motion capture technology and a 3D virtual avatar to construct a third person view of the patient's pose in real-time (Fig.2.12). This method is proven to visualize gait accurately to facilitate training via biofeedback [28], and can successfully improve gait in patients with cerebral palsy [5], stroke [6], and patients with motor neglect (chronic pain) [7]. This thesis uses the VERMONT protocol for motor neglect conditions [7], as it utilises a standard display. This feature makes the protocol more cost-effective, easier to implement in smaller facilities, and more accessible.

Figure 2.12: Avatar-based biofeedback (from [28]).

2.4 System Tech Stack

The gait analysis and physiotherapy system is web-based and utilises the JavaScript front-end development stack. This is made up of three major components:

- **Hypertext Markup Language (HTML)** - Describes the web page's structure using tags. Tags must open and close (Fig.2.13).

- **Cascading Style Sheets (CSS)** - describes the webpage's style. CSS properties are attached to HTML elements.
- **JavaScript/Python** - instructs the website on what to do when the user performs different actions. This includes handling user interaction and updating CSS properties. This project will use JavaScript as its programming language.

```html
<body>
    <h1>Hello World!</h1>
</body>
```

Figure 2.13: An example of HTML tags.

The system uses Next.js for the UI. Next.js [96] is a JavaScript library that builds upon the commonly used JavaScript library React.js [97]. Web developers use React.js to create components that combine HTML, CSS and JavaScript. The current page can then use these components as HTML tags. Due to the JavaScript-based nature of these components, they can have states which can change during runtime. React.js components refresh each time any of their states change. Next.js builds upon and optimizes React and "handles the tooling and configuration needed" to build a React project [98].

The system's UI also uses Tailwind. Tailwind [99] is a framework that makes CSS more efficient. It shortens common CSS styles that usually require multiple properties into a single property that the developer can apply directly to the page's HTML. Common styles include flexboxes [100] and grid views [101].

Finally, to create the 3D biofeedback environment, the solution uses Three.js. Three.js is a 3D library for JavaScript [102]. It can be used to make 3D applications and games (Fig.2.14) and supports VR content in a web environment [34].

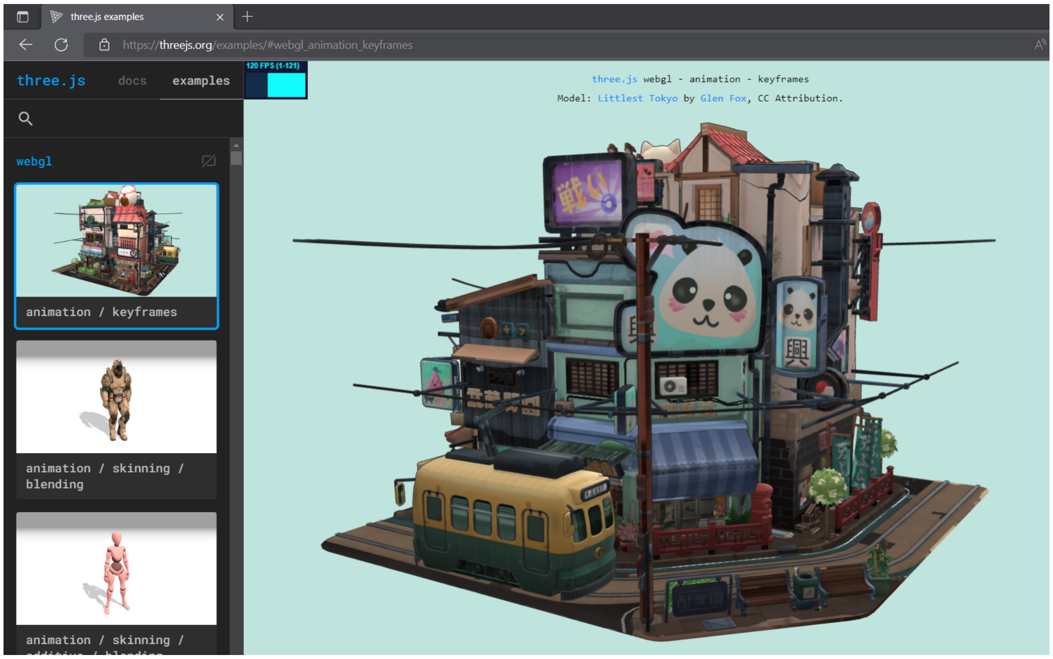

Figure 2.14: Viewing a 3D model in the browser with Three.js (from [29]).

Below is a diagram showing the full tech stack (Fig.2.15).

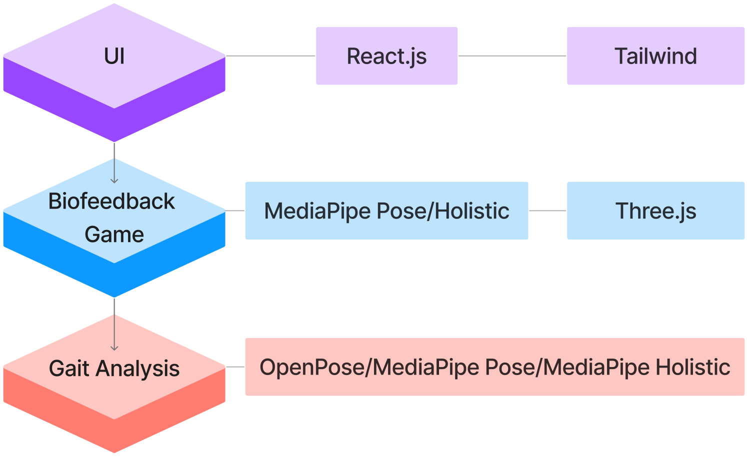

Figure 2.15: Diagram of the proposed tech stack.

2.5 Measuring The System's Accuracy

To validate the system's effectiveness as a gait analysis tool, its performance was compared to that of a gold standard. The gold standard in gait analysis currently involves systems that use retro-reflective markers such as OptiTrack [103] and Vicon [104], as the accuracy is high enough that the system is no longer a significant source of error in clinical gait analysis [105]. This thesis used the OptiTrack retro-reflective system [103], as it was readily available in the University of York's facilities. The OptiTrack system uses multiple infrared cameras to detect retro-reflective markers on the subject. The validation study used the 10 OptiTrack V:100 and V:100 R2 (Fig.2.16) infrared cameras available at the University of York and combined them with OptiTrack's tracking software "Tracking Tools" [106]. Both the V:100 and V:100 R2 cameras have sub-millimetre accuracy [107] [108].

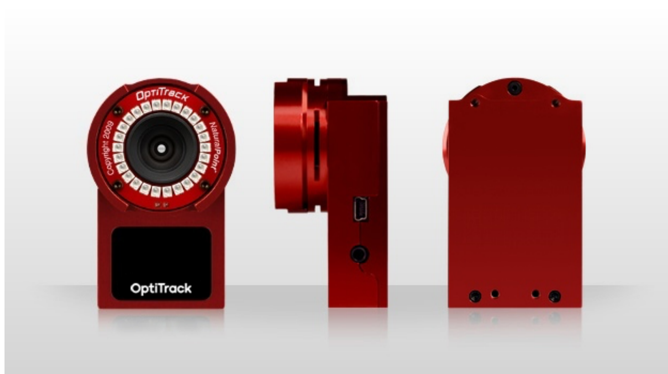

Figure 2.16: OptiTrack V100:R2 camera (from [30]) (the V100 looks almost identical to the V100:R2).

The validation study captured simultaneous pose data from six participants (referred to from now on as "patients") for the OptiTrack and the web-based solution. Once the data was captured, the following gait metrics could be calculated:

- Hip flexion (degrees).
- Knee flexion (degrees).
- Stance/swing time(s).
- Double/single support time(s).
- Stride length (m).
- Distance walked (m).
- Cadence (steps/s).

- Number of steps.

These metrics were chosen from experience, as they are some of the most commonly assessed by clinicians.

The study compared the average gait metrics from each data collection, assessing the similarity of the web-based solution's results to those of the gold standard (the OptiTrack).

The data collection protocol for the validation study is detailed below. For a full description of the protocol, see the patient consent form in appendix D.1. Here are the key points:

- The participant will have markers placed on their hips, knees and ankles.
- They will then stand stationary on the treadmill for the calibration stage.
- After the calibration stage, the participant will walk for nine "rounds", each a minute in length.
- They will walk at three different speeds (three rounds for each speed) - 0.5, 1 and 1.5 (speeds are arbitrary units defined by the treadmill).

A mobile phone captured the video footage for the pose estimator whilst the OptiTrack V100 and V100:R2 cameras captured the gold standard footage using Tracking Tools [106]. The command "3, 2, 1, start" was issued at the beginning of the recording, starting the OptiTrack recording on the word start and stopping at 60 seconds by saying "3, 2, 1, stop" (stopping on "stop"). The speech acted as a timecode so the recordings could be synchronised.

2.6 Project Management Strategy

This thesis used a project management strategy to track how close each of the application's features were to completion. There are two types of project management approach that software engineers use:

- **Waterfall management** - this strategy plans all the project's deliverables before development work commences.
- **Agile management** - this strategy is composed of sprints, where the developers work on a particular set of features. Planning takes place before each sprint, and the learnings from the previous sprint are taken into account when planning the next one.

This thesis used a variation of the Agile management strategy called Scrum [31]. Agile was chosen over Waterfall, as it is more adaptable to changes in knowledge. This flexibility was advantageous, as during the planning phase the pose estimation method was not yet determined and could have changed multiple times throughout the development process.

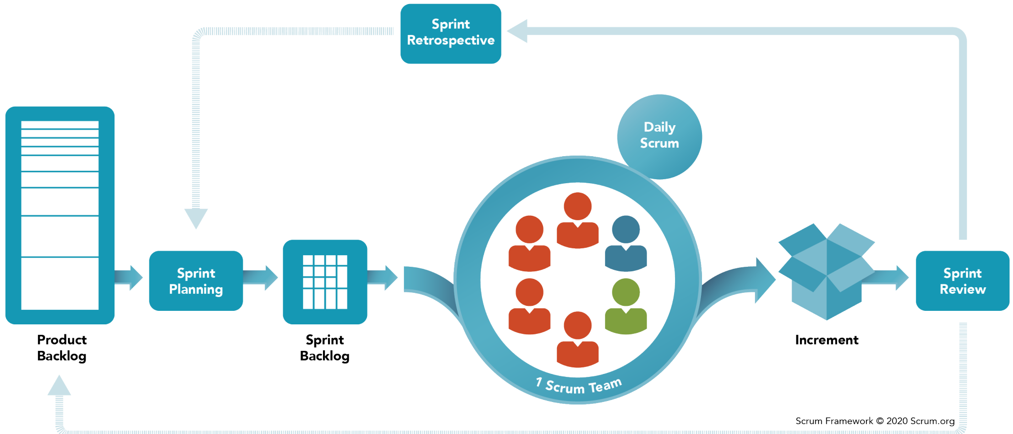

Figure 2.17: The basic concepts of scrum (from [31]).

Each scrum sprint was equivalent to a version of the software (V1.0, V2.0, V3.0). Jira [109], a scrum planning tool, was used to assign deliverables to each sprint. By tracking how many deliverables were completed, Jira could be used to estimate how close each sprint was to completion (Fig. 2.18).

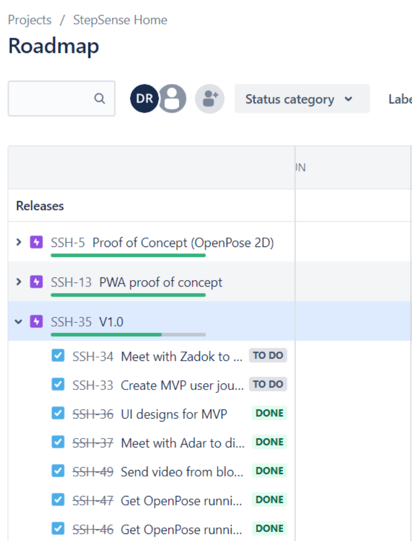

Figure 2.18: Jira allows progress tracking for Scrum sprints.

Each of the deliverables within the Jira project had a series of sub-deliverables that followed the SMART goal-setting strategy:

- **S** - Specific.
- **M** - Measurable.
- **A** - Achievable.

- **R** - Realistic.
- **T** - Time-bound.

For example, one could turn the non-smart goal "Make some of the UI" into a smart one by changing it to "Implement the web app's welcome page as shown in the UI designs within the next two days." Assigning portions of the project calendar to each smart goal ensured they fulfiled the time-bound aspect of the SMART strategy.

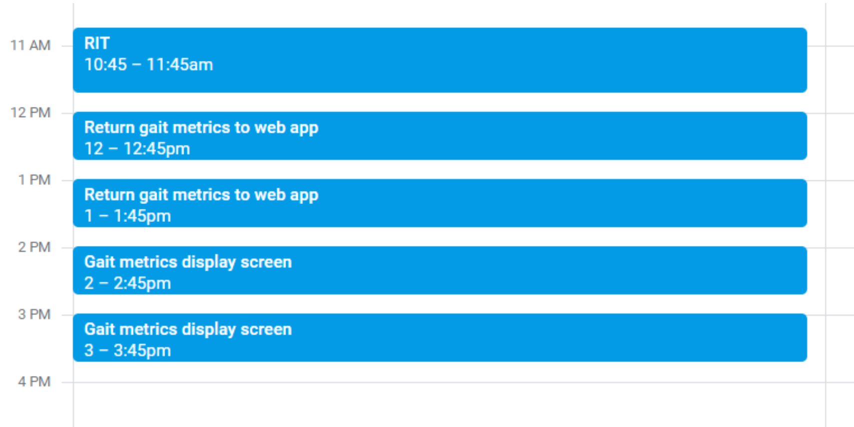

Figure 2.19: Assigning sub-tasks as time-limited calendar items.

2.7 Aims and Objectives

Detailed below are the aims and objectives for the research work. The aims were more general and described what the research work needed to accomplish, whereas the objectives were more specific and explained how the research aims would be completed. The aims and objectives were categorised into "diagnosis" - testing for abnormal gait, and "treatment" - rehabilitating abnormal gait.

Project aims:

- Diagnosis:
  - **Aim 1** - Create a web application that analyses a subject's gait.
  - **Aim 2** - Validate the accuracy of the gait analysis technique.
- Treatment:
  - **Aim 3** - Develop a web-based biofeedback game for gait rehabilitation.

Project objectives:

- Diagnosis:

- **Objective 1** - Implement a pose estimator that calculates the subject's joint locations.
- **Objective 2** - Derive stride length, swing/stance, double/single support, speed, cadence, hip angles, knee angles, and distance walked from the joint locations.
- **Objective 3** - Use OptiTrack motion capture software and hardware to validate the above gait metrics on five test subjects.

- Treatment:
  - **Objective 4** - Use a real-time pose estimator in combination with Three.js to create a biofeedback game with a 3D avatar.

3 Design

3.1 Version 1.0 Designs

Version 1.0 of the web app will implement the core functionality needed for web-based gait analysis. It will achieve this by analysing user-recorded footage using a server-side OpenPose instance. The following user requirements were derived from this brief:

1. As a clinician, I must be able to…
   (a) Select to perform a gait analysis (assessment) session.
   (b) Receive information about how to set up the device camera for capture.
   (c) Start an assessment session.
   (d) Record a 30 second video of the patient.
   (e) Send the video to the OpenPose server for analysis.
   (f) Begin a new session.

A list of deliverables and sub-deliverables was then developed from the user requirements:

1. User interface:
   (a) Pre-assessment page with "start assessment" button.
   (b) Assessment instructions page with "record" button
   (c) Assessment page with camera view.
   (d) Post-assessment page with "new capture" button.
2. Video recording:
   (a) Record a 30 second video using the device's webcam and save it in temporary memory.
   (b) Upload the video to the OpenPose server.
3. Pose estimation:
   (a) Use OpenPose to detect the patient's joint locations.
   (b) Calculate gait metrics from the OpenPose data (on the server side).

To design a user interface (UI) in line with the requirements above, a flow diagram was created to understand the structure of the main pages in the UI and describe how the user would navigate through them.

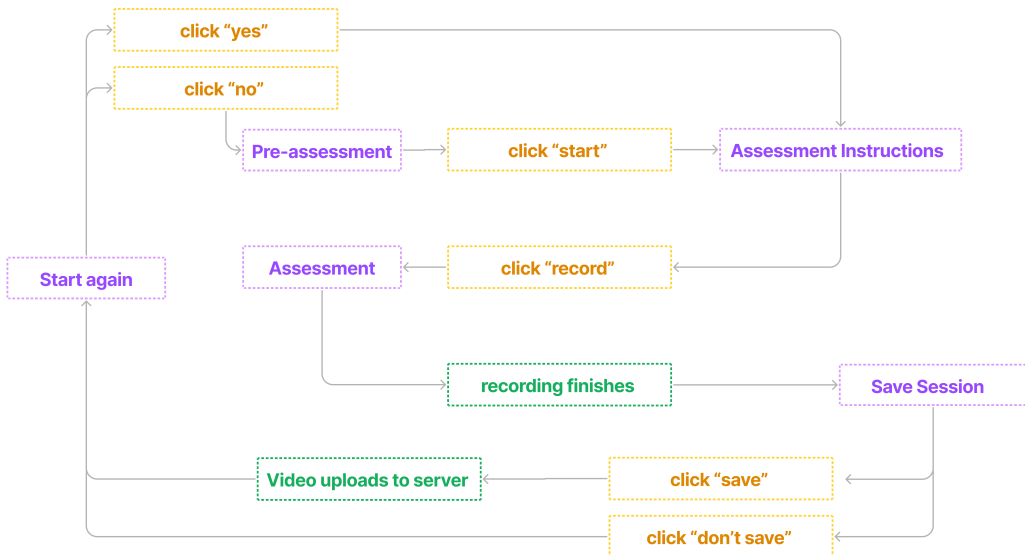

Figure 3.1: Flow diagram for V1.0 UI.

Each UI screen in the flow diagram was then designed. Three primary UI colours were chosen (names given in hex): #135796 (blue), #FFFFFF (white), and #FFDC64 (yellow). These colours were picked as they are contrasting, thus making the page more accessible to colour-blind users.

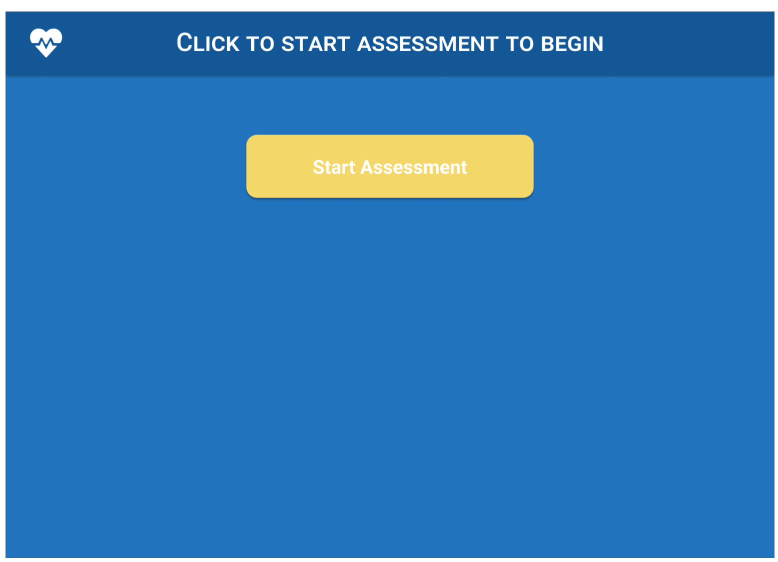

Figure 3.2: Pre-assessment screen (heartbeat logo from [9]).

The pre-assessment screen was designed (Fig.3.2) with later versions of the web app in mind. Space was left below the start assessment button so other buttons, such as “start biofeedback game”, could be added when their functionality had been implemented.

Figure 3.3: Assessment instructions screen (Background image from [32]).

When the user clicks the "start assessment" button on the pre-assessment screen, the web app navigates to the assessment instructions screen. The assessment instructions screen (Fig.3.3) instructs the clinician on how to set up their camera to record a gait assessment. The design features a live camera feed to allow the clinician to adjust the camera position and angle.

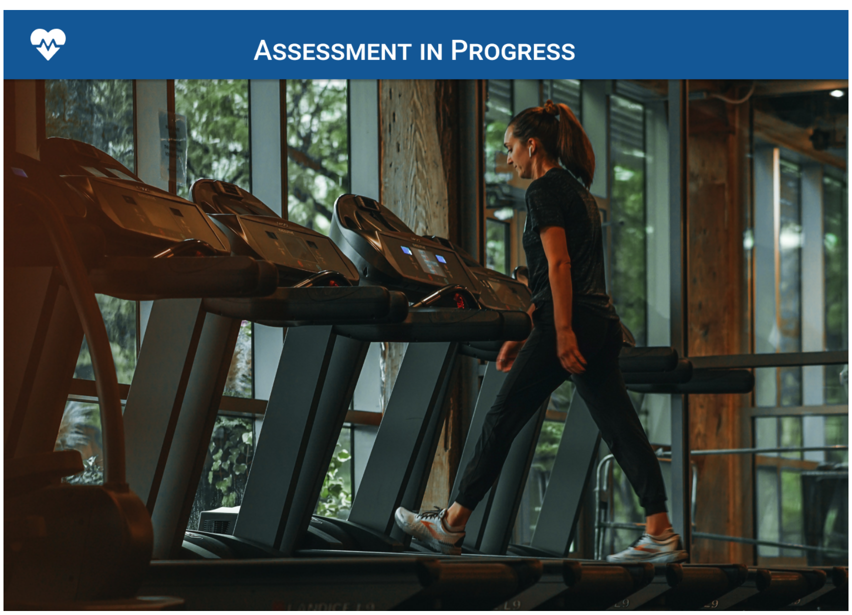

Figure 3.4: Assessment screen (Background image from [32]).

When the user clicks the "start" button on the assessment instructions screen, the web app navigates to the assessment screen. The assessment will record video footage for 30 seconds, displaying the save session screen at the end of the recording period.

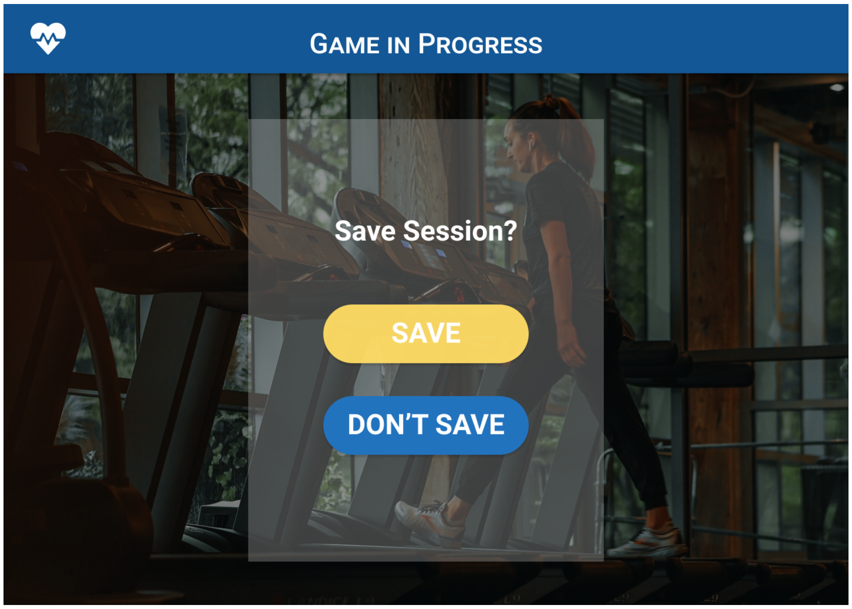

Figure 3.5: Save Session Screen (Background image from [32]).

The save session screen prompts the user to send the session for analysis by clicking "save". If the user clicks save, the web app uploads the recorded video to the OpenPose server and navigates to the start again screen. If the user clicks "don't save", the web app does not upload the video and displays the "start again" screen.

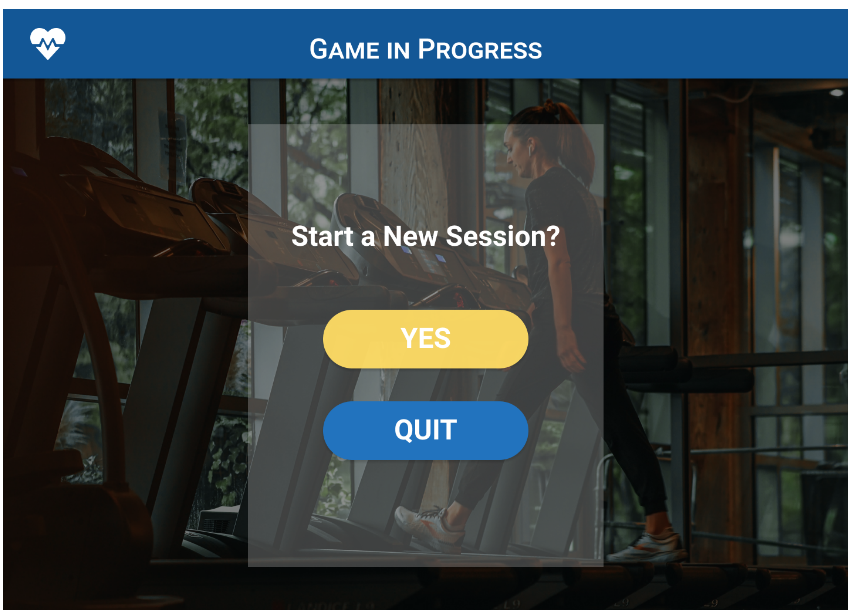

Figure 3.6: Start again screen (Background image from [32]).

On the start new session screen, the user can choose between starting a new session ("yes") and quitting for now ("quit"). If the user clicks "yes", the web app navigates to the assessment instructions screen. If the user clicks "quit", the web app navigates to the pre-assessment screen.

Upon completing the list of requirements and the UI designs, the next step was to specify the individual JavaScript classes and Next.js components that needed to be implemented. This information could then be used to form the tasks and subtasks for the Jira and begin version 1.0's implementation phase.

Five separate flow diagrams were created (see Appendix A.1). Each flow diagram contained the components and classes that needed to be developed by the end of the implementation phase. In the flow diagrams, purple represents a Next.js page, orange represents a Next.js component, and blue represents a JavaScript class.

3.2 Version 2.0 Designs

In version 2.0 of the web app, live body tracking would be used to drive a 3D avatar for patient-facing biofeedback. MediaPipe Holistic was chosen to drive the biofeedback avatar, as in preliminary tests, it ran in real-time on a laptop and a mobile phone. OpenPose would still be used for a post-session gait assessment as it was more accurate. This change in design meant the web app required two camera feeds: a front-facing feed for biofeedback and a side-facing feed for gait assessment. Another goal for version 2.0 was to make it mobile-friendly, which meant redesigning the UI to work on smaller screen sizes.

The first stage of design, involved creating the user stories. As this version of the app included patient-facing content, the user stories were written from the perspective of the patient as well as the clinician:

1. As a clinician, I must be able to...
    (a) Select to perform a biofeedback (training) session.
    (b) Receive information about setup for the training session.
    (c) Start the training session.
    (d) Send the video of the training session to the OpenPose server for analysis.
    (e) Receive gait metrics from the analysis server.
    (f) Start a new training session.
2. As a patient, I must be able to...
    (a) Walk forward in a virtual biofeedback environment as a stick man.

Using these user stories, a list of requirements was formulated. These requirements needed to be met by the end of version 2.0's implementation phase:

1. User interface:
    (a) Initial page with "start training" and "start assessment" buttons (remove start assessment page from V1.0).

(b) Training instructions page with "record" button.

(c) Select camera page allowing the user to select gait analysis and biofeedback cameras.

(d) Training page with camera view and biofeedback view.

(e) Post-training page with "new training session" button.

2. Pose estimation:

   (a) Implement MediaPipe Holistic tracking within the project.

   (b) Create a universal sensor framework that can be extended to multiple pose estimators.

3. Biofeedback Game:

   (a) Avatar class that instantiates a 3D stickman controlled by the chosen pose estimator at a specific location.

   (b) 3D walking game that allows the patient to walk around a virtual space with the avatar.

As in version 1.0, a flow diagram was created to understand the main pages in the UI and how the user would navigate around them.

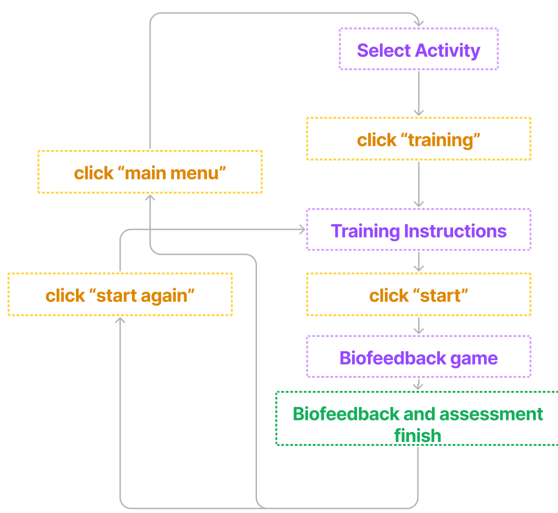

Figure 3.7: Flow diagram for V2.0 UI.

There were two new UI screens that needed to be designed, the first of which was the assessment screen (Fig.3.8). From now on, this new screen will be referred to as the training screen. The screen was split into three sections:

- **Camera feed** - displays the camera feed the device will use for biofeedback. The clinician can choose to make this full-screen.
- **Game feed** - displays the biofeedback game. The clinician can also make this section full-screen.
- **Control panel** - contains the start button for the game. In future versions of the app, the control panel would also contain settings for the biofeedback game.

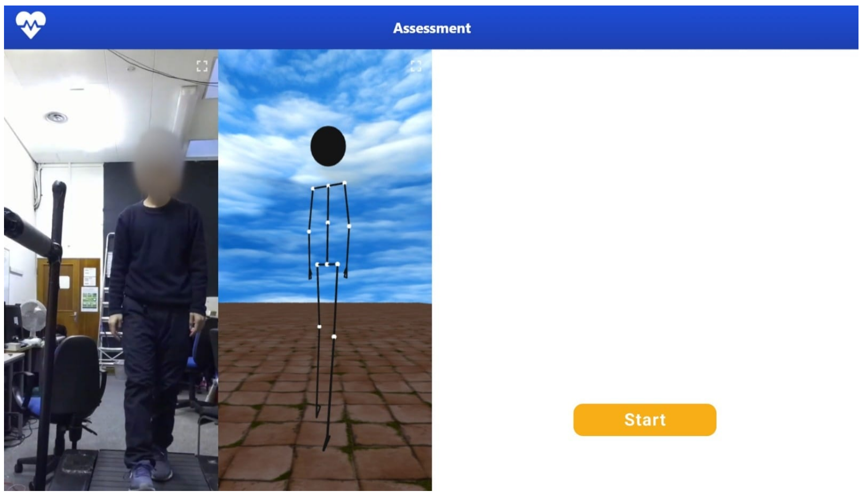

Figure 3.8: UI design for the training page.

The other new screen was the select camera screen. This screen would allow the user to select and position the cameras for biofeedback and gait analysis using two dropdowns and live camera feeds.

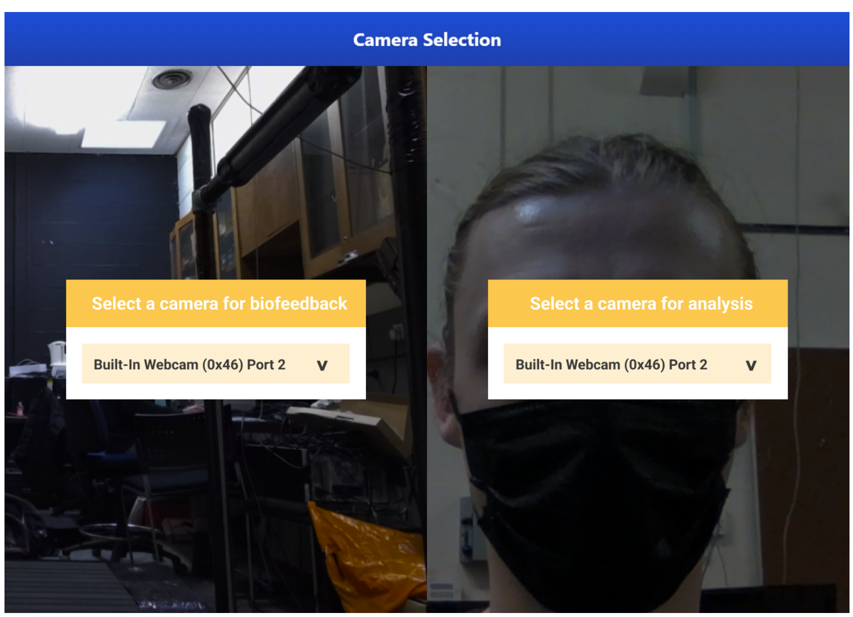

Figure 3.9: New UI design for the select camera page.

In addition to these new screens, some screens needed to be adjusted or copied from version 1.0:

- Rename the pre-assessment screen to “select activity”.
- Add a button to the select activity screen that starts a training session.
- Update the assessment instructions page to give training instructions.
- Duplicate the start again page and adapt it for training sessions.

Finally, mobile designs were created for the select camera screen and the training screen (appendix A.2), as the existing desktop layout did not work well within a mobile environment. Below are a few of the design choices made for the mobile versions of these two screens (images in appendix):

- Laying out content in a column-based view to suit the portrait nature of the phone display (Fig.A.6).
- Using a hamburger menu to hide and show the biofeedback control panel (Fig.A.7 and Fig.A.8).
- Using a button to toggle between the camera feed and the game feed (Fig.A.7 and Fig.A.8).

Flow diagrams were created for new or updated components and classes. These are included in Appendix A.3.

3.3 Version 3.0 Designs

Version 3.0, would implement game customization features. It would also build upon the following advancements in knowledge made in version 2.0:

- The OpenPose server is potentially costly.
- MediaPipe Pose is more accurate than MediaPipe Holistic.
- MediaPipe Pose might provide accurate enough joint positions to calculate gait metrics without OpenPose.

The first step taken to design version 3.0 was forming user stories for the patient and clinician. The user stories were as follows:

1. As a clinician, I must be able to…
    (a) Change the game’s ground and sky textures.
    (b) Choose the colour of the avatar’s bones, joints and head.

(c) Select the session record time.
(d) Decide whether to use a stroop test (a stroop test is a method of adding cognitive load by asking the person to name the colour of a word).
(e) Select the word frequency for the stroop test.

2. As a patient, I must be able to...

   (a) See live gait metrics on the biofeedback screen.

Using these user stories, a list of requirements was formulated that needed to be met by the end of version 3.0's implementation phase:

1. User interface:

   (a) Game customization panel with "Avatar", "Environment" and "Interactions" settings.
   (b) Live metrics display within the biofeedback game.

2. Pose estimation:

   (a) Calculate and display real-time metrics using MediaPipe Pose.

3. Biofeedback Game:

   (a) Avatar can be customised.
   (b) Environment textures can be customised.
   (c) Game parameters can be customised.

In addition to this specification, the app would be designed to output a JSON of the MediaPipe Pose data that could be analysed within MATLAB. This JSON data would be used in the accuracy assessment part of the project (see section 6).

The flow diagram for the UI was the same as in version 2.0 so a new one did not need to be created.

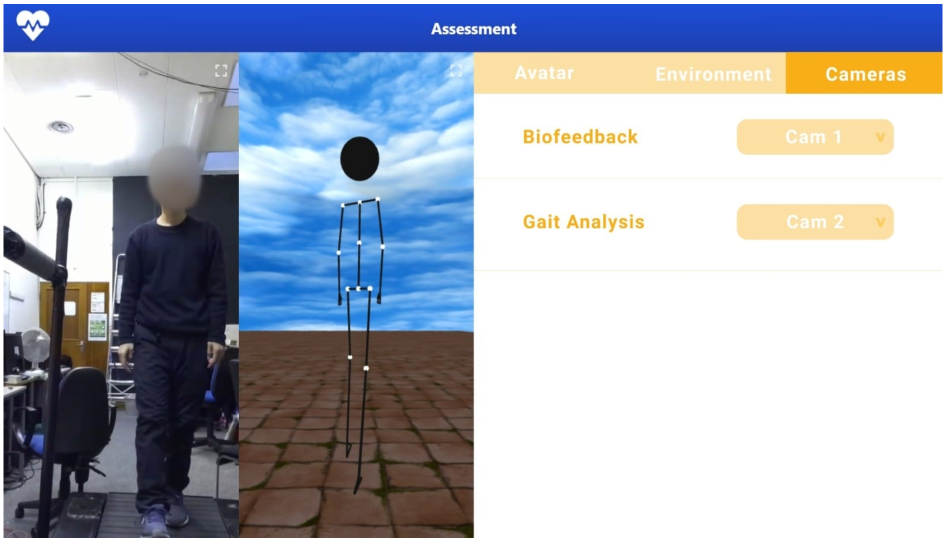

Figure 3.10: Training page with new control panel.

Flow diagrams were created for new or updated components and classes. These can be found in Appendix A.4.

4 Implementation

4.1 Version 1.0 Implementation

4.1.1 OpenPose Server Implementation

In version 1.0's implementation phase, the OpenPose server was implemented first. This task had three parts: implementing OpenPose locally, designing a Python server to run OpenPose remotely, and sending video data from the web application to this Python server.

In the local implementation stage, it was decided the easiest way of running OpenPose would be to use the portable windows binary and execute it using Python. The binary would output JSON files for each frame of video data. Following this, the Python script would analyse the patient's gait using the motion capture data in the JSON files. These stages rewrite and improve upon the approaches used in [9] and [10]. The code for these stages is included in appendix B.1.

The OpenPose binary runs using a command line instruction, executed using Python's **subprocess.run()** function (Fig.4.1). The "–write_json" and "–write_video" flags tell OpenPose to output a JSON for each video frame and a copy of the video with labelled joint points.

```
# runs openpose on a video, outputting the analysed video and the JSON files
def run_openpose(recording_name):
    # run OpenPose on the video and save the result
    subprocess.run(['./bin/OpenPoseDemo.exe','--video', '../videos/' +
        recording_name + '.mp4','--write_json', '../videos-analyzed/' +
        recording_name,'--display', '0', '--write_video','../videos-analyzed/'
         + recording_name + '/' + recording_name + '.avi'])
```

Figure 4.1: Using the subprocess.run() command to analyse a video using OpenPose.

OpenPose is then instructed to output the JSON files for the video, so patient's gait can be analysed. The first step in the analysis process is searching the JSON data for heel-strike and toe-off events. Once these are located and labelled, the required gait metrics can be calculated.

These heel-strike and toe-off events are determined using ankle location (in the x-direction). Ankle location during gait is a periodic waveform (Fig.4.2), where the minima are heel strike events, and the maxima are toe-off events.

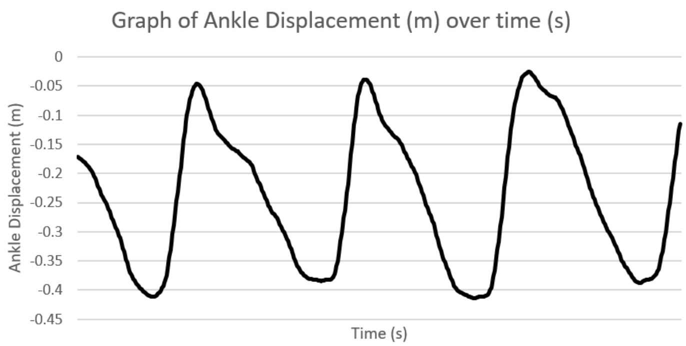

Figure 4.2: Graph of ankle displacement (m) over time (s).

To locate minima and maxima the data is iterated through chronologically and conditional statements detect points where the ankle changes direction. The following criteria are used to detect heel-strike events:

- The last toe-off event was on the leg in question.
- The last heel strike event was on the opposite leg.
- The difference between the hip and leg must be greater than it was in the previous iteration.

The code implementing these criteria is as follows:

```
# left heel strike
if l_hip_dif > l_hip_dif_prev and (gait_trackers['toe_off'] == 'none' or gait_trackers['toe_off'] == 'left') and (gait_trackers['heel_strike'] == 'none' or gait_trackers['heel_strike'] == 'right') and l_heel_x > r_heel_x and l_heel_y > r_heel_y:
        handle_heel_strike(metrics, gait_trackers, frame, l_heel_x, r_heel_x, 'left')
```

Figure 4.3: Detecting a left heel strike using the heel strike criteria.

The following criteria are used to detect toe-off events:

- The last heel-strike event was on the opposite leg.
- The last toe-off event was on the opposite leg.
- The difference between the hip and leg must be smaller than it was in the previous iteration.

The code implementing these criteria is as follows:

```python
    # left heel raise
    elif l_hip_dif < l_hip_dif_prev and (gait_trackers['toe_off'] == 'none' or gait_trackers['toe_off'] == 'right') and gait_trackers['heel_strike'] == 'right' and l_heel_x < r_heel_x and l_heel_y < r_heel_y:
        handle_toe_off(metrics, gait_trackers, frame, 'left')
```

Figure 4.4: Detecting a left toe-off using the toe-off criteria.

Once the heel-strike and toe-off events have been detected, the metrics for each gait cycle can be calculated. The following symbols will be used to describe the equations for each metric:

- H_c: Last recorded heel strike event.
- H_p: Heel strike event directly before H_c.
- H_{pp}: Heel strike event directly before H_p.
- T_c: Last recorded toe-off event.
- T_p: Toe-off event directly before T_c.
- G: Scaling to convert OpenPose co-ordinates to metres.

A subscript “x” is written after the event to denote spatial displacement on the x-axis (e.g. H_{cx}), otherwise, the symbol references time (e.g. H_c). The equations for calculating the metrics are below. All equations assume a heel strike has just occurred on the leg in question.

- Stride length $= G(H_{cx} - T_{cx})$.
- Swing time $= H_c - T_c$.
- Stance time $= T_c - H_{pp}$.
- Double support time $= T_c - H_p$.

The code for calculating these metrics also has to account for a scenario where the heel strike does not happen first. Therefore the code is slightly different to the formulas but still uses the same principles. For the full code see the **calculate_swing_stance_ds_ss()** function in B.1.

The Python script then derives the remaining metrics from the already calculated ones:

- Single support time is identical to the swing time of the opposite leg (see the **calculate_swing_stance_ds_ss()** function in appendix B.1).
- Number of steps is equal to the number of heel strike events (see the **handle_heel_strike()** function in appendix B.1).
- Cadence is equal to the number of steps divided by the video duration (see the **calculate_cadence()** function in Appendix B.1).

- Speed is equal to the sum of the stride lengths divided by the video duration (see the **calculate_speed()** function in appendix B.1).

After creating the gait analysis script, the next step was to develop the server code. The gait analysis script is hosted on a Flask [110] server. Flask [110] is an API that enables Python developers to create servers for their code. The first step to creating a flask server is initialising it on localhost:5000 (Fig.4.5).

```
app = Flask(__name__)

# run the Flask server
if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

Figure 4.5: Initialising the flask server and running it on localhost:5000.

vid_upload_and_analysis() (the main function for gait analysis) was then changed into an API route for the server using **@app.route()** from the Flask API (Fig.4.6). This means that whenever anything makes a http request to localhost:5000/vid-uploader, the script executes **vid_upload_and_analysis()**.

```
# saves video and analyzes with OpenPose
@app.route('/vid-uploader', methods=['POST'])
def vid_upload_and_analysis():
```

Figure 4.6: Using the **@app.route()** decorator to connect a function to an API call.

When the Flask API was complete, the next step was to send the server a video the web app had recorded. To send the video, a call is made to the Flask API from the web app. This is achieved using **XMLHttpRequest()** (Fig.4.7).

```
//Get the http request for the OpenPose server ready
var xmlhttp = new XMLHttpRequest();
```

Figure 4.7: Creating a new http request.

Once a HTTP request had been created, an action listener is instantiated to listen for the response. This listener navigates to the post-assessment page, appending the gait metrics received from the OpenPose server in JSON format to the URL (Fig.4.8) so the web app can access them.

```
xmlhttp.onreadystatechange = ()=>{
    if (xmlhttp.readyState == XMLHttpRequest.DONE) {
        const json = JSON.parse(xmlhttp.responseText)
        console.log(json);
        //Change to the post assessment screen
        router.push({pathname: '/assessment/post-assessment', query: json});
    }
}
```

Figure 4.8: Setting up the http request's event listener.

The final step in sending the video data is converting it to a blob (a blob "is a file-like object of immutable, raw data" [111]) and sending it to the server in the body of the http request (Fig.4.9).

```
xmlhttp.open("POST", 'http://localhost:5000/vid-uploader');

//Create blob with the video data
const blob = new Blob(recordedChunks);

//Save the blob to a form
var fd=new FormData();
fd.append("video", blob, "video.webm");

//Send the form to the OpenPose server using the http request
xmlhttp.send(fd);

//Go to the analyzing page for now
router.push('/assessment/assessment-analyzing');
```

Figure 4.9: Creating a new http request.

4.1.2 Web App UI

Each page in the web app UI is implemented as a Next.js functional component (see section 2.4). The key sections and features of the page are also represented as Next.js components. Most of the Next.js functional components are implemented using solely HTML and Tailwind (see section 2.4), but the **Webcam** and **WebcamRecorder** functional components have some additional functionality that will be explained in more detail below.

Webcam and **WebcamRecorder** both display live webcam footage in a **video** HTML element. **WebcamRecorder** also records the footage. The **WebcamRecorder** component (see appendix B.3) makes use of React's **useEffect()** hook. **useEffect()** is called after the functional components and HTML elements on the page are rendered. This hook allows **WebcamRecorder** to access the <**video**> element on the page after it is rendered so it can attach the webcam stream (Fig.4.11).

Within **useEffect()**, **WebcamRecorder** also calls **GetWebcamStream()**. This function creates a **MediaRecorder** (see Mozilla's MediaRecorder API [112]) to record the video frames, and when it finishes recording, it sends them to temporary storage (Fig.4.10). The **Webcam** component (see appendix B.2) is almost identical to **WebcamRecorder** but it does not need to set up a **MediaRecorder** (it does not need to record the webcam footage).

```
    //Once we have acquired the webcam, attatch it's stream to the video
      object
    navigator.mediaDevices.getUserMedia(vidProperties).then((stream)=>{
        video.current.srcObject = stream;
    });
```

Figure 4.10: Attaching the webcam stream to the <**video**> HTML object.

```
    //Create a media recorder to record the video
    const mediaRecorder = new MediaRecorder(stream,mediaRecorderOptions);

    //Create an array to save the video frames in
    const recordedChunks=[];

    mediaRecorder.ondataavailable=function (e){
      if (e.data.size > 0) {
        //Add image to array
        recordedChunks.push(e.data);
      }
      if (shouldStop === true && stopped === false) {
        mediaRecorder.stop();
        stopped = true;
      }
    };
```

Figure 4.11: Creating a new MediaRecorder and telling it how to save video data.

4.2 Version 2.0 Implementation

4.2.1 Patient Facing Biofeedback

In version 2.0, the patient-facing biofeedback was implemented first. The first step to this goal was implementing the real-time pose estimator (MediaPipe Holistic) in the web app.

Before implementing MediaPipe Holistic, a generic pose estimator class needed to be implemented (**PoseEstimator** (see appendix B.4)). This class allows fast integration of new pose estimators into the system when developers release them. The generic pose estimator class uses the joint points from the implemented pose estimator to render a 3D avatar within the biofeedback game's Three.js scene.

The pose estimator class has four main functions:

- **StartTracking()** - This function is abstract and specific to each pose estimator. **StartTracking()** should perform the necessary steps to set up the pose estimator and start it running on live video footage.
- **assignPose()** - This function is also abstract. The pose estimator should assign x, y, z and confidence values of each joint to a data structure inside **PoseEstimator** so the web app can access them.
- **buildAvatarBody()** - This function instantiates all the Three.js objects necessary to construct the avatar.
- **updateAvatarBody()** - This function updates the position, scale and rotation of all the avatar's body parts. It does this by calculating them using the x,y and z coordinates that **assignPose()** provides.

The functions above are explained in more detail below:
In **buildAvatarBody()**, there are three functions:

- **buildAvatarBones()** - Creates 25 cylinders as Three.js objects to represent the avatar's bones.
- **buildAvatarJoints()** - Creates 11 spheres as Three.js objects to represent the avatar's joints.
- **buildAvatarHead()** - Creates a sphere as a Three.js object to represent the avatar's head.

Instantiating Three.js objects involves creating a geometry and combining it with a material to form a mesh (Fig.4.12).

```
    //Make a bone (cylinder)
    const geometry = new THREE.CylinderGeometry( 0.003, 0.002, 20, 32 );
    const cylinder = new THREE.Mesh( geometry, this.boneMaterial );
```

Figure 4.12: Example of creating a mesh in Three.js.

In **updateAvatarBody()**, the distance the avatar has walked is calculated using **calculateDistanceWalked()** from **PoseEstimator**. To calculate this, the change in distance on the leg currently in stance phase is calculated and added to the total distance value. The code used for this is almost identical to the OpenPose gait analysis code in section 4.1.1. For the full code, see **calculateDistanceWalked()** in appendix B.4. After calculating the distance, the camera position is updated in the Three.js scene so it remains directly behind the avatar (Fig.4.13).

```
    //Update camera using Yoke
    this.camera.position.z = -0.7 + this.avatarDistance;
```

Figure 4.13: Updating the camera to sit directly behind the Three.js avatar.

Finally, **updateAvatarBody()** updates the spheres for the joints (including the head sphere) and the cylinders for bones to be the correct position, size and rotation. The rotation of each bone is calculated by determining its direction vector. This calculation involves finding the difference between the location vectors of the two joints it is attached to (Fig.4.14). The bone is then rotated to point towards the calculated direction vector (Fig.4.15).

```
    /**
     * Calculates the direction vector between two joints
     */
    calculateBoneDirection(v1, v2){
        const v3 = new THREE.Vector3();
        v3.copy(v1);
        v3.sub(v2);
        return v3;
    }
```

Figure 4.14: Finding the direction vector between the bone's two joint locations.

```
    //Calculate the bone's direction vector and use it to update its rotation
    const up = new THREE.Vector3(0,-1,0);
    this.avatarBones[index].quaternion.setFromUnitVectors(up, this.
        calculateBoneDirection(jointPair[0], jointPair[1]).normalize());
```

Figure 4.15: Rotating the bone to face the calculated direction vector.

To implement MediaPipe Holistic, **PoseEstimator**'s abstract methods **assignPose()** and **StartTracking()** are extended in a new class - **MediaPipeHolistic** (see appendix B.5).

In **StartTracking()** **MediaPipeHolistic** creates a new holistic solution from the MediaPipe package. This is done using MediaPipe Holistic's constructor function (Fig.4.16).

```
holistic = new MediaPipe.Holistic({locateFile: (file) => {
    return `https://cdn.jsdelivr.net/npm/@mediapipe/holistic/${file}`;
  }});

  //Set up holistic tracking
  holistic.setOptions({
    modelComplexity: 2,
    smoothLandmarks: true,
    smoothSegmentation: true,
    refineFaceLandmarks: true,
    minDetectionConfidence: 0.5,
    minTrackingConfidence: 0.5
  });
```

Figure 4.16: Creating a new holistic solution and setting the tracking options.

StartTracking() also initializes MediaPipe Holistic and tells it to call **getVideoFrame()** once initialization is complete. **getVideoFrame()** executes once per video frame sending the current frame of webcam footage to MediaPipe Holistic for analysis.

```
/* Gets current video frame and sends it for analysis */
async getVideoFrame(){
  window.requestAnimationFrame(()=> {this.getVideoFrame()});
  if(!this.switchingCams){
    await holistic.send({image:video});
    if(!this.isInitialized){
      this.isInitialized = true;
    }
  }
}
```

Figure 4.17: **getVideoFrame()** sends each frame of data to MediaPipe Holistic.

Finally, **StartTracking()** sets up an event listener that fires when MediaPipeHolistic returns pose data (Fig.4.18). This listener calls **assignPose()** to set the avatar's new pose and **updateAvatar()** (from the **pose-estimator** class) to match the avatar to the new pose data.

```
//Event fires when holistic has completed its analysis
holistic.onResults((results)=> {
  this.assignPose(results);
  this.updateAvatar();
})
```

Figure 4.18: Setting up the MediaPipe Holistic event listener.

assignPose(), adds the coordinates of each joint to an array that is visible to the **PoseEstimator** class. The distance walked is added to each joint's z coordinate to make the avatar appear to move forward with the user.

```
//Assign values for body
results.poseLandmarks.forEach(element => {
  this.body[i] = { coordinates: new THREE.Vector3(-element.x, -element.y, -element.z + this.avatarDistance), confidence: element.visibility}
  i++
});
```

Figure 4.19: Adding MediaPipe co-ordinates to an array.

4.2.2 Biofeedback Environment

The **World** class (see appendix B.6) renders an environment for the avatar to walk in. This environment includes a floor, a sky and lighting for the Three.js scene. The main functions in **World** are **addFloor()** and **addSky()**.

addFloor() creates a plane for the floor and applies a floor-like material to it. It then adds the plane to the scene (Fig.4.20).

```
    /**
     * Adds a plane with a specified texture for the avatar to walk on
     */
    addFloor(){
        //Create floor geometry
        const geometry = new THREE.PlaneBufferGeometry(10,20,512,512);

        //Create and position floor
        this.floor = new THREE.Mesh(geometry, this.floorMaterial);
        this.floor.rotation.x = Math.PI/2;
        this.floor.position.y = -1;
        this.floor.position.z = 5;

        //Add floor to the THREE scene
        this.scene.add(this.floor);
    }
```

Figure 4.20: The **addFloor()** function creates a plane for the floor and adds it into the scene.

addSky() adds a sphere to the scene that is large enough to fit the floor and avatar inside it. **addSky()** then adds a sky texture to the inside of the sphere to give the illusion of a sky surrounding the scene.

```
    /**
     * Adds a sphere with a sky texture to the scene
     */
    addSky(){
        //Create a sky dome
        const geometry = new THREE.SphereGeometry(30,32,32);

        const sphere = new THREE.Mesh(geometry, this.skyMaterial);
        sphere.position.z = 15;
        sphere.position.y = -15;

        this.scene.add(sphere);
    }
```

Figure 4.21: The **addSky()** function creates a sphere with a sky texture and adds it to the scene.

4.3 Version 3.0 Implementation

4.3.1 Game Settings

The first feature implemented in version 3.0 was the Game Control System (GCS). The GCS allows the clinician to change game settings such as the avatar colour or the ground material. The concept of this feature was adapted from the thesis author's previous Master of Engineering Project [8]. To make it easier to change game settings from the GCS, a **game** class was created (see appendix B.7). The **game** class performs all the functionality implemented in version two but also contains a new function - **updateProperty()** (Fig.4.22). This function takes a string as input that describes a property of the game that needs to be updated. As its second input, **updateProperty()** takes the value the property should update to. **updateProperty()** uses these two inputs to change the specified game property to the specified value. Individual settings in the GCS make calls to **updateProperty()** when the user updates them.

```
    /** Updates any property within the game from the GCS */
    updateProperty(name, value){
        switch(name){
            case "avatar bone colour":
                this.avatar.updateBoneColour(value);
                break;
            case "avatar head colour":
                this.avatar.updateHeadColour(value);
                break;
```

Figure 4.22: Excerpt from **updateProperty()** showing how game settings are updated.

4.3.2 MediaPipe Pose Implementation

The second feature implemented in version 3.0 was the MediaPipe Pose pose estimator. This was accomplished by creating a new class **MediapipePose** (see appendix B.8) that extends **PoseEstimator**. MediaPipe Pose was chosen, as when testing its Web Demo [113], it had a noticeably higher accuracy than MediaPipe Holistic. The implementation of MediaPipe Pose is almost identical to that of MediaPipe Holistic (see section 4.2.1).

MediapipePose incorporates avatar scaling into its design. This feature keeps the avatar the same size regardless of the user's height (this was an issue encountered during version 2's development). To scale the avatar, the y coordinates of each joint are normalised with respect to the avatar's height, and an offset is applied so the avatar's feet appear at ground level (Fig.4.23).

```
//Set results with offset and scaling
results.poseLandmarks.forEach(element => {
    this.body[i] = { coordinates: new THREE.Vector3(element.x-1.05, (0.6*((element.y-this.avatarOffsetY)/this.avatarHeight))-0.7, -element.z + this.metrics.realTimeMetrics.avgDistance), confidence: element.visibility}
    i++
});
```

Figure 4.23: Setting each of the joint co-ordinates with scaling.

4.3.3 Metrics Detection

The final feature implemented in version 3.0 was real-time metrics calculation (Fig.4.24) in the **MetricsCalculator** class (see appendix B.9). The approach when implementing this was to convert the OpenPose metrics calculation code from Python into JavaScript. The script includes some code for calculating more accurate post-session metrics but this code was later moved to MATLAB and improved upon, so it will be discussed it in chapter 6.

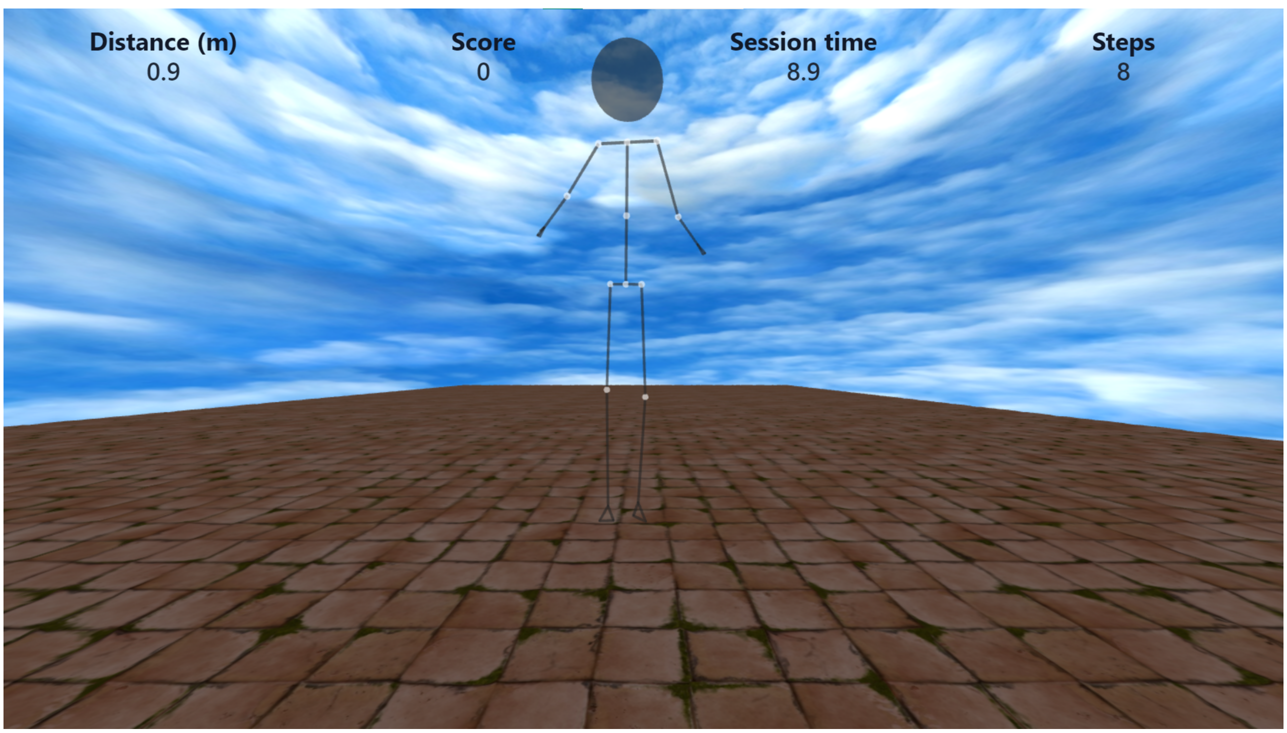

Figure 4.24: Real time metrics within the biofeedback game.

The only other item of note is the data structure in **MetricsCalculator** that keeps track of the important joint locations (**trackerVariables.jointPoints_m**). This is used to output the JSON data for the MATLAB analysis in chapter 6.

5 Testing

The Next.js testing framework Cypress was used to test the UI of the training page. The tests designed are as follows:

Page area	Test Description	Result
Training page...	should open	PASS
Avatar settings...	allows user to select different bone colours	PASS
Avatar settings...	allows the user to select different joint colours	PASS
Avatar settings...	allows the user to select different head colours	PASS
World settings...	allows the user to select different ground textures	PASS
World settings...	allows the user to select different sky textures	PASS
Interactions settings...	allows the user to enable and disable the colours task	PASS
Interactions settings...	allows the user to set the speed for the colours task	PASS
Interactions settings...	allows the user to set the time for the test	PASS
Mobile switch displays button...	should switch to the game display	PASS
Mobile switch displays button...	should switch back to the video display	PASS
Mobile menu open & close button...	should close the settings menu	PASS
Mobile menu open & close button...	should open the settings menu	PASS

Settings navbar...	should only display the avatar settings when the avatar button is clicked	PASS
Settings navbar...	should emphasize the avatar button when the avatar button is clicked	PASS
Settings navbar...	should de-emphasize the world button when the avatar button is clicked	PASS
Settings navbar...	should de-emphasize the interactions button when the avatar button is clicked	PASS
Settings navbar...	should only display the world settings when the world button is clicked	PASS
Settings navbar...	should emphasize the world button when the world button is clicked	PASS
Settings navbar...	should de-emphasize the avatar button when the world button is clicked	PASS
Settings navbar...	should de-emphasize the interactions button when the world button is clicked	PASS
Settings navbar...	should only display the interactions settings when the world button is clicked	PASS
Settings navbar...	should emphasize the interactions button when the interactions button is clicked	PASS
Settings navbar...	should de-emphasize the avatar button when the interactions button is clicked	PASS
Settings navbar...	should de-emphasize the world button when the interactions button is clicked	PASS

Table 5.1: Table describing each of the unit tests for the UI.

The testing code is included in appendix C.1. Cypress was unable to test the biofeedback game since the majority of its features required visual confirmation that they were functioning. A series of visual tests were therefore carried out for the game:

Page area	Test Description	Result
Avatar...	should follow the patient's movements	PASS
Avatar...	should change to the colour the clinician selects	PASS
Avatar...	should move forward when the patient walks forward on the treadmill	PASS
Environment...	should change to the ground texture the clinician selects	PASS
Environment...	should change to the sky texture the clinician selects	PASS
Interactions settings...	should change the test duration	PASS
Interactions settings...	should turn the stroop test on and off	PASS
Interactions settings...	should adjust the speed of the stroop test	PASS

Table 5.2: Table describing each of the visual tests for the game.

6 Validation

6.1 Data Capture and Processing

Upon capturing the data for the first patient, it was evident that Tracking Tools was struggling to capture the six joint positions required. When capturing more than two joint locations, it was impossible to get a consistent trace on each joint's location. This issue occurred because the only version of Tracking Tools available at the time was a limited one not designed for tracking large numbers of objects at once. Therefore, the decision was made to focus solely on ankle location rather than the planned hip, knee and ankle locations. This decision removed hip flexion and knee flexion from the list of metrics that could be measured.

Some trials did not capture data, and were unable to be repeated due to time constraints:

- For Patient01, two rounds were captured for each treadmill speed instead of three.
- For Patient05, two rounds were captured instead of three for the lowest treadmill speed.

The next stage in the study was to apply a low-pass filter to both datasets to optimize them for gait event detection.

Before applying a low pass filter to the OptiTrack data, a preliminary step needed to be performed. When the OptiTrack can not locate a marker, it assigns it a value of zero. This feature creates sporadic jumps in the data (Fig.6.1) which would affect the results of the filtering stage. Cubic interpolation (Fig.6.2) was used to fill in the missing values. This technique was not required for the MediaPipe data as it was continuous (no breaks in the signal). The filtering stage uses a moving average filter with a window size that is 1/10th the period of the waveform. This filter removes high-frequency fluctuations so the peaks of the data can be detected.

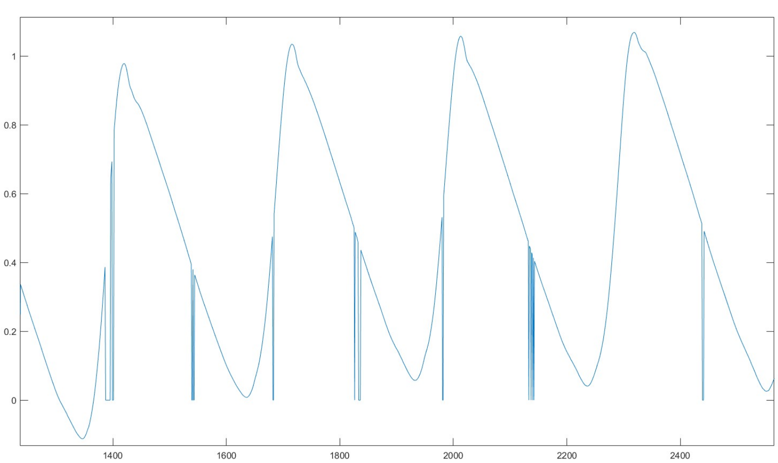

Figure 6.1: Sporadic data from the OptiTrack.

```
function [array] = cubic_spline(array)
%cubic_spline uses cubic interpolation where zeros (null values) appear

        array_start = -1;
        for i = 1:length(array)
            %if the value is null (0) set it to NaN
            if array(i) == 0.0000000
                if array_start ~= -1
                    array(i) = NaN;
                end
            %record first non zero value
            elseif array_start == -1
            array_start = i;
            end
        end

    %interpolate missing values (cubic spline)
    array(array_start:end) = fillmissing(array(array_start:end),'spline');
end
```

Figure 6.2: The cubic_spline function for removing null values using cubic interpolation.

After the filtering stage, outliers needed to be removed from the OptiTrack data. This step was important as the gold standard data needed to be accurate for validation purposes. It was decided the best method of determining outliers was to make a priori assumptions based on normal gait data. Each source of data would then be visually inspected for anomalies and this data would be excluded from the results where appropriate.

It was decided the gait data from the OptiTrack should fit the two following criteria:

- Periodic, sine-like waveform with a roughly constant frequency.
- No sudden jumps in amplitude outside the range of the waveform.

Once the anomalous data had been determined, the decision had to be made whether to exclude it completely or trim it down to remove the outliers. It was decided that any clips with more than 30 seconds of continuous footage without an outlier could be trimmed, otherwise, the clip would be discarded. 30 seconds was chosen as from experience, enough gait data could be collected in this time to calculate average metrics.

Using the criteria mentioned above, six outlier recordings were discarded and eight outlier recordings were trimmed (Tab.6.1 and Tab.6.2). This totalled 482.3 seconds of footage.:

Patient Code	Recording Code	Amount of Data Removed (seconds (s))	Reason
Patient02	Pat02_09	all data - 60.0s	Breaks in waveform periodicity (BWP)
Patient03	Pat03_02	12.0s	Jump in amplitude outside the usual range of the waveform (JIA)
Patient03	Pat03_03	all data - 60.0s	JIA
Patient03	Pat03_05	13.0s	JIA
Patient03	Pat03_06	8.0s	JIA
Patient03	Pat03_07	27.0s	JIA
Patient03	Pat03_08	all data - 60.0s	BWP
Patient03	Pat03_09	13.5s	JIA
Patient04	Pat04_01	all data - 60.0s	BWP
Patient05	Pat05_02	all data - 60.0s	BWP
Patient05	Pat05_06	all data - 60.0s	BWP
Patient06	Pat06_04	25.3s	JIA
Patient06	Pat06_07	10.0s	JIA
Patient06	Pat06_08	13.5s	JIA

Table 6.1: Table describing each of the modifications to the OptiTrack data.

Patient Code	Recordings Trimmed	Recordings Removed
Patient02	0	1
Patient03	5	2
Patient04	0	1
Patient05	0	2
Patient06	3	0

Table 6.2: Table describing how much data was trimmed and discarded for each patient.

Upon inspection, the cause of the above outliers was determined to be large gaps in tracking data on the left and/or right ankle. These gaps were caused by the OptiTrack system failing to detect the markers. Whilst cubic interpolation was useful for smaller gaps in the data, larger gaps were much harder to fill (fig.6.3), causing the outliers seen in tab.6.1.

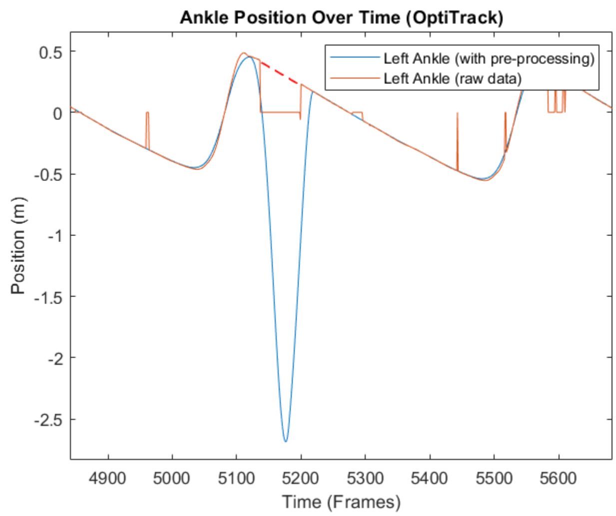

Figure 6.3: Example of the cubic interpolation algorithm losing accuracy with large gaps in data. The orange line is the raw data, and the blue line is the pre-processed data with cubic interpolation. The red dashed line gives an idea of how the interpolation should have looked.

Once the outliers had been discarded, the metrics extraction process could begin. The first stage in this process is to detect minima and maxima from the ankle position data. MATLAB's **findpeaks()** is used to achieve this (Fig.6.4 and Fig.6.5). The minimum distance between peaks (**MinPeakDistance**) was set to be $\frac{3}{4}$ of the waveform's period (calculated by finding 1/cadence) and the minimum drop in amplitude between peaks (**MinPeakProminence**) was set to be $\frac{3}{4}$ of the root mean square (RMS) of the signal. These values were determined to be the best for peak detection through trial and error.

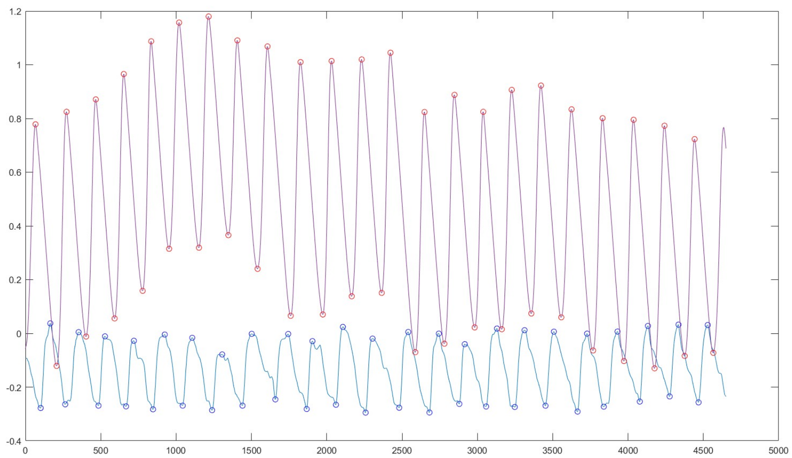

Figure 6.4: Results of the peak detection algorithm. The circle markers denote minima and maxima and the red and blue lines are the OptiTrack data and MediaPipe Pose data respectively.

```
%find maxima in the data
[peaks_l,locs_l] = findpeaks(ankle_l,'MinPeakDistance',(fundamental_l*
    frame_rate)*0.75,'MinPeakProminence',rms_l*0.75);
[peaks_r,locs_r] = findpeaks(ankle_r,'MinPeakDistance',(fundamental_r*
    frame_rate)*0.75,'MinPeakProminence',rms_r*0.75);

%find minima in the data
[min_l,locs_min_l] = findpeaks(-ankle_l,'MinPeakDistance',(fundamental_l*
    frame_rate)*0.75,'MinPeakProminence',rms_l*0.75);
[min_r,locs_min_r] = findpeaks(-ankle_r,'MinPeakDistance',(fundamental_r*
    frame_rate)*0.75,'MinPeakProminence',rms_r*0.75);
```

Figure 6.5: Using **findpeaks()** to detect maxima and minima.

Temporal metrics are calculated by looking at the time intervals between detected peaks. The temporal metrics are:

- Swing time(s).
- Stance time(s).
- Double support time(s).
- Single support time(s).

The temporal metrics were calculated according to the diagram below (Fig.6.6). For the temporal metrics code, see **get_swing_stance()** in appendix E.1 and **get_double_support()** in appendix E.2. The number of steps is also calculated by counting the number of peaks.

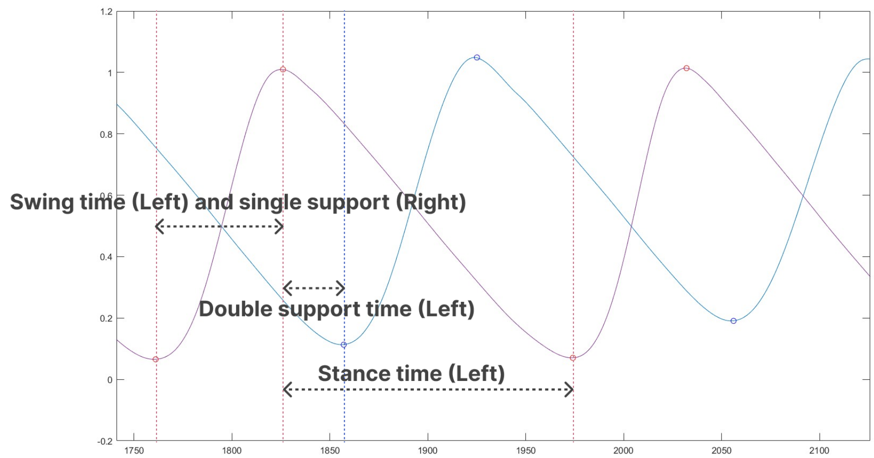

Figure 6.6: Calculating temporal metrics from peak data. The red line is the left ankle data and the blue line is the right ankle data.

In terms of the spatial metrics, stride length is calculated by looking at the distance intervals between minima and maxima (Fig.6.7). The sum of the strides is then calculated to get the distance walked. Finally, the distance walked is divided by the duration of the clip to get the walking speed in m/s. For the stride length code, see **get_swing_stance()** in appendix E.1.

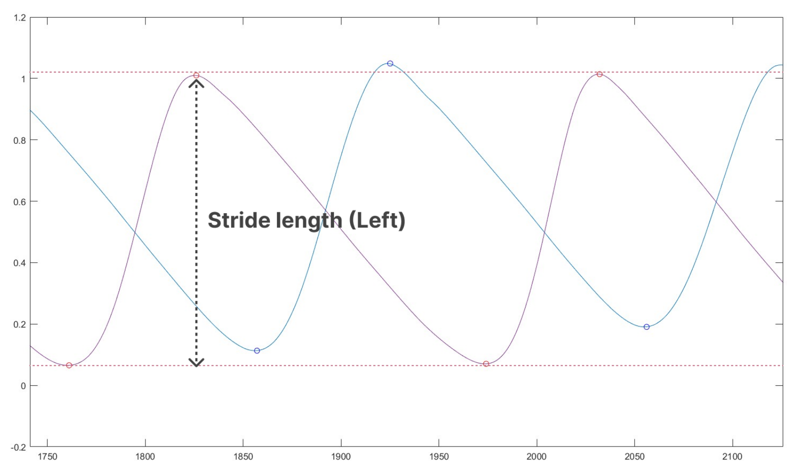

Figure 6.7: Diagram showing how stride length was calculated.

Cadence is calculated by finding the fundamental frequency of the gait data. The Fast Fourier Transform (FFT) is used to split the gait data into it's constituent frequencies and the lowest frequency is assumed to be the fundamental. For the code used to calculate the cadence, see lines 12-16 of **get_metrics()** in appendix E.3 and **get_fundamental()** in appendix E.4.

At this stage, all the metrics had been extracted from the MediaPipe and OptiTrack data. Upon initial inspection of the metrics data, one anomalous result was found and removed (Pat06_01). This result was removed as many of the metric values for the right foot were equal to either null or zero, meaning the result could not be used for accuracy calculations (MATLAB functions cannot be called on null data values). Upon inspecting the step count for the right foot, it was observed that MediaPipe's number of steps was 1, whereas the OptiTrack's was 18. This large difference in step count signified that the peak detection algorithm failed on the MediaPipe data, only detecting one step, hence the null values. This was confirmed when viewing the peak detection results graphically (Fig.6.8).

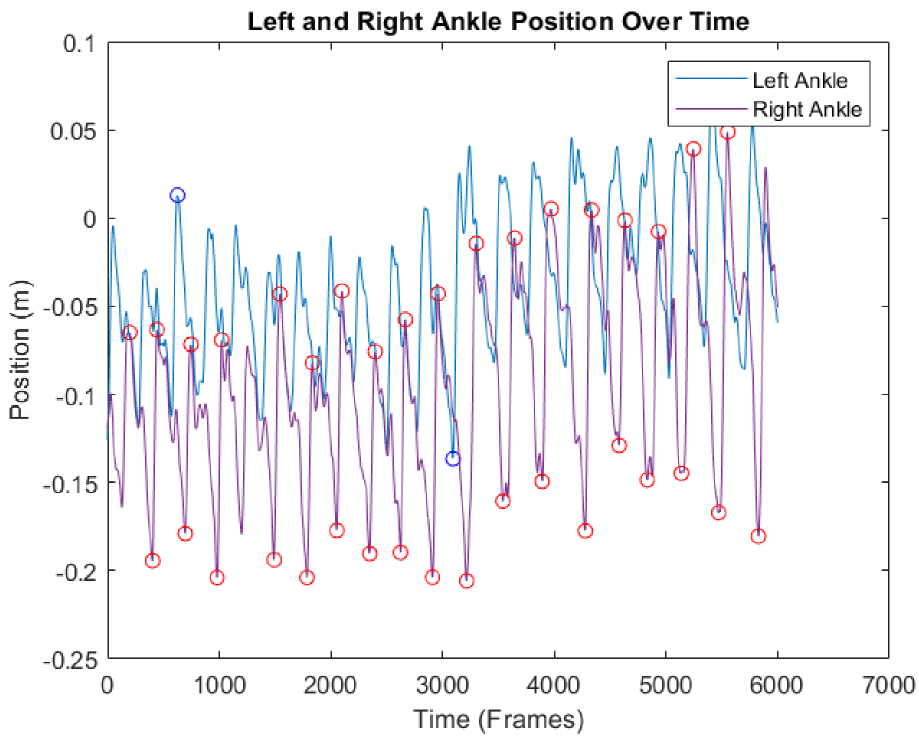

Figure 6.8: Graph showing anomalous peak detection results for Pat06_01.

After this anomalous data was removed, the next step in the validation study was to separate the data into classes. A class consisted of all the data captured for a particular patient and treadmill speed. For example, Pat01_Speed01 means all the data captured at the slowest speed for Patient 1. After the data had been sorted into classes, the mean percentage error was calculated for every metric in each class using the equation in Fig.6.9.

$$\tilde{\epsilon}_{\%} = \frac{1}{n} \sum \frac{m_{med} - m_{opt}}{m_{opt}} \times 100$$

Figure 6.9: Equation for the mean error ($\tilde{\epsilon}_{\%}$) of a metric, where m_{opt} is the metric value for the OptiTrack, m_{med} is the metric value for MediaPipe and n is the number of samples.

The mean errors from every class were then combined to produce a box plot showing the distribution of error for each metric across the different classes (Fig.6.10 and Fig.6.11).

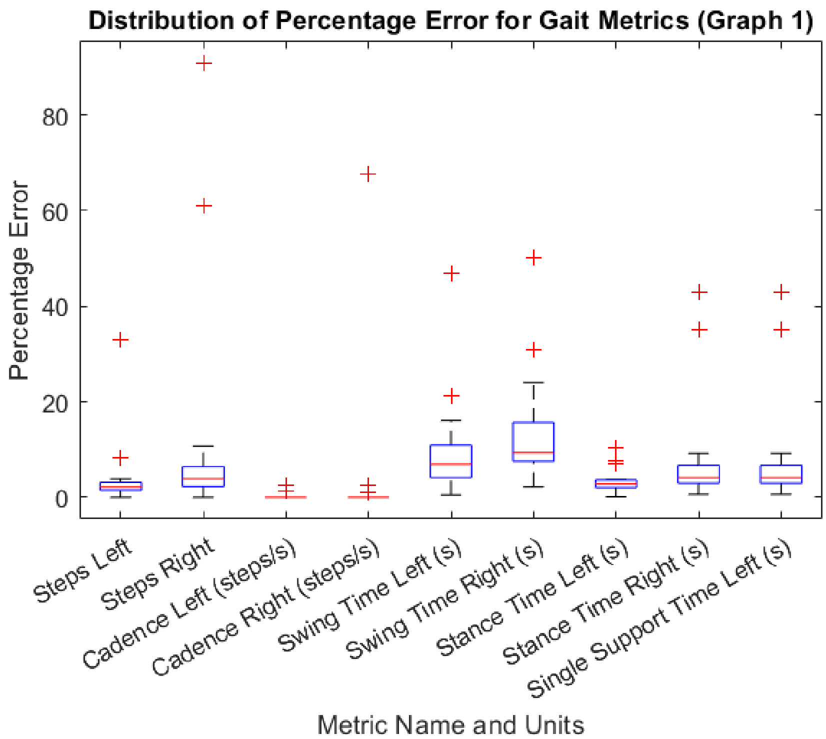

Figure 6.10: Box plot showing distribution of mean percentage error for metrics (Graph 1).

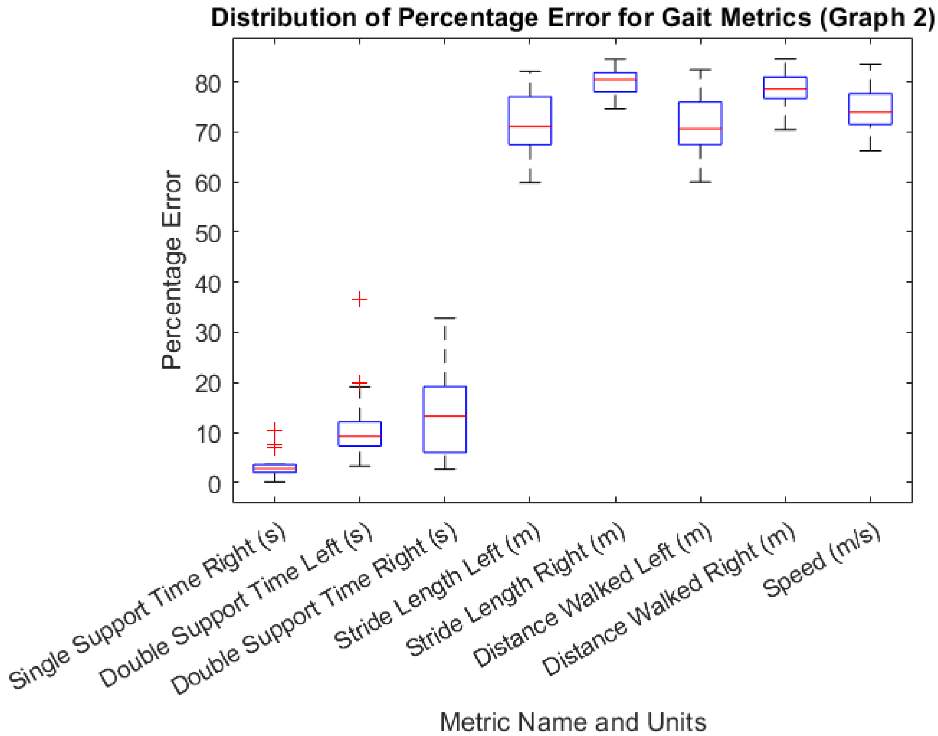

Figure 6.11: Box plot showing distribution of mean percentage error for metrics (Graph 2).

From these box plots, a list of classes that were potential outliers was generated along with the number of metrics that were outliers (Tab.6.3).

Class Name	Number of Metrics Detected as Outliers
Pat01_Speed01	1
Pat02_Speed01	4
Pat02_Speed03	1
Pat04_Speed03	1
Pat06_Speed01	9
Pat06_Speed02	4
Pat06_Speed03	5

Table 6.3: Number of different metrics detected as outliers for outlier classes.

An important initial observation to make is that over half of the outliers occurred for both Pat06 (18/25 outliers) and Speed01 (14/25 outliers). Although these results cannot be discarded, this could suggest that MediaPipe found slower gait speeds and particular types of gait more difficult to process.

It was decided that the data for each of the outlier classes should be investigated individually to determine whether there was a root cause. Some of the data of the outlier classes had high percentage errors for number of steps, similar to the anomalous result Pat06_01. This suggested that the peak detection algorithm was not working correctly. Cadence was found to be the main cause for this, as it was sometimes calculated incorrectly due to very low frequency oscillations in the gait data. This miscalculation meant the **MinPeakDistance** of the peak detection algorithm was set higher than it should have been, resulting in less peaks being detected. The MediaPipe results with this type of error could not be discarded as they were representative of the pose estimation method's accuracy. However, the OptiTrack result for Pat06_02 also displayed this behaviour with a high percentage of the peaks being missed during data processing (Fig.6.12). As the OptiTrack was the gold standard, the Pat06_02 result was discarded. It was decided final results would be shown both with and without this result.

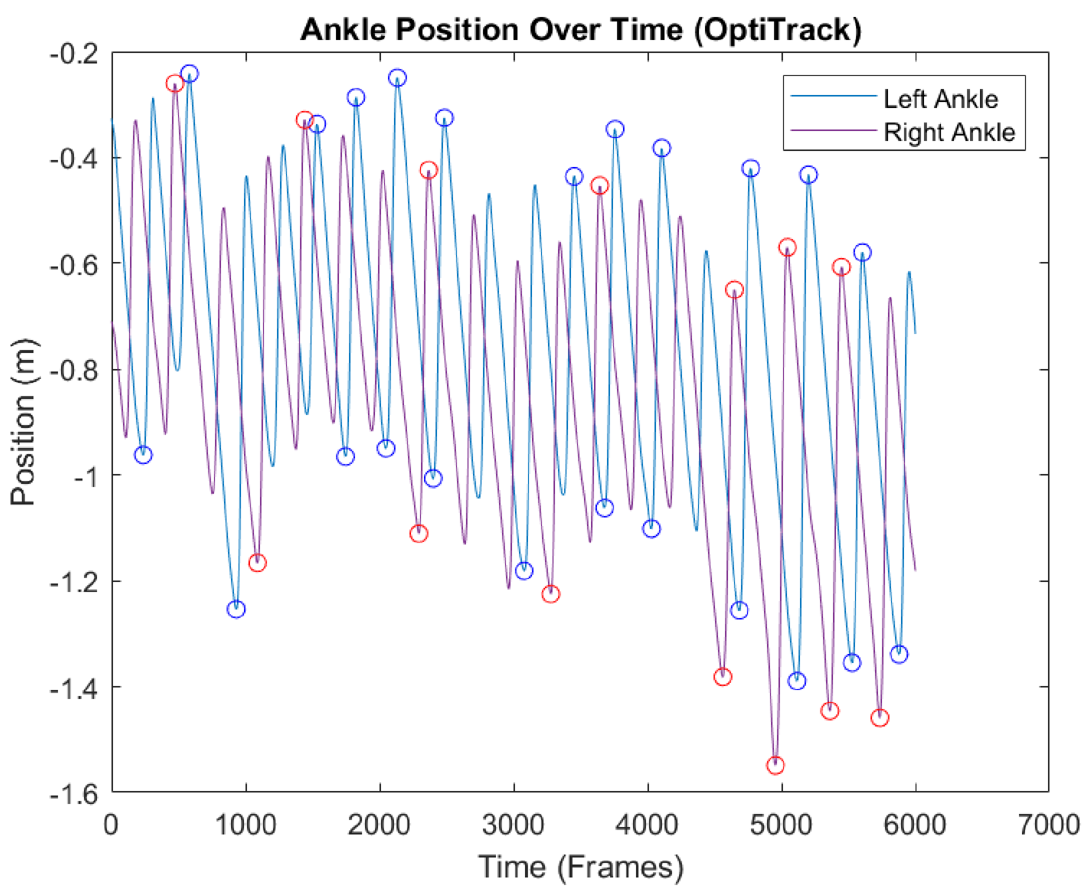

Figure 6.12: Graph showing the high amounts of peaks missed in the Pat06_02 result.

The other type of outlier observed was due to inaccurate data from MediaPipe. Sometimes MediaPipe would detect a secondary fluctuation within the gait cycle that was not present in the OptiTrack data. If it was big enough, this fluctuation caused the peak detection algorithm to trigger twice within one gait cycle (Fig.6.13). Again, these results could not be discarded, as they were representative of the pose estimation method.

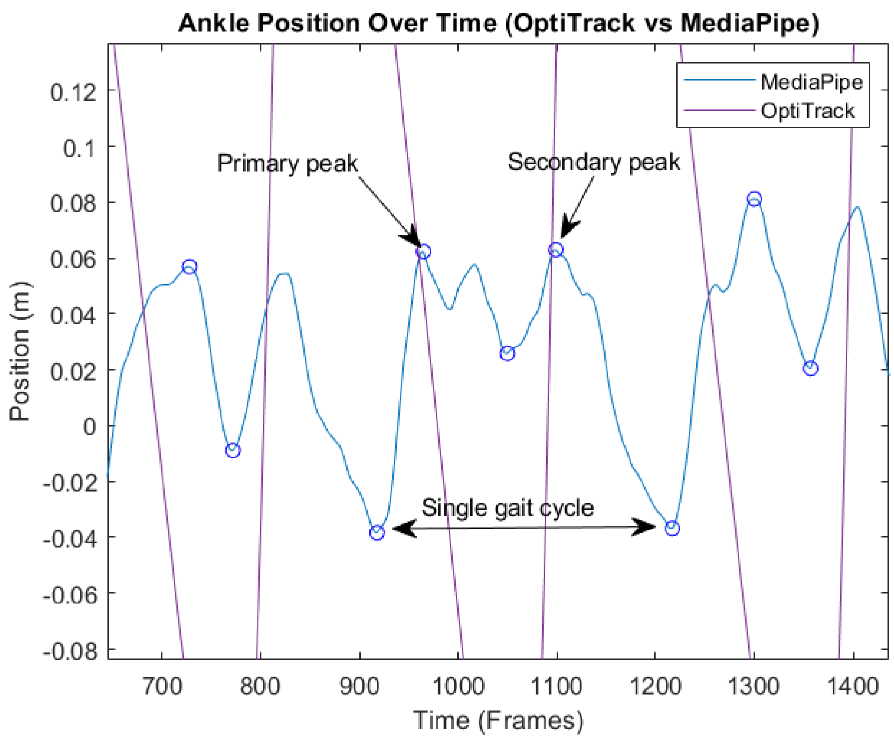

Figure 6.13: Graph showing the secondary fluctuation in the gait cycle detected by MediaPipe.

Once the outliers had been removed, the mean percentage errors were found for the study as a whole. These results are shown overleaf (Tab.6.4).

Metric Name	Mean Percentage Error (2d.p.)	Mean Percentage Error After Removing Outliers (3d.p.)
Steps left	3.96(9.28)%	2.63(3.14)%
Steps right	11.94(30.33)%	8.26(18.55)%
Cadence left	0.24(1.19)%	0.24(1.20)%
Cadence right	4.95(21.19)%	5.07(21.43)%
Swing time left	9.70(11.51)%	8.30(7.07)%
Swing time right	13.54(13.73)%	12.40(11.65)%
Stance time left	3.43(3.61)%	3.03(2.49)%
Stance time right	8.41(14.91)%	7.17(12.66)%
Single support time left	8.41(14.91)%	7.17(12.66)%
Single support time right	3.43(3.61)%	3.03(2.49)%
Double support time left	11.42(11.08)	10.14(7.32)%
Double support time right	13.43(12.12)%	12.40(10.20)%
Stride length left	71.12(6.42)%	70.90(6.33)%
Stride length right	79.86(2.79)%	79.72(2.65)%
Distance left	70.88(6.52)%	70.93(6.59)%
Distance right	78.39(4.73)%	78.81(3.86)%
Speed	74.62(4.79)%	74.82(4.66)%

Table 6.4: Mean percentage error values with standard deviation in brackets.

6.2 Discussion

The mean percentage errors show that MediaPipe's time-based metrics are much more accurate ($0.24 < \tilde{\epsilon}_{\%} < 12.40$) than spatial metrics ($70.90 < \tilde{\epsilon}_{\%} < 79.72$). The results for time-based metrics matched predictions from initial observations of the data and certainly warrant further investigation. It is apparent that whilst Mediapipe shows fluctuations that are not present in the OptiTrack data, it still represents the peaks and troughs of the data accurately enough to estimate time-based metrics.

The Spearman's Rank Correlation Coefficient (SRCC) between the spatial metrics for MediaPipe and the OptiTrack was calculated using the Spearman's Rho function [33] (Fig. 6.14). The SRCCs signified a very strong positive correlation (Tab.6.5) for four out of five metrics (moderate correlation for the left leg's stride length). SRCC was chosen as some of the spatial metrics data had outliers (Fig.6.15) so the Pearson's Correlation Coefficient could not be used. This result could signify that the larger percentage error of the spatial metrics is caused by a scaling issue and could be fixed by applying a gain factor.

$$\rho = 1 - \frac{6\sum(R_{med} - R_{opt})^2}{n(n^2 - 1)}$$

Figure 6.14: Equation for the SRCC (ρ) of a metric, where R_{opt} is the rank of each metric value for the OptiTrack, R_{med} is the rank of each metric value for MediaPipe, and n is the number of samples (from [33]).

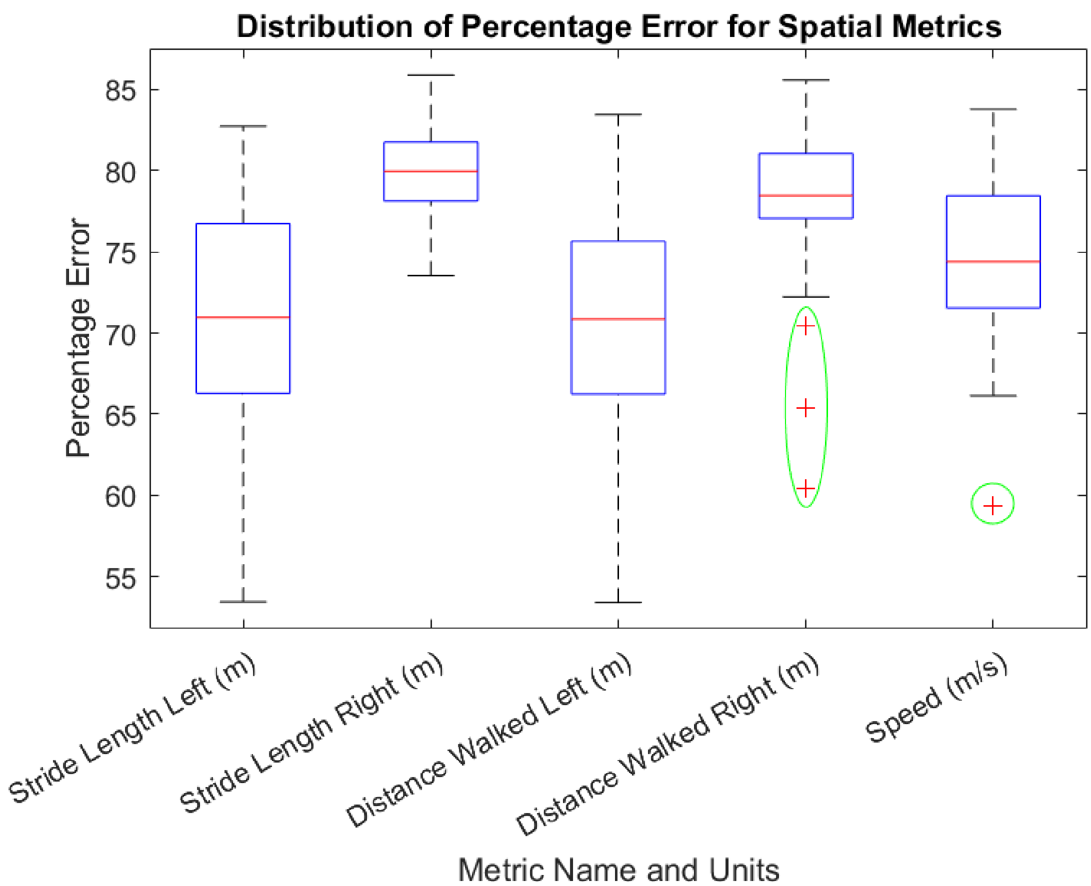

Figure 6.15: Graph showing the distribution of spatial metrics (outliers circled in green).

Metric Name	SRCC (2d.p.)	SRCC After Removing Outliers (2d.p.)
Stride length left	0.40	0.39
Stride length right	0.81	0.82
Distance left	0.75	0.74
Distance right	0.87	0.86
Speed	0.86	0.85

Table 6.5: SRCC for spatial metrics.

Finally, another notable observation is that the percentage error for the right foot is greater than the left for every metric. The only exception to this is single support time (single support time is calculated using data from the opposite foot, so the percentage errors are swapped). This difference in percentage error could mean that MediaPipe's right ankle detection is less accurate than that of the left ankle. Again, a study with a larger sample size would be necessary to validate this claim.

7 Evaluation and Further Work

7.1 Evaluation

Below is a screenshot of the finished web app running (Fig.7.1). The biofeedback game and camera feed are on the left and the GCS is on the right. Evaluating the general appearance of the application, it is split into distinct sections, making it simple and easy to understand. This will make it easy for clinicians to use it without much prior training.

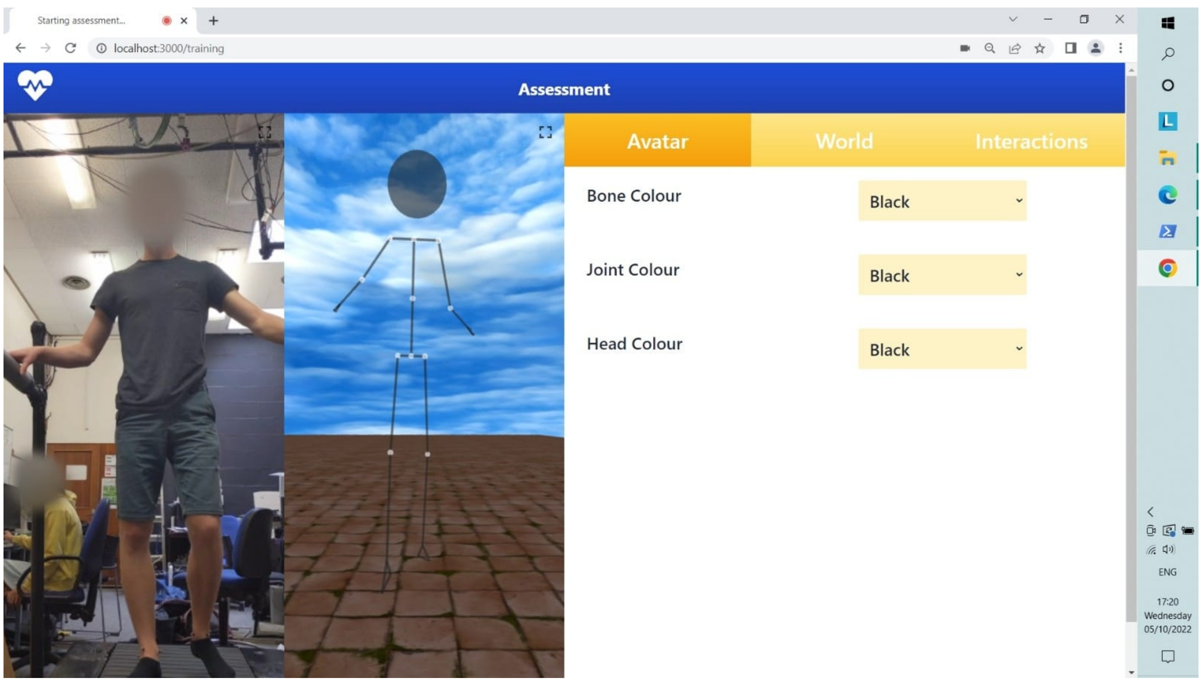

Figure 7.1: Finished web app including GCS, camera feed, and biofeedback game.

Assessing the finished application against the aims and objectives in section 2.7, it is clear that the majority have been accomplished. The MediaPipe Pose pose estimator is implemented, and used to create a biofeedback game that physiotherapists can use for rehabilitation. This accomplishes aim 3, objective 1 and objective 4. A gait analysis algorithm is also developed to calculate the required gait metrics from MediaPipe Pose, and the algorithm's results are compared to those from the OptiTrack. This achieves aim 1, aim 2, and objective 3. Objective 2 is mostly completed, apart from the calculation of hip and knee angles. Hip and knee angles were not included in the metrics due to the limitations of the OptiTrack software discussed in section 6.1. In addition to the aims and objectives completed, a settings panel was also developed to allow the clinician to customise the gameplay to the patient's needs.

7.2 New Game Features

One way to further improve the biofeedback game would be to add new gameplay features. Some examples of possible features are:

- **Support for VR headsets** - Three.js has native support for this [34]. To make the application a VR application, VR must be enabled and a button must be created to start VR mode (Fig.7.2).
- **Objects for the patient to interact with** - This could be achieved by implementing collision detection using Three.js's Raycaster [114]. Implementing this allows the patient to touch objects using their avatar and make them disappear or move.
- **"Movement challenges" such as squats, calf raises, etc.** - Change in joint angles could be used to calculate when the user has completed a repetition. These measurements of range of motion could also be used to measure the patient's improvement over time.

These features would make the biofeedback game more immersive, which could increase the patient's distraction from chronic pain caused by their condition (as seen in [7]).

```
import { VRButton } from 'three/addons/webxr/VRButton.js';

//Add button to page for VR users
document.body.appendChild( VRButton.createButton( renderer ) );

//Enables VR
renderer.xr.enabled = true;

//Can't use requestAnimationFrame for the game loop in VR
renderer.setAnimationLoop( function () {

    renderer.render( scene, camera );

} );
```

Figure 7.2: Using VR within a THREE.js project [34].

7.3 In-app Metric Calculation

MATLAB was used to calculate the metrics in section 6. Whilst this was useful for the accuracy study, the metric calculation should happen on the web app in its production-ready version. Most of the calculations for the metrics can be transferred directly, however the **findpeaks()**, **movmean()** and **fft()** functions need to be implemented in JavaScript. The alternative to integrating the metrics calculation into the web app would be to create a MATLAB server for the user to send the pose data to. However, the metrics calculation is not very compute-intensive, so handling the computations on the client end would be less costly than hosting a metrics server.

7.4 Improving Spatial Metrics

As noted in Section 6.2, MediaPipe's spatial metrics displayed a very strong positive correlation when compared to the OptiTrack's despite their large percentage error. This might indicate that a relationship exists between MediaPipe's spatial data and the OptiTrack's. This relationship could be determined using regression and used to reduce the percentage error of the results.

7.5 Validating Joint Angles

Due to the limitations of the Tracking Tools software at the University, only ankle data could be captured. This limitation meant that joint angles could not be calculated from the OptiTrack data. A future study could use the full-body tracking version of Tracking Tools to capture hip and knee angles. The equivalent angles from MediaPipe Pose could then be obtained and compared to those from the OptiTrack using mean percentage error calculations.

8 Conclusion

Gait disorders are one of the most common health issues affecting the elderly population. Due to the higher incidence of disability and falls [2] [3] [4] in people with this type of disorder, early detection and intervention is paramount in preventing the occurrence of injuries and accidents.

This thesis has emphasised the value of gait analysis and physiotherapy as a holistic method of detecting and treating gait disorders. It has also highlighted how combining VR-based gait analysis and physiotherapy is more effective than conventional methods at treating various gait disorders, thus establishing the need for a cost-effective version of this system.

The design, implementation and testing phases of the system were documented and the process of assessing it's accuracy against the OptiTrack was described. The results demonstrated that the system's percentage error ($\tilde{\epsilon}_{\%}$) was much less for temporal gait metrics ($0.24 < \tilde{\epsilon}_{\%} < 12.40$) than it was for spatial ones ($70.90 < \tilde{\epsilon}_{\%} < 79.72$). Four out of five spatial metrics also had a "very strong correlation" ($0.74 < r < 0.86$) when compared to the OptiTrack's metrics, meaning the spatial inaccuracy could be reduced using a gain factor. These results support the use of the system as a first response tool for measuring temporal gait metrics and warrant further investigation in an accuracy study with a higher sample size measuring more gait metrics.

This system could make gait analysis and VR rehabilitation more accessible to the general public and would allow medical clinicians to measure gait without the need for specialist motion capture facilities. This accessibility would improve early detection rates for gait abnormalities, improving outcomes for those affected and reducing the risk of accident or injury. The system uses a standard webcam, making it affordable and easy to set up, and can be used on a mobile device in the home. These features could alleviate the strain on healthcare services caused by the cost and lack of accessibility of current gait analysis systems.

A Web App Design

A.1 Version 1 Architecture

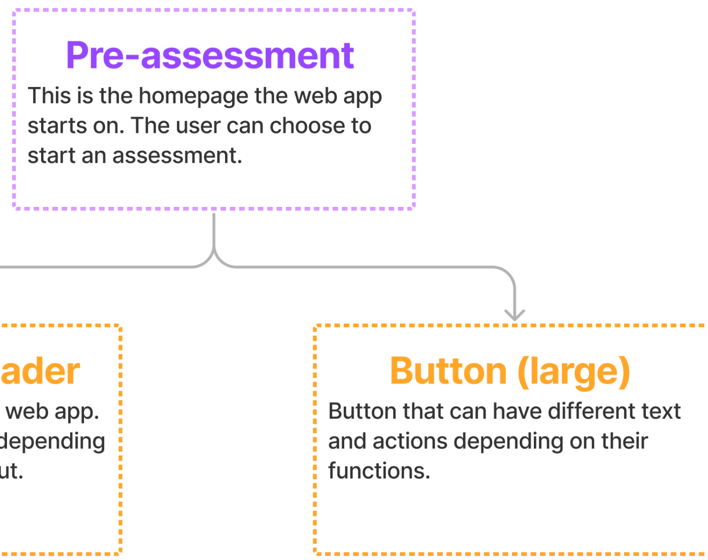

Figure A.1: Flow diagram for pre-assessment page.

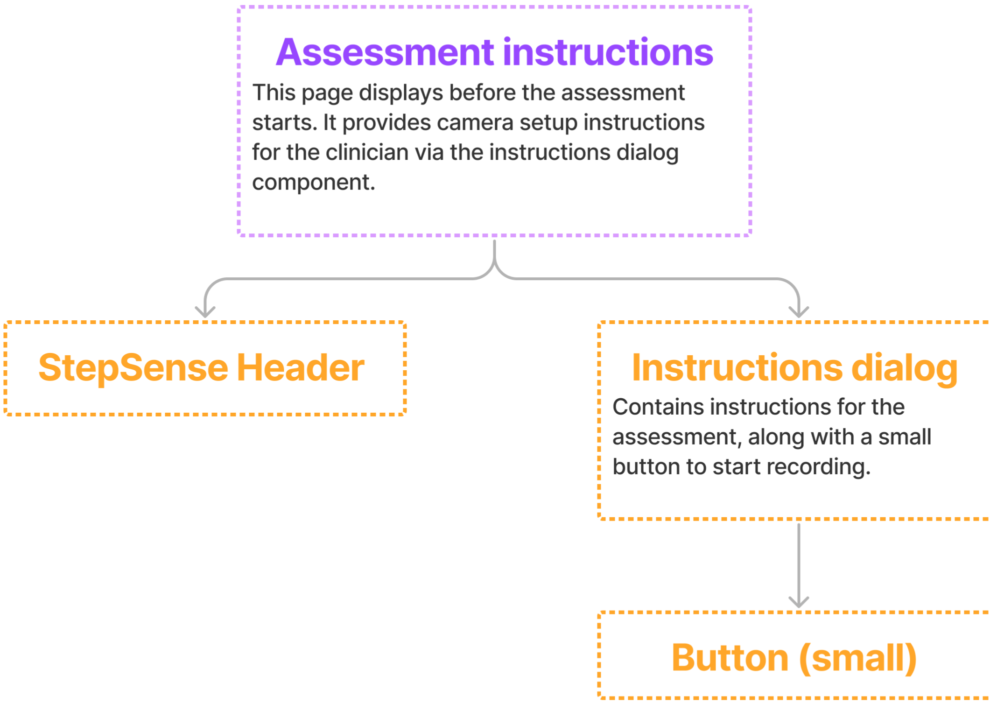

Figure A.2: Flow diagram for assessment instructions page.

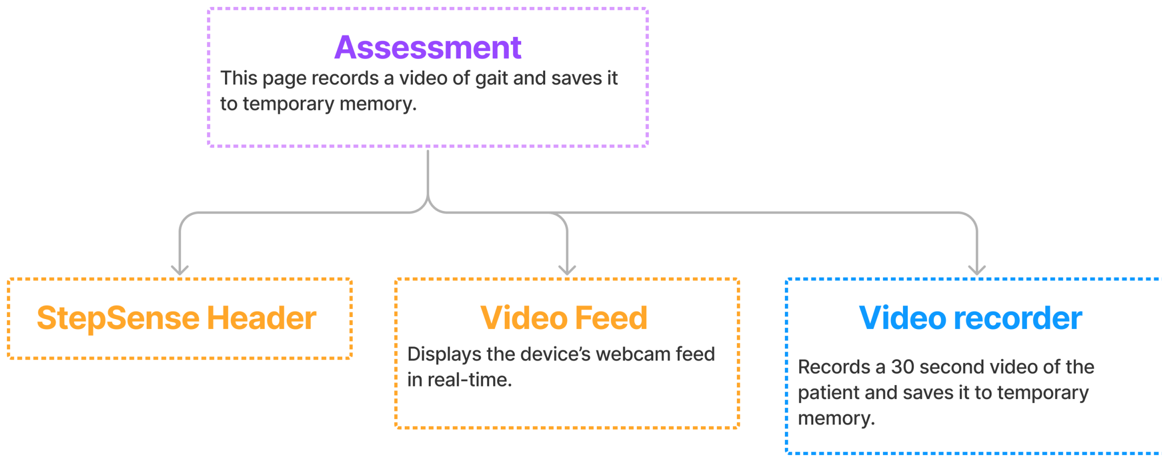

Figure A.3: Flow diagram for assessment page.

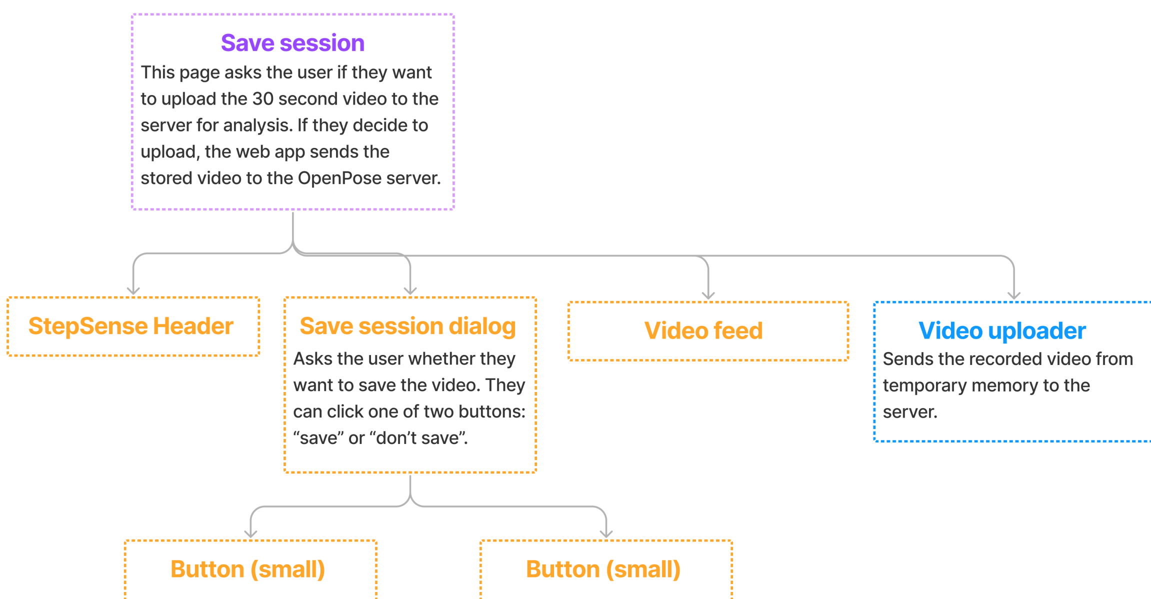

Figure A.4: Flow diagram for save session page.

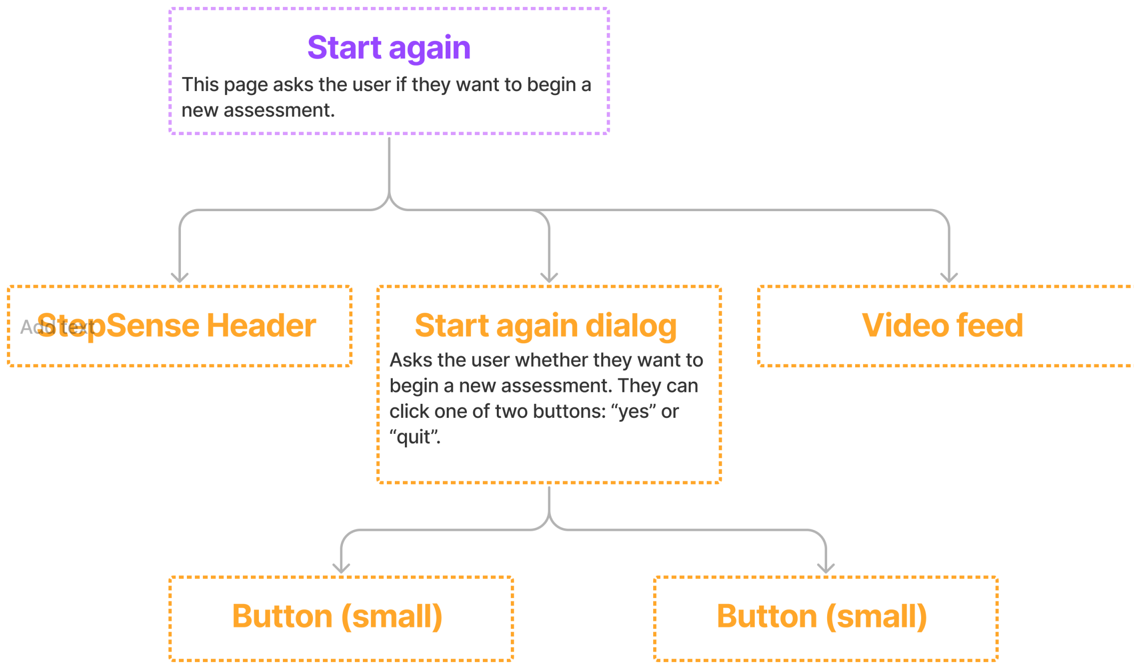

Figure A.5: Flow diagram for start again page.

A.2 Version 2 Mobile UI

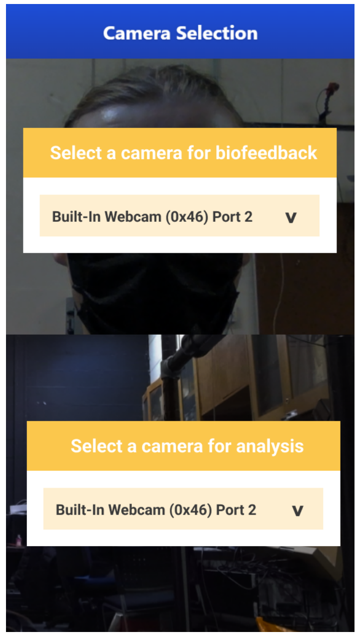

Figure A.6: Mobile layout for select camera page.

Figure A.7: Mobile layout for training camera view. The settings button on the left toggles the settings hamburger menu and the game button on the right switches to the training game view.

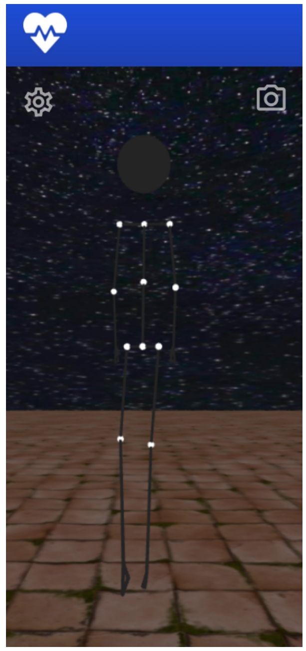

Figure A.8: Mobile layout for training game view. The settings button on the left toggles the settings hamburger menu and the camera button on the right switches to the training camera view.

A.3 Version 2 Architecture

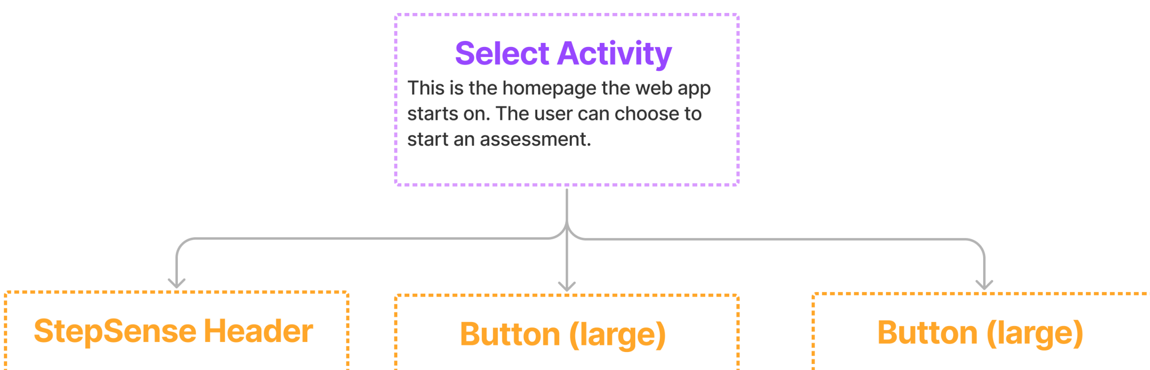

Figure A.9: Flow diagram for select activity page.

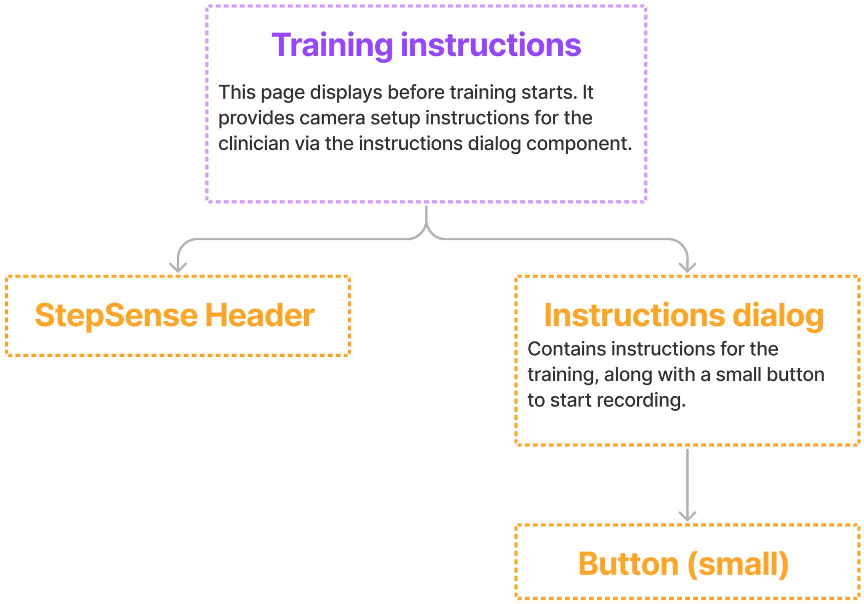

Figure A.10: Flow diagram for training instructions page.

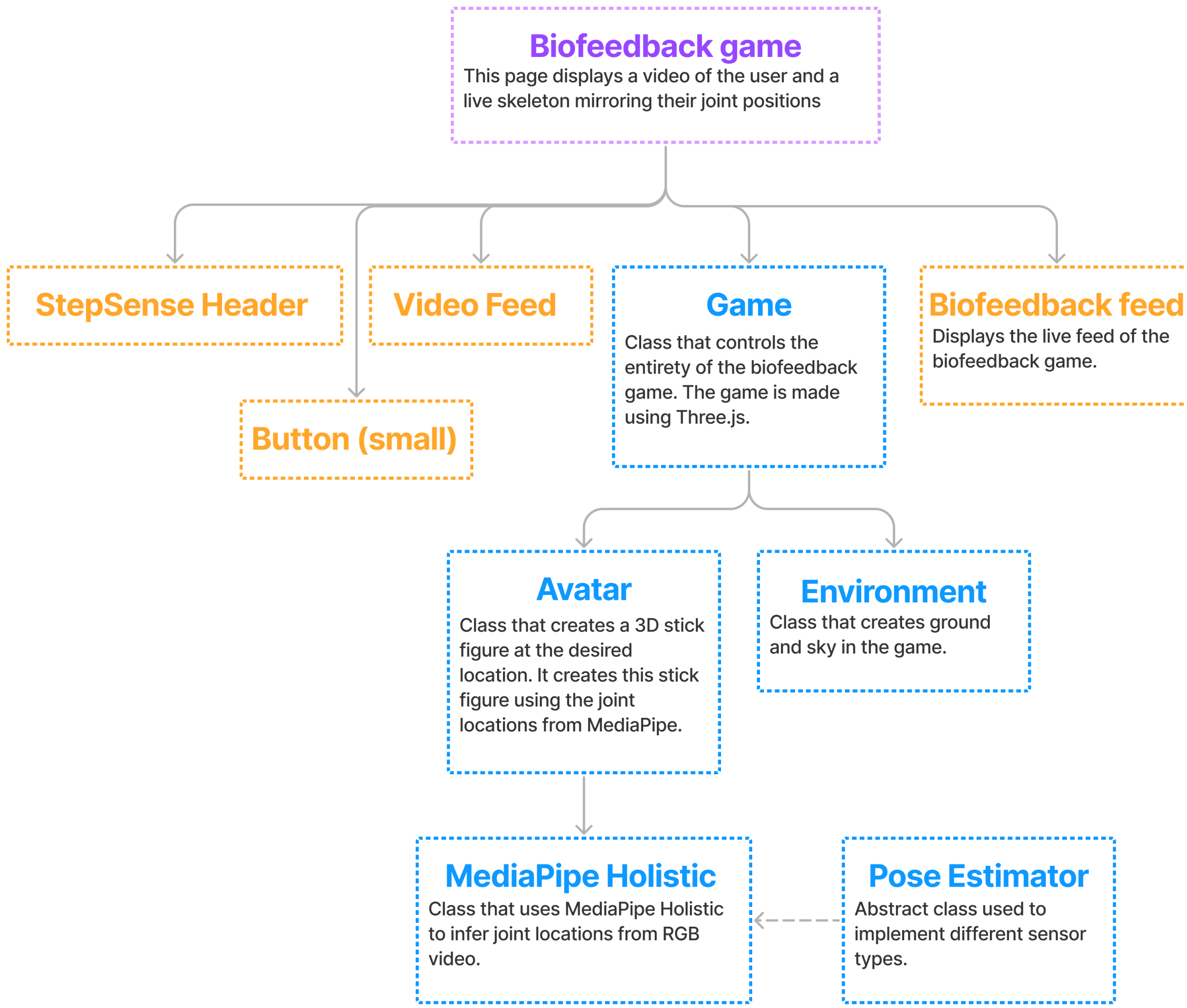

Figure A.11: Flow diagram for biofeedback game page.

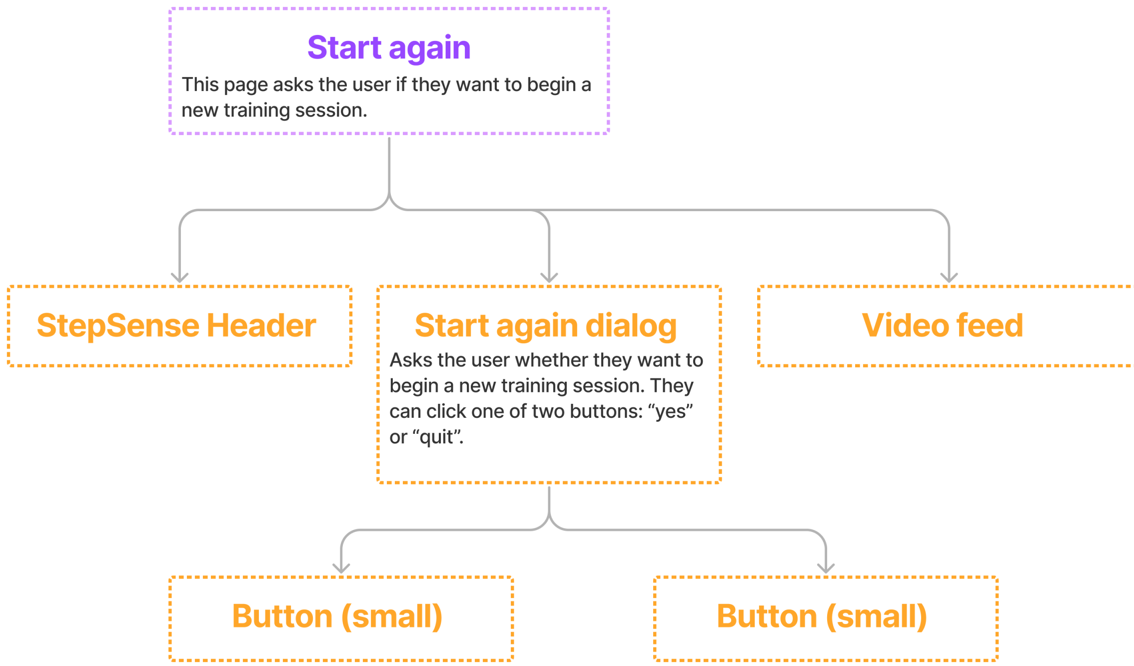

Figure A.12: Flow diagram for start again page (biofeedback version).

A.4 Version 3 Architecture

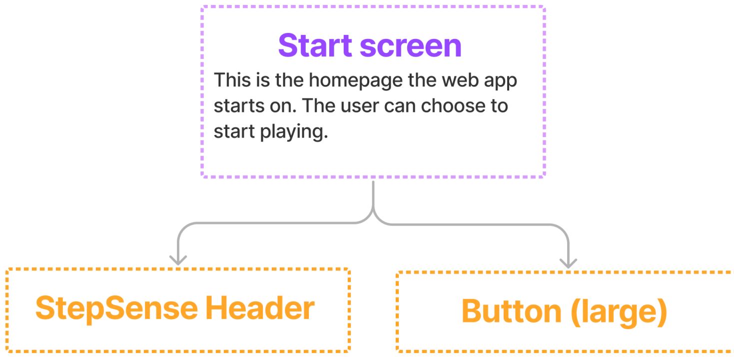

Figure A.13: Flow diagram for start screen.

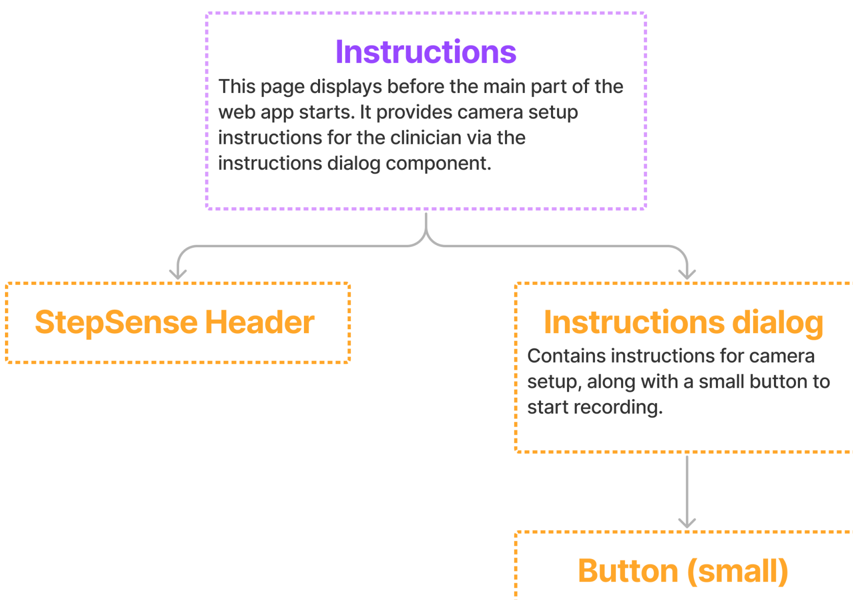

Figure A.14: Flow diagram for instructions page.

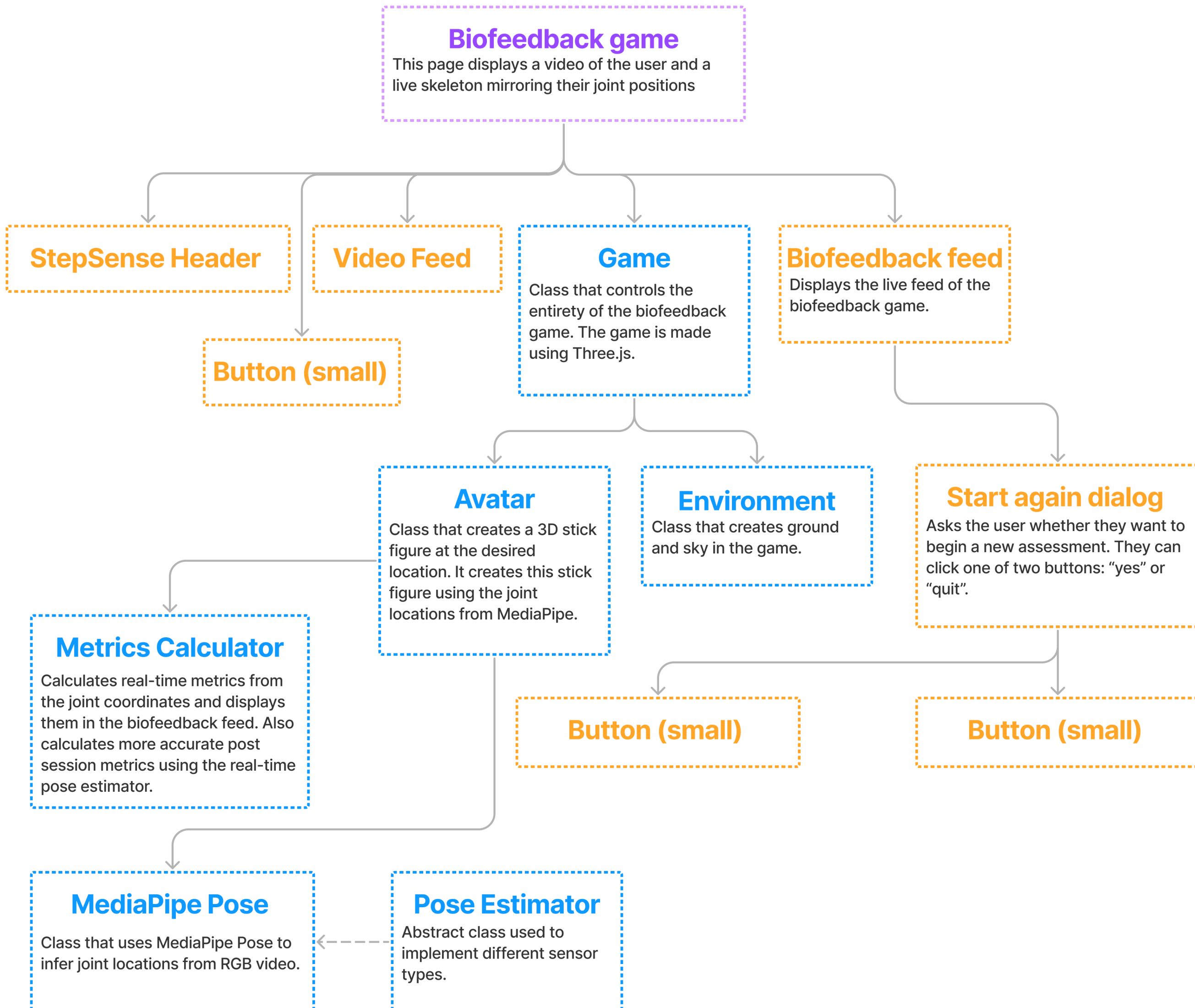

Figure A.15: Flow diagram for biofeedback game page.

B Web App Implementation

B.1 Version 1.0 OpenPose Server

Below is the code for the OpenPose Gait analysis server.

```python
import os
import subprocess
import json
from cv2 import cv2
from flask import Flask, request, make_response
import numpy as np

app = Flask(__name__)

# openpose body_25 indexes for left and right hip and heel
L_HEEL = {'x': 63, 'y': 64, 'c': 65}
R_HEEL = {'x': 72, 'y': 73, 'c': 74}
L_HIP = {'x': 36, 'y': 37, 'c': 38}
R_HIP = {'x': 27, 'y': 28, 'c': 29}

# converts length of walkway in pixels to metres
SCALE_FACTOR = 4.0  # TODO need to also multiply by screen res


# sends the response to the PWA
def request_response(metrics):
    # allow CORS
    response = make_response()
    response.headers.add('Access-Control-Allow-Origin', '*')

    # set the data in the response to a JSON containing the metric values
    response.data = json.dumps(metrics)
    return response


# saves video and analyzes with OpenPose
@app.route('/vid-uploader', methods=['POST'])
def vid_upload_and_analysis():
    recording_name = 'test_conv'
    file_name = 'test'

    save_video_to_storage(file_name, recording_name)

    # move to the openpose directory
    os.chdir('./openpose')

```

```
    run_openpose(recording_name)

    # move to the json directory
    os.chdir('../videos-analyzed/' + recording_name)

    # acquire a list of all JSONs for the analysis
    json_names_list = [file for file in os.listdir(
        './') if file.endswith('.json')]

    # load the saved video into OpenCV
    video = cv2.VideoCapture('../../videos/' + recording_name + '.mp4')

    # analyse gait and print the result
    metrics, gait_trackers = analyse_gait_data(json_names_list, video)

    # display metrics and heel trackers
    print(metrics)
    print(gait_trackers)

    # output the frames where there is a heel strike
    render_heel_strike_images(gait_trackers, video)

    os.chdir('../../')
    return request_response(metrics)


# runs openpose on a video, outputting the analysed video and the JSON files
def run_openpose(recording_name):
    # run OpenPose on the video and save the result"""
    subprocess.run(['./bin/OpenPoseDemo.exe', '--video', '../videos/' + recording_name + '.mp4', '--write_json', '../videos-analyzed/' +
                    recording_name, '--display', '0', '--write_video', '../videos-analyzed/' + recording_name + '/' + recording_name + '.avi'])


# saves the video captured by the client to the server
def save_video_to_storage(recording_name, file_name):
    # get the video
    blob = request.files['video']

    # save the video and convert to mp4 using FFMPEG
    blob.save('./videos/' + recording_name + '.webm')
    subprocess.run(['./ffmpeg/bin/ffmpeg.exe', '-i', './videos/' + recording_name +
                    '.webm', '-vf', 'transpose=0', './videos/' + file_name + '.mp4'])
    os.remove('./videos/' + recording_name + '.webm')

    # make a directory for the JSONs
    os.mkdir('./videos-analyzed/' + recording_name)


# outputs frames where there is a heel strike to the JSON directory
```

```python
def render_heel_strike_images(gait_trackers, video):
    # track progress in the for loop
    index = 0

    # save each of the heel-strike frames as an image
    for frame_index in gait_trackers['toe_off_frames']:
        video.set(1, frame_index)
        ret, frame = video.read()
        cv2.imwrite('./' + str(frame_index) + '_' +
                    str(gait_trackers['toe_off_legs'][index]) + '.jpg', frame)
        index = index + 1


# calculates and returns gait metrics for the video
def analyse_gait_data(json_names_list, video):
    # current and previous BODY_25 poses
    current_pose: list[float]
    previous_pose: list[float]

    # gait metrics
    metrics = {
        'step_count': 0,
        'avg_swing_left': 0.0,
        'avg_stance_left': 0.0,
        'avg_swing_right': 0.0,
        'avg_stance_right': 0.0,
        'avg_swing_left_cent': 0.0,
        'avg_stance_left_cent': 0.0,
        'avg_swing_right_cent': 0.0,
        'avg_stance_right_cent': 0.0,
        'avg_double_support_left': 0.0,
        'avg_single_support_left': 0.0,
        'avg_double_support_right': 0.0,
        'avg_single_support_right': 0.0,
        'cadence': 0.0,
        'avg_stride_length': 0.0,
        'speeds': [],
        'avg_speed': 0.0
    }

    # dictionary items used internally to track gait
    gait_trackers = {
        'heel_strike': 'none',
        'toe_off': 'none',
        'heel_start': 0.0,
        'heel_strike_frames': [],
        'toe_off_frames': [],
        'heel_strike_legs': [],
        'toe_off_legs': [],
        'step_lengths': [],
        'first_step': True,
        'first_iteration': True,
        'frame': 0,
        'total_frames': 0,
```

```
            'duration_min': 0.0,
            'swing_left': [],
            'stance_left': [],
            'swing_right': [],
            'stance_right': [],
            'double_support_left': [],
            'single_support_left': [],
            'double_support_right': [],
            'single_support_right': []
        }

        for file_name in json_names_list:
            # open the next json in the list
            json_data = open(file_name)

            # deserialize into a dict
            de_json_data = json.load(json_data)

            if len(de_json_data['people']) != 0:
                if not gait_trackers['first_iteration']:
                    # previous array of joint points
                    previous_pose = current_pose

                    # current array of joint points
                    current_pose = de_json_data['people'][0]['pose_keypoints_2d']

                    # analyse gait for the frame
                    analyse_frame(previous_pose, current_pose,
                                  gait_trackers, metrics, gait_trackers['frame'])

                else:
                    # current array of joint points
                    current_pose = de_json_data['people'][0]['pose_keypoints_2d']

                    # notify the loop the first iteration has passed
                    gait_trackers['first_iteration'] = False
            else:
                print('no json data for frame ' + str(gait_trackers['frame']))

            # move onto next frame
            gait_trackers['frame'] += 1

        # get video duration in minutes
        calculate_video_duration(video, gait_trackers)

        # avg stride length rounded to 1 d.p.
        calculate_avg_step_length(gait_trackers, metrics)

        # cadence (steps per minute)
        calculate_cadence(gait_trackers, metrics)

        # speed (m/s)
        calculate_speeds(gait_trackers, metrics)

```

```python
    # swing/stance ratio and double support/single support
    calculate_swing_stance_ds_ss(gait_trackers, metrics)

    return metrics, gait_trackers


# generates an average for an array of n items
def avg_array(array):
    # variables to contain average
    avg = 0

    # calculate average
    for value in array:
        avg += value
    avg /= len(array)
    return avg


# returns swing and stance as percentages
def swing_stance_percent(swing, stance):
    total = swing + stance
    # round to 1d.p.
    swing_percent = np.round((swing / total) * 100, decimals=1)
    stance_percent = np.round((stance / total) * 100, decimals=1)
    return swing_percent, stance_percent


# converts a numbers of frames to a duration in seconds
def frames_to_sec(frames, total_frames, video_length_min):
    return (frames / total_frames) * video_length_min * 60


# calculates the swing stance ratio
def calculate_swing_stance_ds_ss(gait_trackers, metrics):
    # determine the amount of gait phase times we can calculate
    if len(gait_trackers['toe_off_frames']) < len(gait_trackers['heel_strike_frames']):
        length = len(gait_trackers['toe_off_frames'])
    else:
        length = len(gait_trackers['heel_strike_frames'])

    difference = 0
    first_event = 1
    # determine if heel strike happens before heel raise
    while difference == 0:
        difference = gait_trackers['toe_off_frames'][first_event] - \
            gait_trackers['heel_strike_frames'][first_event]
        first_event += 1

    strike_first = difference > 0

    for i in range(first_event, length - 1):
        # get name of legs that have just entered strike and raise
        strike_leg = gait_trackers['heel_strike_legs'][i]
```

```
        raise_leg = gait_trackers['toe_off_legs'][i]

        # if heel strike comes before heel raise
        if strike_first:
            # calculate swing and stance phase in frames
            swing_phase = gait_trackers['heel_strike_frames'][i +
                                                              1] - gait_trackers['toe_off_frames'][i]
            stance_phase = gait_trackers['toe_off_frames'][i +
                                                           1] - gait_trackers['heel_strike_frames'][i]

            # calculate double support in frames
            double_support = gait_trackers['toe_off_frames'][i] - \
                gait_trackers['heel_strike_frames'][i]

        # if heel raise comes before heel strike
        else:
            # calculate swing and stance phase in frames
            swing_phase = gait_trackers['heel_strike_frames'][i] - \
                gait_trackers['toe_off_frames'][i]
            stance_phase = gait_trackers['toe_off_frames'][i +
                                                           2] - gait_trackers['heel_strike_frames'][i]

            # calculate double support in frames
            double_support = gait_trackers['toe_off_frames'][i +
                                                             1] - gait_trackers['heel_strike_frames'][i]

        # calculate swing and stance phase in secs
        swing_phase_sec = frames_to_sec(
            swing_phase, gait_trackers['total_frames'], gait_trackers['duration_min'])
        stance_phase_sec = frames_to_sec(
            stance_phase, gait_trackers['total_frames'], gait_trackers['duration_min'])

        # append values to swing and stance array
        gait_trackers['swing_' + raise_leg].append(swing_phase_sec)
        gait_trackers['stance_' + strike_leg].append(stance_phase_sec)

        # calculate double support in seconds
        double_support_sec = frames_to_sec(
```

```
            double_support, gait_trackers['total_frames'], gait_trackers['
                duration_min'])

        # append double support
        gait_trackers['double_support_' +
                      strike_leg].append(double_support_sec)

        # single support is swing of opposite leg
        gait_trackers['single_support_' + strike_leg].append(swing_phase_sec)

        i += 1

    # calculate average swing/stance for left and right legs
    avg_swing_left = avg_array(gait_trackers['swing_left'])
    avg_stance_left = avg_array(gait_trackers['stance_left'])
    avg_swing_right = avg_array(gait_trackers['swing_right'])
    avg_stance_right = avg_array(gait_trackers['stance_right'])

    # round post calculation
    metrics['avg_swing_left'] = np.round(avg_swing_left, decimals=2)
    metrics['avg_stance_left'] = np.round(avg_stance_left, decimals=2)
    metrics['avg_swing_right'] = np.round(avg_swing_right, decimals=2)
    metrics['avg_stance_right'] = np.round(avg_stance_right, decimals=2)

    # calculate swing/stance percentage for left and right legs
    metrics['avg_swing_left_cent'], metrics['avg_stance_left_cent'] =
        swing_stance_percent(
        avg_swing_left, avg_stance_left)
    metrics['avg_swing_right_cent'], metrics['avg_stance_right_cent'] =
        swing_stance_percent(
        avg_swing_right, avg_stance_right)

    # calculate average double/single support for left and right legs
    metrics['avg_double_support_left'] = np.round(
        avg_array(gait_trackers['double_support_left']), decimals=2)
    metrics['avg_single_support_left'] = np.round(
        avg_array(gait_trackers['single_support_left']), decimals=2)
    metrics['avg_double_support_right'] = np.round(
        avg_array(gait_trackers['double_support_right']), decimals=2)
    metrics['avg_single_support_right'] = np.round(
        avg_array(gait_trackers['single_support_right']), decimals=2)


# calculates speed by step and average speed (both in m/s)
def calculate_speeds(gait_trackers, metrics):
    # track step number in the for loop
    index = 0

    # duration of the whole video
    vid_duration_sec = gait_trackers['duration_min'] * 60

    # use this variable to calculate average speed over the whole video
    speed_total = 0

```

```
    # frame at which the heel strike event starts
    frame_start = 0

    # calculate speed during each heel strike phase
    for distance in gait_trackers['step_lengths']:
        # frame at which the heel strike event ends
        frame_end = gait_trackers['heel_strike_frames'][index]

        if index != 0:
            distance_m = distance / 100

            # number of frames the heel strike phase occupies
            num_frames = frame_end - frame_start

            # duration of heel strike phase as a fraction of the total frames
                in the video
            step_duration = num_frames / gait_trackers['total_frames']

            # position in the video in seconds
            step_duration_sec = vid_duration_sec * step_duration

            # speed (m/s) for this heel strike phase
            speed_ms = distance_m / step_duration_sec

            # add speed to list of speeds and cumulative speed
            metrics['speeds'].append(np.round(speed_ms, decimals=2))
            speed_total += speed_ms

        index += 1
        frame_start = frame_end

    # calculate and return average speed for the whole video
    metrics['avg_speed'] = np.round(
        speed_total / len(metrics['speeds']), decimals=2)


# calculate steps per minute
def calculate_cadence(gait_trackers, metrics):
    metrics['cadence'] = metrics['step_count'] / gait_trackers['duration_min']


# gets video duration in minutes
def calculate_video_duration(video, gait_trackers):
    # Calculate the video duration in seconds and convert it to minutes
    total_frames = video.get(cv2.CAP_PROP_FRAME_COUNT)
    frame_rate = video.get(cv2.CAP_PROP_FPS)
    time_s = total_frames/frame_rate
    gait_trackers['duration_min'] = time_s / 60
    gait_trackers['total_frames'] = total_frames


# calculate the mean step length
def calculate_avg_step_length(gait_trackers, metrics):
    # find mean step length and round it to 1d.p.
```

```python
    avg_step_length = avg_array(gait_trackers['step_lengths'])
    metrics['avg_stride_length'] = np.round(avg_step_length, decimals=1)


# updates the metrics and gait trackers as analyse_gait_data iterates through frames
def analyse_frame(previous_pose, current_pose, gait_trackers, metrics, frame):
    # get the height of each heel
    l_heel_y = current_pose[L_HEEL['y']]
    r_heel_y = current_pose[R_HEEL['y']]

    # get the x position of each hip (current and previous)
    l_hip_x = current_pose[L_HIP['x']]
    r_hip_x = current_pose[R_HIP['x']]
    l_hip_x_prev = previous_pose[L_HIP['x']]
    r_hip_x_prev = previous_pose[R_HIP['x']]

    # get the x position of each heel (current and previous)
    l_heel_x = current_pose[L_HEEL['x']]
    r_heel_x = current_pose[R_HEEL['x']]
    l_heel_x_prev = previous_pose[L_HEEL['x']]
    r_heel_x_prev = previous_pose[R_HEEL['x']]

    # calculate the x distance travelled by left and right heels in relation to the hips
    l_hip_dif = l_hip_x - l_heel_x
    r_hip_dif = r_hip_x - r_heel_x
    l_hip_dif_prev = l_hip_x_prev - l_heel_x_prev
    r_hip_dif_prev = r_hip_x_prev - r_heel_x_prev

    # left heel strike
    if l_hip_dif > l_hip_dif_prev and (gait_trackers['toe_off'] == 'none' or gait_trackers['toe_off'] == 'left') and (gait_trackers['heel_strike'] == 'none' or gait_trackers['heel_strike'] == 'right') and l_heel_x > r_heel_x and l_heel_y > r_heel_y:
        handle_heel_strike(metrics, gait_trackers, frame,
                           l_heel_x, r_heel_x, 'left')

    # right heel strike
    elif r_hip_dif > r_hip_dif_prev and (gait_trackers['toe_off'] == 'none' or gait_trackers['toe_off'] == 'right') and (gait_trackers['heel_strike'] == 'none' or gait_trackers['heel_strike'] == 'left') and r_heel_x > l_heel_x and r_heel_y > l_heel_y:
        handle_heel_strike(metrics, gait_trackers, frame,
                           r_heel_x, l_heel_x, 'right')

    # right heel raise
    if r_hip_dif < r_hip_dif_prev and (gait_trackers['toe_off'] == 'none' or gait_trackers['toe_off'] == 'left') and gait_trackers['heel_strike'] == 'left' and r_heel_x < l_heel_x and r_heel_y < l_heel_y:
        handle_toe_off(metrics, gait_trackers, frame, 'right')

    # left heel raise
```

```python
    elif l_hip_dif < l_hip_dif_prev and (gait_trackers['toe_off'] == 'none' or gait_trackers['toe_off'] == 'right') and gait_trackers['heel_strike'] == 'right' and l_heel_x < r_heel_x and l_heel_y < r_heel_y:
        handle_toe_off(metrics, gait_trackers, frame, 'left')


# updates the metrics and gait trackers when a heel raise occurs
def handle_toe_off(metrics, gait_trackers, frame, heel_name):
    # update the gait trackers with the new heel raise event data
    gait_trackers['toe_off_frames'].append(frame)
    gait_trackers['toe_off_legs'].append(heel_name)

    # Set heel lift to new foot
    gait_trackers['toe_off'] = heel_name


# updates the metrics and gait trackers when a heel strike occurs
def handle_heel_strike(metrics, gait_trackers, frame, heel_x, other_heel_x, heel_name):
    # update the gait trackers and metrics with the new heel strike event data
    metrics['step_count'] = metrics['step_count'] + 1
    gait_trackers['heel_strike_frames'].append(frame)
    gait_trackers['heel_strike_legs'].append(heel_name)

    # distance counting
    if not gait_trackers['first_step']:
        distance = -(other_heel_x - gait_trackers['heel_start'])
        distance_m = distance / SCALE_FACTOR
        gait_trackers['step_lengths'].append(distance_m)
    else:
        # variable to store whether the first step has been taken
        gait_trackers['first_step'] = False

    # variable to store the location of the heel when it first hits the floor
    gait_trackers['heel_start'] = heel_x

    # Set heel strike to new foot
    gait_trackers['heel_strike'] = heel_name


# run the Flask server
if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

B.2 Version 1.0 Webcam Class

Below is the code for the **Webcam** class.

```javascript
import { useRef } from "react";
```

```
import { useEffect } from "react";

export default function Webcam() {
    const video=useRef(null);

    //UseEffect hook allows us to activate the navigator
    useEffect(()=> {
        //Properties for the webcam stream
        const vidProperties = {audio:false, video:{width:1920, height:1080}};

        GetWebcamStream(navigator,video,vidProperties);
    }, [])

    //Return the video object, creating a reference to it for the webcam to use
    return <video className="w-full" autoPlay ref={video} />
}

/* Attaches the webcam feed to a reference of the video object */
function GetWebcamStream(navigator,video,vidProperties) {
    //Once we have acquired the webcam, attatch it's stream to the video object
    navigator.mediaDevices.getUserMedia(vidProperties).then((stream)=>{
        video.current.srcObject = stream;
    });
}
```

B.3 Version 1.0 WebcamRecorder Class

Below is the code for the **WebcamRecorder** class.

```
import { useRef } from "react";
import { useEffect } from "react";
import { useRouter } from "next/router"

let shouldStop;
let stopped;

export default function WebcamRecorder() {
    const video=useRef(null);
    const router=useRouter();

    //Both variables describe the state of video capture
    shouldStop = false;
    stopped = false;

    //UseEffect hook allows us to activate the navigator
```

```
    useEffect(()=> {
        //Properties for the webcam stream
        const vidProperties = {audio:false, video:{width:1920, height:1080}};

        //When stop is clicked, the media recorder is told to stop
        const stopButton = document.getElementById("stopButton");
        stopButton.addEventListener('click',()=>{
            shouldStop=true;
        });

        GetWebcamStream(navigator,video,vidProperties,router);
    }, [])

    //Return the video object, creating a reference to it for the webcam to
        use
    return <video className="w-full" autoPlay ref={video} />
}

/* Attaches the webcam feed to a reference of the video object */
function GetWebcamStream(navigator,video,vidProperties,router) {
        //Acquire the webcam
        navigator.mediaDevices.getUserMedia(vidProperties).then((stream)=>{
            //Set video format to webm
            const mediaRecorderOptions={mimeType:'video/webm'};

            //Create a media recorder to record the video
            const mediaRecorder = new MediaRecorder(stream,
                mediaRecorderOptions);

            //Create an array to save the video frames in
            const recordedChunks=[];

            mediaRecorder.ondataavailable=function (e){
                if (e.data.size > 0) {
                    //Add image to array
                    recordedChunks.push(e.data);
                }
                if (shouldStop === true && stopped === false) {
                    mediaRecorder.stop();
                    stopped = true;
                }
            };

            mediaRecorder.onstop = function() {
                //Get the http request for the OpenPose server ready
                var xmlhttp = new XMLHttpRequest();

                xmlhttp.onreadystatechange = ()=>{
                    if (xmlhttp.readyState == XMLHttpRequest.DONE) {
                        const json = JSON.parse(xmlhttp.responseText)
                        console.log(json);
                        //Change to the post assessment screen
                        router.push({pathname: '/assessment/post-assessment',
                            query: json});
```

```
                        }

                    }

                    xmlhttp.open("POST", 'http://localhost:5000/vid-uploader');

                    //Create blob with the video data
                    const blob = new Blob(recordedChunks);

                    //Save the blob to a form
                    var fd=new FormData();
                    fd.append("video", blob, "video.webm");

                    //Send the form to the OpenPose server using the http request
                    xmlhttp.send(fd);

                    //Go to the analyzing page for now
                    router.push('/assessment/assessment-analyzing');
                };

                //Assign stream to video HTML element
                video.current.srcObject = stream;

                //Set FPS to 30
                mediaRecorder.start(33.333);
            });
}
```

B.4 Version 2.0 PoseEstimator Class

Below is the code for the Generic **PoseEstimator** class.

```
import * as MediaPipe from '@mediapipe/holistic'
import * as THREE from 'three'
import { Vector3 } from 'three';
import { loadMaterial } from '../game/material-loader';

/**
 * Uses a pose estimator to render and return a 3D avatar
 */
export default class PoseEstimator{
    constructor(scene, camera){
        /* Each array contains a list of keypoints
           The keypoints have the following format: {coordinates:THREE.Vector3
              , confidence:number} */
        this.body = [];
        this.face = [];
```

```
        this.handL = [];
        this.handR = [];

        //Track whether the model has been loaded
        this.isInitialized = false;

        //False for testing mode, true for deployment mode
        this.useWebcam = true;

        //Lists containing the meshes to build the avatar
        this.avatarBones = [];
        this.avatarJoints = [];
        this.avatarHead = new THREE.Mesh();
        this.boneMaterial = new THREE.MeshStandardMaterial({
            color: 0x212121,
            roughness:0.262,
            metalness:0.1
        });
        this.headMaterial = new THREE.MeshStandardMaterial({
            color: 0x212121,
            roughness:0.262,
            metalness:0.1
        });
        this.jointMaterial = new THREE.MeshStandardMaterial({
            color: 0xFFFFFF,
            roughness:0.262,
            metalness:0.1
        });

        //Indexes to draw the spheres for joints (and the head)
        this.jointIndexes = [11,12,13,14,23,24,25,26];
        this.spineIndexes = {hip:[23,24],neck:[11,12]};
        this.headIndex = 0;

        //Reference to the THREE scene and camera
        this.scene = scene;
        this.camera = camera;
        this.camera.position.y = -0.6;

        //8 point moving average filter for Yoke
        this.avatarDistanceChangeBuffer = [];
        this.avatarDistanceChangeBuffer.length = 8;
        this.avatarDistanceChangeBuffer.fill(0,0,7);

        //Variables for realtime metric calculation
        this.avatarDistance = 0.0;
        this.avatarDistance_m = 0.0;
        this.avgCurrent_m = 0.0;
        this.avgCurrentUpdated = false;
        this.lastTime = 0.0;
        this.scaleFactor = 2;
        this.steps = 0;
        this.lerpValue = 0.6;

```

```
        //Track specific joints for Yoke calculations
        this.leftAnkle = 0.0;
        this.rightAnkle = 0.0;
        this.leftAnklePrev = 0.0;
        this.rightAnklePrev = 0.0;
        this.leftHip = 0.0;
        this.leftHipPrev = 0.0;
        this.rightHip = 0.0;
        this.rightHipPrev = 0.0;

        //Track which heel is currently in heel strike phase
        this.heelStrike = 'none';

        //True for the first frame of a new heel strike
        this.heelChange = false;
    }

    /**
     *  Starts the pose estimator (*make sure to specify the trackingType*)
     */
    async StartTracking(){
        throw new Error('Start tracking method not implemented');
    }

    /* Assigns the x,y,z and confidence values to the Sensor */
    assignPose(){
        throw new Error('Assign pose method not implemented');
    }

    /**
     *  Builds the avatar in the way specified by tracking type
     */
    buildAvatar(){
        switch (this.trackingType){
            //Body is just plain body tracking
            case 'body':
                this.buildAvatarBody();
                break;
            default:
                throw new Error('Unrecognized tracking type, please assign in
                    the startTracking function');
                break;
        }
    }

    /**
     *  Updates the avatar in the way specified by tracking type
     */
    updateAvatar(){
        this.updateAvatarBody();
    }

    colourToHex(colour){
        switch(colour){
```

```
            case "Red":
                return 0xFF0000;
            case "Orange":
                return 0xFFA500;
            case "Yellow":
                return 0xFFFF00;
            case "Green":
                return 0x00FF00;
            case "Blue":
                return 0x0000FF;
            case "Purple":
                return 0x6A0DAD;
            case "Black":
                return 0x000000;
            case "White":
                return 0xFFFFFF;
            default:
                break;
        }
    }

    /**
     * Changes the colour of the avatar's joints, head and bones
     */
    updateJointColour(jointColour){
        const colour = this.colourToHex(jointColour.target.value);

        //Update bone material
        this.jointMaterial.color.setHex(colour);
    }

    updateBoneColour(boneColour){
        const colour = this.colourToHex(boneColour.target.value);

        //Update bone material
        this.boneMaterial.color.setHex(colour);
    }

    updateHeadColour(headColour){
        const colour = this.colourToHex(headColour.target.value);

        //Update head material
        this.headMaterial.color.setHex(colour);
    }

    /**
     *  Builds the avatar's body as a THREE mesh
     */
    buildAvatarBody(){
        this.buildAvatarBones();
        this.buildAvatarJoints();
        this.buildAvatarHead();
    }

```

```
    /**
     * Builds the avatar's head as a THREE sphere
     */
    buildAvatarHead(){
        //Make the head (a sphere)
        const geometry = new THREE.SphereGeometry( 0.1, 16, 16 );
        this.avatarHead = new THREE.Mesh( geometry, this.headMaterial );

        //Add the head to the scene
        this.scene.add(this.avatarHead);
    }

    /**
     *  Builds the avatar's bones as THREE cylinders and assigns them to the avatarBones variable
     */
     buildAvatarBones(){
        //POSE_CONNECTIONS is an array describing the connections between joints
        for(let i=0; i<25; i++){
            //Make a bone (cylinder)
            const geometry = new THREE.CylinderGeometry( 0.003, 0.002, 20, 32 );
            const cylinder = new THREE.Mesh( geometry, this.boneMaterial );

            //Add the bone to the array of bones and to the THREE js scene
            this.avatarBones[i] = cylinder;
            this.scene.add(this.avatarBones[i]);
        }
    }

    /**
     *  Builds the avatar's joints as THREE spheres and assigns them to the avatarJoints variable
     */
    buildAvatarJoints(){
        //Create the joint connectors and add them to the scene
        for(let i=0; i<11; i++){
            //Make a joint (sphere)
            const geometry = new THREE.SphereGeometry( 0.005, 32, 16 );
            const sphere = new THREE.Mesh( geometry, this.jointMaterial );

            //Add the joint to the array of joints and to the THREE js scene
            this.avatarJoints[i] = sphere;
            this.scene.add(this.avatarJoints[i]);
        }
    }

    /**
     * Updates the avatar in the THREE scene
     */
    updateAvatarBody(){
        //Calculate Yoke
        this.calculateDistanceWalked();
```

```

        //Update camera using Yoke
        this.camera.position.z = -0.7 + this.avatarDistance;

        //Update avatar
        this.updateAvatarBones();
        this.updateAvatarJoints();
    }

    /**
     * Updates the position, length and rotation of the avatar's bones
     */
    updateAvatarBones(){
        var i = 0;

        //POSE_CONNECTIONS is an array describing the connections between joints
        MediaPipe.POSE_CONNECTIONS.forEach(element =>{
            if(((element[0] == this.spineIndexes.neck[0]) && (element[1] == this.spineIndexes.hip[0])) || ((element[0] == this.spineIndexes.neck[1]) && (element[1] == this.spineIndexes.hip[1]))){
            }
            else{
                if(element[0] <= 11 && element[1] <= 11){
                }
                else{
                    //Get the coordinates of the joint and its child
                    const joint1 = new THREE.Vector3();
                    const joint2 = new THREE.Vector3();
                    joint1.copy(this.body[element[0]].coordinates);
                    joint2.copy(this.body[element[1]].coordinates);
                    this.updateSingleBone([joint1,joint2],i);

                    //Only render bones which have a high confidence value
                    if(this.body[element[0]].confidence < 0.2 || this.body[element[1]].confidence < 0.2){
                        this.avatarBones[i].visible = false;
                    }
                    else{
                        this.avatarBones[i].visible = true;
                    }
                    i++;
                }
            }
        });

        //Get coordinates necessary to calculate spine bone
        const leftHip_v = this.body[this.spineIndexes.hip[0]];
        const rightHip_v = this.body[this.spineIndexes.hip[1]];
        const leftShoulder_v = this.body[this.spineIndexes.neck[0]];
        const rightShoulder_v = this.body[this.spineIndexes.neck[1]];

```

```
        //Create a fake spine bone based on midpoint between shoulders and hips
        const spineBase = new Vector3();
        const spineTop = new Vector3();
        spineBase.copy(this.calculateMidpointVector(leftHip_v.coordinates, rightHip_v.coordinates));
        spineTop.copy(this.calculateMidpointVector(leftShoulder_v.coordinates, rightShoulder_v.coordinates));
        this.updateSingleBone([spineBase, spineTop], i);

        //Only render bones which have a high confidence value
        if(leftHip_v.confidence < 0.2 || rightHip_v.confidence < 0.2 || leftShoulder_v < 0.2 || rightShoulder_v < 0.2){
            this.avatarBones[i].visible = false;
        }
        else{
            this.avatarBones[i].visible = true;
        }
    }

    /**
     * Update a bone's position, rotation, and length
     */
    updateSingleBone(jointPair, index){

        //Calculate the midpoint between the two joints and assign this as the bone's location
        this.avatarBones[index].position.copy(this.calculateMidpointVector(jointPair[0], jointPair[1]));

        //Calculate the bone's new length and update it
        const boneLength = this.calculateBoneLength(jointPair[0], jointPair[1]);
        this.avatarBones[index].geometry = new THREE.CylinderGeometry( 0.002, 0.002, boneLength, 32 );

        //Calculate the bone's direction vector and use it to update its rotation
        const up = new THREE.Vector3(0,-1,0);
        this.avatarBones[index].quaternion.setFromUnitVectors(up, this.calculateBoneDirection(jointPair[0], jointPair[1]).normalize());
    }

    /**
     * Updates the position of the spheres representing joints
     */
    updateAvatarJoints(){
        var i=0;

        //Update standard joints
        this.jointIndexes.forEach((element)=>{
            this.updateSingleJoint(element,i);
            i++;
        });
```

```
            //Calculate location of fake hip, neck and mid spine joints
            const hip = this.calculateMidpointVector(this.body[this.spineIndexes.
                hip[1]].coordinates,this.body[this.spineIndexes.hip[0]].
                coordinates);
            const neck = this.calculateMidpointVector(this.body[this.spineIndexes.
                neck[1]].coordinates,this.body[this.spineIndexes.neck[0]].
                coordinates);
            const spineMid = this.calculateMidpointVector(hip,neck);

            //Change position of fake joints
            this.avatarJoints[i].position.copy(hip);
            this.avatarJoints[i+1].position.copy(neck);
            this.avatarJoints[i+2].position.copy(spineMid);

            //Calculate size of head and update it
            const headRadius = this.calculateBoneLength(this.body[this.headIndex].
                coordinates,neck)/5;
            const headPosition = new THREE.Vector3(this.body[this.headIndex].
                coordinates.x, this.body[this.headIndex].coordinates.y, neck.z);
            this.updateAvatarHead(headRadius, headPosition);
        }

        /**
         * Update the position of each joint
         */
        updateSingleJoint(jointIndex,index){
            const joint = new THREE.Vector3();
            joint.copy(this.body[jointIndex].coordinates);
            this.avatarJoints[index].position.copy(joint);
        }

        /**
         * Updates the position and size of the avatar's head
         */
        updateAvatarHead(radius,position){
            //Update the size of the head sphere
            this.avatarHead.geometry = new THREE.SphereGeometry( radius, 32, 16 );

            //Update the head sphere's position
            this.avatarHead.position.copy(position);
        }

        /**
         * Calculates Yoke based on the ankle and
         */
        calculateDistanceWalked(){
            //Set previous positions to last current positions
            this.leftAnklePrev = this.leftAnkle;
            this.rightAnklePrev = this.rightAnkle;
            this.leftHipPrev = this.leftHip;
            this.rightHipPrev = this.rightHip;

            //Set current positions
```

```
        this.leftAnkle = this.body[MediaPipe.POSE_LANDMARKS_LEFT.LEFT_ANKLE].coordinates.z;
        this.rightAnkle = this.body[MediaPipe.POSE_LANDMARKS_RIGHT.RIGHT_ANKLE].coordinates.z;
        this.leftHip = this.body[MediaPipe.POSE_LANDMARKS.LEFT_HIP].coordinates.z;
        this.rightHip = this.body[MediaPipe.POSE_LANDMARKS.RIGHT_HIP].coordinates.z;

        //Get difference between leg and hip for previous and current positions
        const leftHipDif = this.leftHip - this.leftAnkle;
        const rightHipDif = this.rightHip - this.rightAnkle;
        const leftHipDifPrev = this.leftHipPrev - this.leftAnklePrev;
        const rightHipDifPrev = this.rightHipPrev - this.rightAnklePrev;

        //Detect left heel strike
        if (leftHipDif > leftHipDifPrev && (this.heelStrike == "none" || this.heelStrike == "right") && this.leftAnkle > this.rightAnkle){
            this.heelStrike = 'left';
            this.heelChange = true;
            this.steps++;
        }
        //Detect right heel strike
        else if (rightHipDif > rightHipDifPrev && (this.heelStrike == "none" || this.heelStrike == "left") && this.rightAnkle > this.leftAnkle){
            this.heelStrike = 'right';
            this.heelChange = true;
            this.steps++;
        }

        //Exclude erroneous events (left foot) - assume we are always walking forward
        if (this.heelStrike == "left"){
            const distance = this.leftAnklePrev - this.leftAnkle;
            if (distance > 0){
                //Add distance to the queue
                this.avatarDistanceChangeBuffer.unshift((distance + this.avatarDistanceChangeBuffer[0]) * this.lerpValue);
            }
            else{
                //0 used to replace erroneous values
                this.avatarDistanceChangeBuffer.unshift((0.0));
            }
            //Dequeue one distance value to maintain buffer size
            this.avatarDistanceChangeBuffer.pop();
        }
        //Exclude erroneous events (right foot) - assume we are always walking forward
        else if (this.heelStrike == "right"){
            const distance = this.rightAnklePrev - this.rightAnkle;
            if (distance > 0){
                //Add distance to the queue
```

```
                this.avatarDistanceChangeBuffer.unshift((distance + this.
                    avatarDistanceChangeBuffer[0]) * this.lerpValue);
            }
            else{
                //0 used to replace erroneous values
                this.avatarDistanceChangeBuffer.unshift((0.0));
            }
            //Dequeue one distance value to maintain buffer size
            this.avatarDistanceChangeBuffer.pop();
        }

        //Read average distance from the buffer
        let avg = 0.0;
        this.avatarDistanceChangeBuffer.forEach((element)=>{
            avg += element;
        });
        avg /= this.avatarDistanceChangeBuffer.length;

        //Update distance away from camera
        this.avatarDistance += avg;
        this.avatarDistance_m = Math.round(this.avatarDistance * 2 * 10)/10;
        this.avgCurrent_m = Math.round(avg * 2 * 10)/10;
        this.avgCurrentUpdated = false;

        //Reset variable
        this.heelChange = false;
    }

    /**
     * Calculates the length of a bone given two joints
     */
    calculateBoneLength(v1, v2){
        const v3 = new THREE.Vector3();
        v3.copy(v1);
        const distance = v3.distanceTo(v2);
        return distance;
    }

    /**
     * Calculates where to place the bone in the scene
     */
    calculateMidpointVector(v1, v2){
        const location = new THREE.Vector3();
        //Find location by getting the point midway between two joints
        location.addVectors(v1,v2);
        location.divideScalar(2);
        return location;
    }

    /**
     * Calculates the direction vector between two joints
     */
    calculateBoneDirection(v1, v2){
        const v3 = new THREE.Vector3();
```

```
        v3.copy(v1);
        v3.sub(v2);
        return v3;
    }
}
```

B.5 Version 2.0 MediaPipeHolistic Class

Below is the code for the **MediaPipeHolistic** class.

```
import * as MediaPipe from '@mediapipe/holistic'
import * as CameraUtils from '@mediapipe/camera_utils'
import * as THREE from 'three'
import PoseEstimator from './pose-estimator';
import Webcam from '../data-capture/webcam';
import { useRef } from 'react';

//Config file TODO: implement config locally
const config = {locateFile: (file) => {
  return `https://cdn.jsdelivr.net/npm/@mediapipe/holistic/${file}`;
}};

let holistic;
let video;
var switchingCams = false

/**
 * Uses MediaPipe's hollistic tracking module to track body, facial landmarks
     and hand movements
 */
export default class MediapipeHolistic extends PoseEstimator{
    async StartTracking(id){
      //Get the video from the HTML document
      video = document.getElementById("input_video");

      console.log(id);

      //For patients
      if(this.useWebcam){
        const webcam = new Webcam(navigator,video);
        await webcam.ChangeDeviceById(id);
      }

      //Set tracking type to full body for now, face + hands coming in future
      this.trackingType = 'body';

      holistic = new MediaPipe.Holistic({locateFile: (file) => {
```

```
      return `https://cdn.jsdelivr.net/npm/@mediapipe/holistic/${file}`;
    }});

    //Set up holistic tracking
    holistic.setOptions({
      modelComplexity: 2,
      smoothLandmarks: true,
      smoothSegmentation: true,
      refineFaceLandmarks: true,
      minDetectionConfidence: 0.5,
      minTrackingConfidence: 0.5
    });

    console.log("yeet");

    //Start tracking
    holistic.initialize().then(()=>{this.getVideoFrame()});

    //Add the avatar to the THREE scene
    this.buildAvatar();

    //Event fires when holistic has completed its analysis
    holistic.onResults((results)=> {
        this.assignPose(results);
        this.updateAvatar();
    })

    console.log("initialized")
  }

  async OnChangeCamera(value,webcam){
    this.switchingCams = true;
    await webcam.ChangeDevice(value);
    this.switchingCams = false;
    //await holistic.reset();
  }

  /* Gets current video frame and sends it for analysis */
  async getVideoFrame(){
    window.requestAnimationFrame(()=> {this.getVideoFrame()});
    if(!this.switchingCams){
      console.log("yeet");
      await holistic.send({image:video});
      if(!this.isInitialized){
        this.isInitialized = true;
      }
    }
  }

  /**
   *  Assigns the x,y,z and confidence values to the generic Sensor
   */
  assignPose(results){
    var i=0;
```

```
        if(results.poseLandmarks){
          //Assign values for body
          results.poseLandmarks.forEach(element => {
            this.body[i] = { coordinates: new THREE.Vector3(-element.x, -element
                .y, -element.z + this.avatarDistance), confidence: element.
                visibility}
            i++
          });
          i=0;
        }

        if(results.faceLandmarks){
          //Assign values for face
          results.faceLandmarks.forEach(element => {
            this.face[i] = { coordinates: new THREE.Vector3(element.x, element.y
                , element.z + this.avatarDistance), confidence: -1}
            i++
          });
          i=0;
        }

        if(results.leftHandLandmarks){
          //Assign values for left hand
          results.leftHandLandmarks.forEach(element => {
            this.handL[i] = { coordinates: new THREE.Vector3(element.x, element.
                y, element.z + this.avatarDistance), confidence: -1}
            i++
          });
          i=0;
        }

        if(results.rightHandLandmarks){
          //Assign values for right hand
          results.rightHandLandmarks.forEach(element => {
            this.handR[i] = { coordinates: new THREE.Vector3(element.x, element.
                y, element.z + this.avatarDistance), confidence: -1}
            i++
          });
        }
      }
}
```

B.6 Version 2.0 World Class

Below is the code for the **World** class.

```
import * as THREE from 'three';
import { loadMaterial } from './material-loader';

```

```
/**
 * Renders an environment in the provided scene. This includes ground and a
    sky.
 */
export default class World{
    constructor(scene, camera){
        //References to three scene & camera
        this.scene = scene;
        this.camera = camera;

        //Location of different textures
        this.floorFilepath = "./images/textures/ground/terracotta";
        this.skyFilepath = "./images/textures/sky/cloudy";

        //Material for floor
        this.floorMaterial = loadMaterial(this.floorFilepath, 8, true);
        this.skyMaterial = loadMaterial(this.skyFilepath,2,false);

        //Add elements of environment to scene
        this.addFloor();
        this.addLighting();
        this.addSky();

    }

    /**
     * Changes the floor material
     */
    setFloorMaterial(materialName){
        //Update filepath based on material name
        switch(materialName.target.value){
            case "Rock":
                this.floorFilepath = "./images/textures/ground/rock_stylized";
                break;
            case "Moss":
                this.floorFilepath = "./images/textures/ground/rock_moss";
                break;
            case "Terracotta":
                this.floorFilepath = "./images/textures/ground/terracotta";
                break;
            default:
                break;
        }

        //Update the floor material with the new filepath
        this.floorMaterial.copy(loadMaterial(this.floorFilepath, 8, true));
    }

    /**
     * Changes the sky material
     */
     setSkyMaterial(materialName){
        //Update filepath based on material name
        switch(materialName.target.value){
```

```
            case "Cloudy":
                this.skyFilepath = "./images/textures/sky/cloudy";
                break;
            case "Overcast":
                this.skyFilepath = "./images/textures/sky/overcast";
                break;
            case "Night":
                this.skyFilepath = "./images/textures/sky/night";
                break;
            default:
                break;
        }

        //Update the sky material with the new filepath
        this.skyMaterial.copy(loadMaterial(this.skyFilepath, 2, false));
    }

    /**
     * Changes the sky material
     */

    /**
     * Adds a sphere with a sky texture to the scene
     */
    addSky(){
        //Create a sky dome
        const geometry = new THREE.SphereGeometry(30,32,32);

        const sphere = new THREE.Mesh(geometry,this.skyMaterial);
        sphere.position.z = 15;
        sphere.position.y = -15;

        this.scene.add(sphere);
    }

    /**
     * Adds lighting to the scene
     */
    addLighting(){
        //Add ambient lighting
        const light = new THREE.HemisphereLight( 0xffffff, 0xffffff, 1.2 );
        this.scene.add(light);
    }

    /**
     * Adds a plane with a specified texture for the avatar to walk on
     */
    addFloor(){
        //Create floor geometry
        const geometry = new THREE.PlaneBufferGeometry(10,20,512,512);

        //Create and position floor
        this.floor = new THREE.Mesh(geometry,this.floorMaterial);
        this.floor.rotation.x = Math.PI/2;
```

```
        this.floor.position.y = -1;
        this.floor.position.z = 5;

        //Add floor to the THREE scene
        this.scene.add(this.floor);
    }
}
```

B.7 Version 3.0 Game Class

Below is the code for the **Game** class.

```
import MediapipePose from '../data-processing/mediapipe-pose';
import World from './world';
import { Vector3 } from 'three';
import ColourDistractor from './colour-distractor';
import AlertHandler from '../../pages/training/game-feed/alert-handler';
import NLP from './nlp';
import id from '../../pages/training/id'
import Webcam from '../data-capture/webcam';
import MetricsCalculator from '../data-processing/metrics-calculator';
import Grapher from './grapher';

/** Renders an environment in the provided scene. This includes ground and a
    sky. */
 export default class Game{
    constructor(scene,camera,renderer,query,setGameActive){
        this.scene = scene;
        this.camera = camera;
        this.query = query;
        this.renderer = renderer;
        this.setGameActive = setGameActive;

        //Initialise the avatar and pass it the THREE scene
        this.avatar = new MediapipePose(this);
        this.world = new World(this);
        this.colourDistractor = new ColourDistractor(this);
        this.alertHandler = new AlertHandler(3000,1000,['hey','yo','konichiwa'
            ,'guten tag']);
        this.NLP = new NLP(this);
        this.grapher = new Grapher(this);

        //Variables describing game state
        this.sessionTime = 0.0;
        this.recordTime = 10.0;
        this.shouldStopRecording = false;
        this.recording = false;
        this.displayTick = false;
        this.score = 0;
```

```
        this.lastScore = 0;
        this.attempts = 0;
        this.lastAttempts = 0;

        const gaitFeed = document.getElementById(id.Training.GaitFeed);
        this.webcam = new Webcam(navigator,gaitFeed);
    }

    /** Get rid of the tick in the game UI */
    clearCorrect(){
        this.displayTick = false;
    }

    /** Initializes the game (but doesnt begin recording) */
    async init(){
        //Place the camera behind the avatar
        this.camera.position.copy(new Vector3(-0.55,-0.5,-1));
        this.camera.rotation.y = Math.PI;

        //Start avatar tracking
        await this.avatar.startTracking(this.query.bioCam);
    }

    /** Checks an answer from the NLP vs the actual answer from the colour
        distractor */
    checkAnswer(answer){
        if(this.recording){
            if(answer == this.colourDistractor.currentAnswer && this.
                sessionTime < (this.recordTime-(this.colourDistractor.timeGap
                /1000))){
                //Add to the score and display the tick for 1 second
                this.score++;
                this.displayTick = true;
                window.setTimeout(()=>{this.clearCorrect()},1000);
            }
        }
    }

    /** Starts recording metrics, scores etc. */
    async startRecording(){
        this.sessionTime = 0.0;
        this.avatar.metrics = new MetricsCalculator(this);
        await this.webcam.changeDeviceById(this.query.gaitCam);
        //await this.webcam.startRecording();
        this.recording = true;
        this.sessionTimer();
    }

    /** 0.1s interval timer */
    sessionTimer(){
        if(this.recording){
            this.sessionTime = Math.round((this.avatar.metrics.realTimeMetrics
                .time_s)*10)/10;
            window.setTimeout(()=>{this.sessionTimer()},100);
```

```
        }
    }

    stopRecording(){
        //Keep a record of the last score and the last number of attempts
        this.lastScore = this.score;
        this.lastAttempts = this.attempts;

        //Reset score, attempts,session time and metrics
        this.score = 0;
        this.attempts = 0;

        //Stop recording
        //this.webcam.stopRecording();
        this.recording = false;
        this.shouldStopRecording = true;
        this.avatar.metrics.output();
        this.grapher.graph();
        this.sessionTime = this.avatar.metrics.realTimeMetrics.time_s;
    }

    /** Renders each frame of the game */
    update(){
        //Render THREE scene
        this.renderer.render( this.scene, this.camera );
    }


    /** Sets total test time from GCS */
    setTestTime(time_m){
        this.recordTime = time_m.target.value * 60;
    }

    /** Updates any property within the game from the GCS */
    updateProperty(name,value){
        switch(name){
            case "avatar bone colour":
                this.avatar.updateBoneColour(value);
                break;
            case "avatar head colour":
                this.avatar.updateHeadColour(value);
                break;
            case "avatar joint colour":
                this.avatar.updateJointColour(value);
                break;
            case "world floor material":
                this.world.setFloorMaterial(value);
                break;
            case "world sky material":
                this.world.setSkyMaterial(value);
                break;
            case "world sky material":
                this.world.setSkyMaterial(value);
                break;
```

```
            case "interactions cognitive distractor":
                this.colourDistractor.setDistractorActive(value);
                value.target.value == "Yes" ?  this.setGameActive(true) : this
                    .setGameActive(false);
                break;
            case "interactions guidance":
                this.alertHandler.setAlertsActive(value);
                break;
            case "interactions distractor speed":
                this.colourDistractor.setDistractorSpeed(value);
            case "interactions test time":
                this.setTestTime(value);
                break;
            case "graph selection":
                this.grapher.graph(value.target.value);
                break;
            default:
                break;
        }
    }
}
```

B.8 Version 3.0 MediapipePose Class

Below is the code for the **MediapipePose** class.

```
import * as MediaPipe from '@mediapipe/pose'
import * as THREE from 'three'
import PoseEstimator from './pose-estimator';
import Webcam from '../data-capture/webcam';
import id from '../../pages/training/id';
import { POSE_LANDMARKS_LEFT,POSE_LANDMARKS_RIGHT } from "@mediapipe/pose";

/** Uses MediaPipe's hollistic tracking module to track body, facial landmarks
    and hand movements */
export default class MediapipePose extends PoseEstimator{
  constructor(game){
    super(game);
    this.pose = new MediaPipe.Pose({locateFile: (file) => {
      return `https://cdn.jsdelivr.net/npm/@mediapipe/pose@0.5/${file}`;
    }});
    this.video = document.getElementById(id.Training.BiofeedbackCam);
    this.useWebcam = false;
  }

  async startTracking(id){
    //For patients
```

```
    if(this.useWebcam){
      const webcam = new Webcam(navigator,this.video);
      await webcam.changeDeviceById(id);
    }

    //Set tracking type to full body for now, face + hands coming in future
    this.trackingType = 'body';

    //Set up holistic tracking
    this.pose.setOptions({
      modelComplexity: 2,
      smoothLandmarks: true,
      enableSegmentation: false,
      smoothSegmentation: true,
      minDetectionConfidence: 0.5,
      minTrackingConfidence: 0.5
    });

    this.getVideoFrame();

    //Add the avatar to the THREE scene
    this.buildAvatar();

    //Event fires when holistic has completed its analysis
    this.pose.onResults(async (results)=>{
      await this.assignPose(results);
      await this.updateAvatar();
    })
  }

  async OnChangeCamera(value,webcam){
    this.switchingCams = true;
    await webcam.changeDevice(value);
    this.switchingCams = false;
    //await holistic.reset();
  }

  /** Gets current video frame and sends it for analysis */
  async getVideoFrame(){
    if(!this.switchingCams){
      await this.pose.send({image:this.video});
      if(!this.isInitialized){
        this.isInitialized = true;
      }
    }
    window.requestAnimationFrame(()=> {this.getVideoFrame()});
  }

  /** Assigns the x,y,z and confidence values to the generic Sensor with
      scaling and offset */
  async assignPose(results){
    var i=0;
    if(results.poseLandmarks){
      this.calulateHeightAndOffset(results);
```

```
      if(this.game.recording){
        this.metrics.updateMetrics(results)
      }

      //Set results with offset and scaling
      results.poseLandmarks.forEach(element => {
        this.body[i] = { coordinates: new THREE.Vector3(element.x-1.05,
            (0.6*((element.y-this.avatarOffsetY)/this.avatarHeight))-0.7, -
            element.z + this.metrics.realTimeMetrics.avgDistance), confidence:
             element.visibility}
        i++
      });
    }
  }

  /** Stops the game from recording */
  stopRecording(){
    this.metrics.reset();
  }

  /** Calculates the scaling parameters for the avatar */
  calulateHeightAndOffset(results){
    //Get body parts needed for scaling calculations
    const rightAnkle = results.poseLandmarks[MediaPipe.POSE_LANDMARKS_RIGHT.
        RIGHT_ANKLE].y;
    const leftAnkle = results.poseLandmarks[MediaPipe.POSE_LANDMARKS_LEFT.
        LEFT_ANKLE].y;
    const nose = results.poseLandmarks[MediaPipe.POSE_LANDMARKS_NEUTRAL.NOSE].
        y;

    //If left ankle on the ground
    if(leftAnkle<rightAnkle){
      this.avatarOffsetY = leftAnkle;
      this.avatarHeight = nose-leftAnkle;
    }
    //If right ankle on the ground
    else{
      this.avatarOffsetY = rightAnkle;
      this.avatarHeight = nose-rightAnkle;
    }
  }
}
```

B.9 Version 3.0 MetricsCalculator Class

Below is the code for the **MetricsCalculator** class.

```
import { POSE_LANDMARKS_LEFT,POSE_LANDMARKS_NEUTRAL,POSE_LANDMARKS_RIGHT }
    from "@mediapipe/pose";
import * as THREE from 'three';

export default class MetricsCalculator{
    constructor(game){
        this.game = game;
        //Used to smooth this.realTimeMetrics.avgDistance
        this.lerpValue = 0.6;

        //Metrics calculated whilst the session is running
        this.realTimeMetrics = {
            "distance_m": 0.0,
            "avgDistance": 0.0,
            "time_s": 0.0,
            "speed_ms": 0.0,
            "cadence_ss": 0.0,
            "steps": 0.0
        }

        //Metrics calculated post session
        this.postSessionGraphs = {
            "filteredAnkle_left": [],
            "filteredAnkle_right": [],
            "ankleMaxima_left": [],
            "ankleMaxima_right": [],
            "ankleMinima_left": [],
            "ankleMinima_right": [],
            "filteredBalance": [],
            "balanceMinima": [],
            "balanceMaxima": [],
            "balanceMinima_values": [],
            "balanceMaxima_values": [],
            "strideLengths_left": [],
            "strideLengths_right": [],
            "swings_left": [],
            "swings_right": [],
            "stances_left": [],
            "stances_right": [],
            "doubleSupports_left": [],
            "doubleSupports_right": [],
            "kneeFlexion_left": [],
            "kneeFlexion_right": []
        }

        //Average metrics calculated post session
        this.postSessionMetrics = {
            "swing_left_s": 0.0,
            "swing_left_cent": 0.0,
            "swing_right_s": 0.0,
            "swing_right_cent": 0.0,
            "stance_left_s": 0.0,
```

```
            "stance_left_cent": 0.0,
            "stance_right_s": 0.0,
            "stance_right_cent": 0.0,
            "double_left_s": 0.0,
            "double_left_cent": 0.0,
            "double_right_s": 0.0,
            "double_right_cent": 0.0,
            "single_left_s": 0.0,
            "single_left_cent": 0.0,
            "single_right_s": 0.0,
            "single_right_cent": 0.0,
            "strideLength_left": 0.0,
            "strideLength_right": 0.0,
            "strideLength_left_stdDev": 0.0,
            "strideLength_right_stdDev": 0.0,
            "hipElevation_left": 0.0,
            "hipElevation_right": 0.0,
            "hipElevation_left_stdDev": 0.0,
            "hipElevation_right_stdDev": 0.0,
            "kneeFlexion_left": 0.0,
            "kneeFlexion_right": 0.0,
            "kneeFlexion_left_stdDev": 0.0,
            "kneeFlexion_right_stdDev": 0.0
        }

        //Variables used by the metrics algorithms
        this.trackerVariables = {
            "heelStrike": "none",
            "heelRaise": "none",
            "firstStep": true,
            "firstIteration": true,
            "time_ms": 0,
            "timeDif_ms": 0,
            "prevTime_ms": 0,
            "startTime_ms": 0,
            "distanceChange_m": 0.0,
            "avatarDistanceBuffer": [0.0, 0.0, 0.0, 0.0],
            "lastEvent": "none",

            //Important joint points for metric calculation
            "jointPoints_m": {
                "ankle_left": {
                    "x":[],
                    "y":[],
                    "z":[]
                },
                "ankle_right": {
                    "x":[],
                    "y":[],
                    "z":[]
                },
                "hip_left": {
                    "x":[],
                    "y":[],
```

```
                    "z":[]
                },
                "hip_right": {
                    "x":[],
                    "y":[],
                    "z":[]
                },
                "knee_left": {
                    "x":[],
                    "y":[],
                    "z":[]
                },
                "knee_right": {
                    "x":[],
                    "y":[],
                    "z":[]
                },
            },

             //Important joint points for metric calculation
             "jointPoints": {
                "ankle_left": 0.0,
                "ankle_right": 0.0,
                "hip_left": 0.0,
                "hip_right": 0.0,
                "dif_left": 0.0,
                "dif_right": 0.0
            },

            //Above, but for previous frame
            "jointPointsPrev": {
                "ankle_left": 0.0,
                "ankle_right": 0.0,
                "hip_left": 0.0,
                "hip_right": 0.0,
                "dif_left": 0.0,
                "dif_right": 0.0
            }
        }
        //List of the time values at each frame
        this.times_s = [];
        this.recording = false;
    }

    /** Updates all of the metrics that are calculated in real-time */
    updateMetrics(body){
        if(this.trackerVariables.time_ms > Math.round(this.game.recordTime*1000)){
            this.game.stopRecording();
        }
        else {
            //Get coordinates (metres and normalized) of key joint points used for gait calculations
            this.updateJointPoints_m(body);
```

```
            this.updateJointPoints_norm(body);

            //If previous joint points have a value
            if(!this.trackerVariables.firstIteration){
                this.updateTime();

                //Determine if any new gait events have occured
                const event = this.detectEvents();

                //Handle the different types of event
                switch(event){
                    case "heelRaise_left":
                        this.handleHeelRaise('left');
                        break;
                    case "heelRaise_right":
                        this.handleHeelRaise('right');
                        break;
                    case "heelStrike_left":
                        this.handleHeelStrike('left');
                        break;
                    case "heelStrike_right":
                        this.handleHeelStrike('right');
                        break;
                    default:
                        break;
                }

                //Only update metrics after first step
                if(!this.trackerVariables.firstStep){
                    this.updateDistance();
                    this.updateSpeed();
                    this.updateCadence();
                }
            }
            else{
                this.init();
            }
        }
    }

    /** Prints the metrics into the console */
    output(){
        this.calculatePostSessionValues();
        console.log(this);
    }

    /** Calculates the average metrics and metrics graphs using the hip and ankle location graphs*/
    calculatePostSessionValues(){
        this.filterData_ankles();
        this.calculateMinimaMaxima_ankles();
        this.getFilteredData_hips();
        this.calculateMinimaMaxima_hips();
        this.calculateMetricsGraphs();
```

```
        this.calculateKneeAngles();
        this.calculateAverageMetrics();
    }

    calculateKneeAngles(){
        const leftFlexion = this.calculateKneeFlexion(this.trackerVariables.
            jointPoints_m.hip_left, this.trackerVariables.jointPoints_m.
            knee_left, this.trackerVariables.jointPoints_m.ankle_left);
        const rightFlexion = this.calculateKneeFlexion(this.trackerVariables.
            jointPoints_m.hip_right, this.trackerVariables.jointPoints_m.
            knee_right, this.trackerVariables.jointPoints_m.ankle_right);
        this.postSessionGraphs.kneeFlexion_left = this.movingAvgFilter(
            leftFlexion,20);
        this.postSessionGraphs.kneeFlexion_right = this.movingAvgFilter(
            rightFlexion,20);
    }

    /** Calculates stride lengths, swings, and stances  */
    calculateStrides(maxima,minima,distanceArray,distanceArray_opposite,
        timeArray,swings,stances,strideLengths){
        //Variables to track the current position in the maxima and minima
            arrays
        var max = 0;
        var min = 0;

        //While the size of the maxima and minima arrays have not been
            exceeded
        while(max < maxima.length && min < minima.length){
            //Get current max and min and next max and min
            const heelStrike = maxima[max];
            const heelRaise = minima[min];
            const heelStrike_next = maxima[max + 1];
            const heelRaise_next = minima[min + 1];

            //Stance event detection
            if(heelRaise > heelStrike){
                if(heelStrike_next > heelRaise){
                    console.log("a");
                    const stance = timeArray[heelRaise] - timeArray[heelStrike
                        ];
                    stances.push(stance)
                }
                max++;
            }
            //Swing event detection
            else{
                if(heelRaise_next > heelStrike){
                    console.log("b");
                    const strideLength = distanceArray[heelStrike] -
                        distanceArray_opposite[heelStrike];
                    const swing = timeArray[heelStrike] - timeArray[heelRaise
                        ];
                    strideLengths.push(strideLength);
                    swings.push(swing);
```

```
                }
                min++;
            }
        }
    }

    /** Calculates double support values */
    calculateDoubleSupports(maxima,minima,timeArray,doubleSupports){
        //Variables to track the current position in the maxima and minima
            arrays
        var max = 0;
        var min = 0;

        //While the size of the maxima and minima arrays have not been
            exceeded
        while(max < maxima.length && min < minima.length){
            //Get max, min and next max
            const heelStrike = maxima[max];
            const heelRaise = minima[min];
            const heelStrike_next = maxima[max + 1];

            //Double support event
            if(heelRaise > heelStrike){
                if(heelStrike_next > heelRaise){
                    const doubleSupport = timeArray[heelRaise] - timeArray[
                        heelStrike];
                    doubleSupports.push(doubleSupport)
                }
                max++;
            }
            else{
                min++;
            }
        }
    }

    /** Calculates filtered balance angle from hips data and get maxima and
        minima */
    calculateMinimaMaxima_hips(){
        this.postSessionGraphs.balanceMaxima = this.getMaximaWindowed(this.
            postSessionGraphs.filteredBalance,60,0.5);
        this.postSessionGraphs.balanceMinima = this.getMinimaWindowed(this.
            postSessionGraphs.filteredBalance,60,0.5);
        this.getBalancePeakValues();
    }

    getFilteredData_hips(){
        const hipAngles = this.calculateBalance(this.trackerVariables.
            jointPoints_m.hip_left.y,this.trackerVariables.jointPoints_m.
            hip_right.y,this.trackerVariables.jointPoints_m.hip_left.x,this.
            trackerVariables.jointPoints_m.hip_right.x);
        this.postSessionGraphs.filteredBalance = this.movingAvgFilter(
            hipAngles,20);
    }
```

```
    /** Filter left and right ankle data */
    filterData_ankles(){
        this.postSessionGraphs.filteredAnkle_left = this.movingAvgFilter(this.trackerVariables.jointPoints_m.ankle_left.z,20);
        this.postSessionGraphs.filteredAnkle_right = this.movingAvgFilter(this.trackerVariables.jointPoints_m.ankle_right.z,20);
    }

    /** Calculate the minimum and maximum of the ankle locations (heel strike and raise) */
    calculateMinimaMaxima_ankles(){
        //Get mean of left and right ankle data
        const mean_left = this.calculateAvg(this.postSessionGraphs.filteredAnkle_left);
        const mean_right = this.calculateAvg(this.postSessionGraphs.filteredAnkle_right);

        //Get standard deviation of left and right ankle data
        const stdDev_left = this.calculateStdDev(mean_left,this.postSessionGraphs.filteredAnkle_left);
        const stdDev_right = this.calculateStdDev(mean_right,this.postSessionGraphs.filteredAnkle_right);

        //Get maxima and minima for both angles
        const ankleMaxima_left = this.getMaximaWindowed(this.postSessionGraphs.filteredAnkle_left,100,0.3);
        const ankleMaxima_right = this.getMaximaWindowed(this.postSessionGraphs.filteredAnkle_right,100,0.3);
        const ankleMinima_left = this.getMinimaWindowed(this.postSessionGraphs.filteredAnkle_left,100,0.3);
        const ankleMinima_right = this.getMinimaWindowed(this.postSessionGraphs.filteredAnkle_right,100,0.3);

        //Filter out any maxima or minima that are not close to the extremes
        this.postSessionGraphs.ankleMaxima_left = this.checkMaxima(mean_left,stdDev_left,ankleMaxima_left,this.postSessionGraphs.filteredAnkle_left);
        this.postSessionGraphs.ankleMaxima_right = this.checkMaxima(mean_right,stdDev_right,ankleMaxima_right,this.postSessionGraphs.filteredAnkle_right);
        this.postSessionGraphs.ankleMinima_left = this.checkMinima(mean_left,stdDev_left,ankleMinima_left,this.postSessionGraphs.filteredAnkle_left);
        this.postSessionGraphs.ankleMinima_right = this.checkMinima(mean_right,stdDev_right,ankleMinima_right,this.postSessionGraphs.filteredAnkle_right);
    }

    /** Calculates graphs of stride length, swing time and stance time */
    calculateMetricsGraphs(){
        this.calculateStrides(this.postSessionGraphs.ankleMaxima_left,this.postSessionGraphs.ankleMinima_left,this.postSessionGraphs.filteredAnkle_left,this.postSessionGraphs.filteredAnkle_right,this
```

```
            .times_s, this.postSessionGraphs.swings_left, this.postSessionGraphs
            .stances_left, this.postSessionGraphs.strideLengths_left);
        this.calculateStrides(this.postSessionGraphs.ankleMaxima_right, this.
            postSessionGraphs.ankleMinima_right, this.postSessionGraphs.
            filteredAnkle_right, this.postSessionGraphs.filteredAnkle_left, this
            .times_s, this.postSessionGraphs.swings_right, this.
            postSessionGraphs.stances_right, this.postSessionGraphs.
            strideLengths_right);
        this.calculateDoubleSupports(this.postSessionGraphs.ankleMaxima_left,
            this.postSessionGraphs.ankleMinima_right, this.times_s, this.
            postSessionGraphs.doubleSupports_left);
        this.calculateDoubleSupports(this.postSessionGraphs.ankleMaxima_right,
            this.postSessionGraphs.ankleMinima_left, this.times_s, this.
            postSessionGraphs.doubleSupports_right);
    }

    /** Calculate average of important metrics */
    calculateAverageMetrics(){
        //Calculate swing/stance
        this.postSessionMetrics.swing_left_s = this.postSessionMetrics.
            single_right_s = this.calculateAvg(this.postSessionGraphs.
            swings_left)/1000;
        this.postSessionMetrics.swing_right_s = this.postSessionMetrics.
            single_left_s = this.calculateAvg(this.postSessionGraphs.
            swings_right)/1000;
        this.postSessionMetrics.stance_left_s = this.calculateAvg(this.
            postSessionGraphs.stances_left)/1000;
        this.postSessionMetrics.stance_right_s = this.calculateAvg(this.
            postSessionGraphs.stances_right)/1000;
        this.postSessionMetrics.swing_left_cent = this.calculateDualPercentage
            (this.postSessionMetrics.swing_left_s, this.postSessionMetrics.
            stance_left_s);
        this.postSessionMetrics.swing_right_cent = this.
            calculateDualPercentage(this.postSessionMetrics.swing_right_s, this
            .postSessionMetrics.stance_right_s);
        this.postSessionMetrics.stance_left_cent = 100 - this.
            postSessionMetrics.swing_left_cent;
        this.postSessionMetrics.stance_right_cent = 100 - this.
            postSessionMetrics.swing_right_cent;

        //Calculate double/single
        this.postSessionMetrics.double_left_s = this.calculateAvg(this.
            postSessionGraphs.doubleSupports_left)/1000;
        this.postSessionMetrics.double_right_s = this.calculateAvg(this.
            postSessionGraphs.doubleSupports_right)/1000;
        this.postSessionMetrics.single_left_cent = this.
            calculateDualPercentage(this.postSessionMetrics.single_left_s, this
            .postSessionMetrics.double_left_s);
        this.postSessionMetrics.single_right_cent = this.
            calculateDualPercentage(this.postSessionMetrics.single_right_s,
            this.postSessionMetrics.double_right_s);
        this.postSessionMetrics.double_left_cent = 100 - this.
            postSessionMetrics.single_left_cent;
```

```
        this.postSessionMetrics.double_right_cent = 100 - this.
            postSessionMetrics.single_right_cent;

        //Calculate stride length
        this.postSessionMetrics.strideLength_left = this.calculateAvg(this.
            postSessionGraphs.strideLengths_left);
        this.postSessionMetrics.strideLength_left_stdDev = this.
            calculateStdDev(this.postSessionMetrics.strideLength_left, this.
            postSessionGraphs.strideLengths_left);
        this.postSessionMetrics.strideLength_right = this.calculateAvg(this.
            postSessionGraphs.strideLengths_right);
        this.postSessionMetrics.strideLength_right_stdDev = this.
            calculateStdDev(this.postSessionMetrics.strideLength_right, this.
            postSessionGraphs.strideLengths_right);

        //Calculate hip elevation
        this.postSessionMetrics.hipElevation_left = this.calculateAvg(this.
            postSessionGraphs.balanceMaxima_values);
        this.postSessionMetrics.hipElevation_left_stdDev = this.
            calculateStdDev(this.postSessionMetrics.hipElevation_left, this.
            postSessionGraphs.balanceMaxima_values);
        this.postSessionMetrics.hipElevation_right = this.calculateAvg(this.
            postSessionGraphs.balanceMinima_values);
        this.postSessionMetrics.hipElevation_right_stdDev = this.
            calculateStdDev(this.postSessionMetrics.hipElevation_right, this.
            postSessionGraphs.balanceMinima_values);

        this.postSessionMetrics.kneeFlexion_left = this.calculateAvg(this.
            postSessionGraphs.kneeFlexion_left);
        this.postSessionMetrics.kneeFlexion_left_stdDev = this.calculateStdDev
            (this.postSessionMetrics.kneeFlexion_left, this.postSessionGraphs.
            kneeFlexion_left);
        this.postSessionMetrics.kneeFlexion_right = this.calculateAvg(this.
            postSessionGraphs.kneeFlexion_right);
        this.postSessionMetrics.kneeFlexion_right_stdDev = this.
            calculateStdDev(this.postSessionMetrics.kneeFlexion_right, this.
            postSessionGraphs.kneeFlexion_right);
    }

    /** Filters an array using a smoothing filter */
    movingAvgFilter(array, bufferSize){
        var i = 0;
        var filteredArray = [];

        //Filter a given window of data each iteration
        array.forEach((value)=>{
            var filteredValue = 0.0;
            if(i<array.length-bufferSize){
                //Find the sum of values to be averaged
                for(let j=0; j<bufferSize; j++){
                    filteredValue += array[i+j];
                }
                //Calculate average from sum
                filteredValue /= bufferSize;
```

```
                filteredArray.push(filteredValue);
            }
            i++;
        })
        return filteredArray;
    }

    /** Peak detection from array data */
    getMaximaWindowed(array,windowSize,overlap){
        var peaks = [];

        //Window of size "windowSize" that overlaps a fractional amount defined by overlap
        for(let i = 0; i < array.length; i += (windowSize*overlap)){
            //Detect highest peak in the window
            const window = array.slice(i,i+windowSize);
            const peak = this.getMaximum(window);

            //Check for identical peaks on window boundaries
            if(peak!=null && (((i + peak) - peaks[peaks.length-1] >= (windowSize*overlap)) || peaks.length == 0)){
                peaks.push(i + peak)
            }
        }
        return peaks;
    }

    /** Gets the minima of an array */
    getMinimaWindowed(array,windowSize,overlap){
        const negatedArray = [];
        //Minima become maxima when array is negated
        array.forEach((element)=>{
            negatedArray.push(-element);
        })
        const minima = this.getMaximaWindowed(negatedArray,windowSize,overlap);
        return minima;
    }

    /** Checks if minima are smaller than mean - (0.5 * standard deviation) */
    checkMinima(mean,stdDev,indexes,array){
        var newArray = [];
        const limit = mean-(stdDev*0.5);

        indexes.forEach((element)=>{
            if(array[element] < limit){
                newArray.push(element);
            }
        })
        return newArray;
    }

    /** Checks if maxima are greater than mean + (0.5 * standard deviation) */
    checkMaxima(mean,stdDev,indexes,array){
```

```
        var newArray = [];
        const limit = mean+(stdDev*0.5);

        //If the maximum meets the criteria, add it to the final array of maxima
        indexes.forEach((element)=>{
            if(array[element] > limit){
                newArray.push(element);
            }
        })
        return newArray;
    }

    /** Gets the maximum from an array */
    getMaximum(array){
        var peak = -Infinity;
        let peakIndex;
        var peakDetected = false;

        for(let i = 1; i < array.length-1; i++){
            //Peak is defined as an upside down V shape (lower number --> highest number --> lower number)
            const a = (array[i-1] - array[i]);
            const b = (array[i+1] - array[i]);
            if(a<0 && b<0){
                if(!peakDetected){
                    peakDetected = true;
                }
                //Search for the largest peak in the window
                if(array[i] > peak){
                    peakIndex = i;
                    peak = array[i];
                }
            }
        }
        if(peakDetected){
            return peakIndex;
        }
        else{
            return null;
        }
    }

    findKneeAngle(hip,knee,ankle,i){
        //Initialize co-ordinates as 2D vectors
        const hip_loc = new THREE.Vector2(hip.z[i],hip.y[i]);
        const knee_loc = new THREE.Vector2(knee.z[i],knee.y[i]);
        const ankle_loc = new THREE.Vector2(ankle.z[i],ankle.y[i]);

        //Get direction
        let shin_dir = new THREE.Vector2();
        shin_dir.subVectors(ankle_loc,knee_loc);
        let thigh_dir = new THREE.Vector2();
        thigh_dir.subVectors(hip_loc,knee_loc);
```

```
        const shin_angle = shin_dir.angle();
        const thigh_angle = thigh_dir.angle();

        console.log((shin_angle*180)/Math.PI);
        console.log((thigh_angle*180)/Math.PI);

        return (Math.abs(shin_angle-thigh_angle)*180)/Math.PI;
    }

    calculateKneeFlexion(hip,knee,ankle){
        var angles = [];

        //Calculate knee flexion for every frame and return an array of the
            angle data
        for(let i=0; i<hip.y.length; i++){
            const angle = this.findKneeAngle(hip,knee,ankle,i);
            angles.push(angle);
        }
        return angles;
    }

    /** Finds the angle between left and right hips */
    findHipAngle(hip_left_y,hip_right_y,hip_left_x,hip_right_x){
        //Absolute as we are only interested in overall balance
        const x = Math.abs(hip_left_x - hip_right_x);
        const y = hip_left_y - hip_right_y;

        //Calculate and return angle in degrees
        const angle = Math.atan2(y,x);
        return ((angle*180)/Math.PI);
    }

    /** Calculates a graph of balance based on hip angle */
    calculateBalance(hip_left_y,hip_right_y,hip_left_x,hip_right_x){
        var angles = [];

        //Calculate hip angles for every hip position and return an array of
            the angle data
        for(let i=0; i<hip_left_y.length; i++){
            const angle = this.findHipAngle(hip_left_y[i],hip_right_y[i],
                hip_left_x[i],hip_right_x[i]);
            angles.push(angle);
        }
        return angles;
    }

    /** Calculates actual values of balance minima and maxima */
    getBalancePeakValues(){
        this.postSessionGraphs.balanceMaxima.forEach((value)=>{
            this.postSessionGraphs.balanceMaxima_values.push(this.
                postSessionGraphs.filteredBalance[value]);
        })
        this.postSessionGraphs.balanceMinima.forEach((value)=>{
```

```
            this.postSessionGraphs.balanceMinima_values.push(this.postSessionGraphs.filteredBalance[value]);
        })
    }

    /** Calculates the average of values in an array */
    calculateAvg(array){
        var total = 0.0;

        //Get the total of the array and divide by the number of elements
        array.forEach((element)=>{
            total += element;
        })
        return total/array.length
    }

    /** Calculates the standard deviation of items in an array given the mean */
    calculateStdDev(mean,array){
        var total = 0.0;

        //Get sum of variances
        array.forEach((element)=>{
            total += Math.pow(element-mean,2);
        })
        return Math.sqrt(total/array.length);
    }

    /** Calculate a as a percentage of a + b */
    calculateDualPercentage(a,b){
        const cent = Math.round((a/(a+b))*1000)/10
        return cent;
    }

    /** Sets timer to 0 and signals that one iteration of pose estimation has passed */
    init(){
        this.trackerVariables.startTime_ms = Date.now();
        this.trackerVariables.firstIteration = false;
    }

    /** Update the current time value and the previous time value */
    updateTime(){
        //Set previous time to lat recorded time
        this.trackerVariables.prevTime_ms = this.trackerVariables.time_ms;

        //Set new time and calculate difference between the previous
        this.trackerVariables.time_ms = Date.now() - this.trackerVariables.startTime_ms;
        this.trackerVariables.timeDif_ms = this.trackerVariables.time_ms - this.trackerVariables.prevTime_ms;
        this.realTimeMetrics.time_s = this.trackerVariables.time_ms/1000
        this.times_s.push(this.trackerVariables.time_ms);
    }
```

```
    /** Update a record of the important joint points (metres) for gait (and
        their values for the previous pose) */
    updateJointPoints_m(body){
        //Set current positions
        this.trackerVariables.jointPoints_m.ankle_left.z.push(-body.
            poseWorldLandmarks[POSE_LANDMARKS_LEFT.LEFT_HEEL].z);
        this.trackerVariables.jointPoints_m.ankle_left.x.push(body.
            poseWorldLandmarks[POSE_LANDMARKS_LEFT.LEFT_HEEL].x);
        this.trackerVariables.jointPoints_m.ankle_left.y.push(body.
            poseWorldLandmarks[POSE_LANDMARKS_LEFT.LEFT_HEEL].y);
        this.trackerVariables.jointPoints_m.ankle_right.z.push(-body.
            poseWorldLandmarks[POSE_LANDMARKS_RIGHT.RIGHT_HEEL].z);
        this.trackerVariables.jointPoints_m.ankle_right.x.push(body.
            poseWorldLandmarks[POSE_LANDMARKS_RIGHT.RIGHT_HEEL].x);
        this.trackerVariables.jointPoints_m.ankle_right.y.push(body.
            poseWorldLandmarks[POSE_LANDMARKS_RIGHT.RIGHT_HEEL].y);
        this.trackerVariables.jointPoints_m.hip_left.z.push(-body.
            poseWorldLandmarks[POSE_LANDMARKS_LEFT.LEFT_HIP].z);
        this.trackerVariables.jointPoints_m.hip_left.x.push(body.
            poseWorldLandmarks[POSE_LANDMARKS_LEFT.LEFT_HIP].x);
        this.trackerVariables.jointPoints_m.hip_left.y.push(body.
            poseWorldLandmarks[POSE_LANDMARKS_LEFT.LEFT_HIP].y);
        this.trackerVariables.jointPoints_m.hip_right.z.push(-body.
            poseWorldLandmarks[POSE_LANDMARKS_RIGHT.RIGHT_HIP].z);
        this.trackerVariables.jointPoints_m.hip_right.x.push(body.
            poseWorldLandmarks[POSE_LANDMARKS_RIGHT.RIGHT_HIP].x);
        this.trackerVariables.jointPoints_m.hip_right.y.push(body.
            poseWorldLandmarks[POSE_LANDMARKS_RIGHT.RIGHT_HIP].y);
        this.trackerVariables.jointPoints_m.knee_left.z.push(-body.
            poseWorldLandmarks[POSE_LANDMARKS_LEFT.LEFT_KNEE].z);
        this.trackerVariables.jointPoints_m.knee_left.x.push(body.
            poseWorldLandmarks[POSE_LANDMARKS_LEFT.LEFT_KNEE].x);
        this.trackerVariables.jointPoints_m.knee_left.y.push(body.
            poseWorldLandmarks[POSE_LANDMARKS_LEFT.LEFT_KNEE].y);
        this.trackerVariables.jointPoints_m.knee_right.z.push(-body.
            poseWorldLandmarks[POSE_LANDMARKS_RIGHT.RIGHT_KNEE].z);
        this.trackerVariables.jointPoints_m.knee_right.x.push(body.
            poseWorldLandmarks[POSE_LANDMARKS_RIGHT.RIGHT_KNEE].x);
        this.trackerVariables.jointPoints_m.knee_right.y.push(body.
            poseWorldLandmarks[POSE_LANDMARKS_RIGHT.RIGHT_KNEE].y);
    }

    /** Update a record of the important joint points (normalized units) for
        gait (and their values for the previous pose) */
     updateJointPoints_norm(body){
        //Set previous positions to last current positions
        this.trackerVariables.jointPointsPrev.ankle_left = this.
            trackerVariables.jointPoints.ankle_left;
        this.trackerVariables.jointPointsPrev.ankle_right = this.
            trackerVariables.jointPoints.ankle_right;
        this.trackerVariables.jointPointsPrev.hip_left = this.trackerVariables
            .jointPoints.hip_left;
```

```
        this.trackerVariables.jointPointsPrev.hip_right = this.trackerVariables.jointPoints.hip_right;

        //Set current positions
        this.trackerVariables.jointPoints.ankle_left = body.poseLandmarks[POSE_LANDMARKS_LEFT.LEFT_ANKLE].z;
        this.trackerVariables.jointPoints.ankle_right = body.poseLandmarks[POSE_LANDMARKS_RIGHT.RIGHT_ANKLE].z;
        this.trackerVariables.jointPoints.hip_left = body.poseLandmarks[POSE_LANDMARKS_LEFT.LEFT_HIP].z;
        this.trackerVariables.jointPoints.hip_right = body.poseLandmarks[POSE_LANDMARKS_RIGHT.RIGHT_HIP].z;

        //Get difference between leg and hip for previous and current positions
        this.trackerVariables.jointPoints.dif_left = this.trackerVariables.jointPoints.hip_left - this.trackerVariables.jointPoints.ankle_left;
        this.trackerVariables.jointPoints.dif_right = this.trackerVariables.jointPoints.hip_right - this.trackerVariables.jointPoints.ankle_right;
        this.trackerVariables.jointPointsPrev.dif_left = this.trackerVariables.jointPointsPrev.hip_left - this.trackerVariables.jointPointsPrev.ankle_left;
        this.trackerVariables.jointPointsPrev.dif_right = this.trackerVariables.jointPointsPrev.hip_right - this.trackerVariables.jointPointsPrev.ankle_right;
    }

    /** Detects if there has been a heel strike or raise on either foot */
    detectEvents(){
        this.trackerVariables.heelChange = false;

        const hipDif_l = this.trackerVariables
        //If the last event was a raise the next must be a strike. A strike is always the first event detected
        if(this.trackerVariables.lastEvent == "none" || this.trackerVariables.lastEvent == "raise"){
            //Detect left heel strike
            if (this.trackerVariables.jointPoints.dif_left > this.trackerVariables.jointPointsPrev.dif_left && (this.trackerVariables.heelStrike == "none" || this.trackerVariables.heelStrike == "right") && this.trackerVariables.jointPoints.ankle_left > this.trackerVariables.jointPoints.ankle_right){
                this.trackerVariables.lastEvent = "strike";
                return "heelStrike_left";
            }
            //Detect right heel strike
            else if (this.trackerVariables.jointPoints.dif_right > this.trackerVariables.jointPointsPrev.dif_right && (this.trackerVariables.heelStrike == "none" || this.trackerVariables.heelStrike == "left") && this.trackerVariables.jointPoints.ankle_right > this.trackerVariables.jointPoints.ankle_left){
                this.trackerVariables.lastEvent = "strike";
```

```
                return "heelStrike_right";
            }
        }
        //If the last event was a strike, the next event must be a raise
        else{
            //Detect right heel raise
            if(this.trackerVariables.jointPoints.dif_right < this.trackerVariables.jointPointsPrev.dif_right && (this.trackerVariables.heelRaise == "none" || this.trackerVariables.heelRaise =="left") && this.trackerVariables.jointPoints.ankle_right < this.trackerVariables.jointPoints.ankle_left){
                this.trackerVariables.lastEvent = "raise";
                return "heelRaise_right";
            }
            //Detect left heel raise
            else if(this.trackerVariables.jointPoints.dif_left < this.trackerVariables.jointPointsPrev.dif_left && (this.trackerVariables.heelRaise == "none" || this.trackerVariables.heelRaise =="right") && this.trackerVariables.jointPoints.ankle_left < this.trackerVariables.jointPoints.ankle_right){
                this.trackerVariables.lastEvent = "raise";
                return "heelRaise_left";
            }
        }
        return "";
    }

    /** Calculates distance (for metrics) and average distance (for moving the avatar forward) */
    updateDistance(){
        if(this.trackerVariables.heelStrike == "left" || this.trackerVariables.heelStrike == "right"){
            var distance = 0.0;
            var distance_m = 0.0;

            let difference_m;
            let difference;

            //Get distance change
            if (this.trackerVariables.heelStrike == "left"){
                const length = this.trackerVariables.jointPoints_m.ankle_left.z.length;
                difference_m = this.trackerVariables.jointPoints_m.ankle_left.z[length-2] - this.trackerVariables.jointPoints_m.ankle_left.z[length-1];
                difference = this.trackerVariables.jointPointsPrev.ankle_left - this.trackerVariables.jointPoints.ankle_left;
            }
            else if(this.trackerVariables.heelStrike == "right"){
                const length = this.trackerVariables.jointPoints_m.ankle_right.z.length;
                difference_m = this.trackerVariables.jointPoints_m.ankle_right.z[length-2] - this.trackerVariables.jointPoints_m.ankle_right.z[length-1];
```

```
                difference = this.trackerVariables.jointPointsPrev.ankle_right - this.trackerVariables.jointPoints.ankle_right;
            }

            //Add the distance change to the buffer if it is positive
            if (difference > 0){
                distance = difference;
                distance_m = difference_m;
                this.trackerVariables.avatarDistanceBuffer.unshift((distance + this.trackerVariables.avatarDistanceBuffer[0]) * this.lerpValue);
            }
            //If the distance change is negative, just add 0
            else{
                this.trackerVariables.avatarDistanceBuffer.unshift((0.0));
            }

            //Update non-filtered distance
            this.realTimeMetrics.distance_m += distance_m;
            this.trackerVariables.distanceChange_m = distance_m;

            //Remove last element from the buffer
            this.trackerVariables.avatarDistanceBuffer.pop();

            //Read average distance from the buffer
            let avg = 0.0;
            this.trackerVariables.avatarDistanceBuffer.forEach((element)=>{
                avg += element;
            });
            avg /= this.trackerVariables.avatarDistanceBuffer.length;

            //Update distance away from camera
            this.realTimeMetrics.avgDistance = (this.realTimeMetrics.avgDistance + avg) % 14;
        }
    }

    /** Update speed in metres per second and add it to the speed graph */
    updateSpeed(){
        const speed = this.trackerVariables.distanceChange_m / (this.trackerVariables.timeDif_ms / 1000);
        this.realTimeMetrics.speed_ms = speed;
    }

    /** Update cadence in steps per second and add it to the cadence graph */
    updateCadence(){
        const cadence = this.realTimeMetrics.steps/(this.trackerVariables.time_ms / 1000);
        this.realTimeMetrics.cadence_ss = cadence;
    }

    /** Update the arrays of heel raise info and record the start of a step */
    handleHeelRaise(heel){
        //Record the heel raise
```

```
            this.trackerVariables.heelRaise = heel;
        }

        /** Update the arrays of heel strike info and record the step length */
        handleHeelStrike(heel){
            if(this.trackerVariables.firstStep){
                this.trackerVariables.firstStep = false;
            }
            //Inform the algorithm of a heel change
            this.trackerVariables.heelChange = true;
            this.realTimeMetrics.steps++;
            this.trackerVariables.heelStrike = heel;
        }
}
```

C Web App Testing

C.1 Unit Tests for Training Page

Below is the code for the cypress unit tests.

```
/// <reference types="Cypress" />

import id from "../../components/pages/training/id";

const screenSizes = ['macbook-15','iphone-x'];

const colourList = ["Black","White","Red","Orange","Yellow","Green","Blue","Purple","Select an option"];
const groundValuesList = ["Terracotta","Moss","Rock","Select an option"];
const skyValuesList = ["Cloudy","Overcast","Night","Select an option"];
const colourTaskSpeed = ["1","2","3","Select an option"];
const testTime = ["0.5","1","1.5","2","Select an option"];
const coloursTask = ["Yes","No","Select an option"];
const guidance = ["Yes","No","Select an option"];

describe('Training page',()=>{
    screenSizes.forEach((size)=>{
        it('should open',()=>{
            cy.viewport(size);
            cy.visit("http://localhost:3000/training");
        })
    });
})

describe('Avatar settings',()=>{
    screenSizes.forEach((size)=>{
        beforeEach(()=>{
            cy.viewport(size);
        });
        it(`allows user to select different bone colours (screen size: ${size})`,()=>{
            checkDropdownText("#"+id.Training.BoneColour, colourList);
        });
        it(`allows the user to select different joint colours (screen size: ${size})`,()=>{
            checkDropdownText("#"+id.Training.JointColour, colourList);
        });
        it(`allows the user to select different head colours (screen size: ${size})`,()=>{
            checkDropdownText("#"+id.Training.HeadColour, colourList);
        });
```

```
    });
});

describe('World settings',()=>{
    screenSizes.forEach((size)=>{
        beforeEach(()=>{
            cy.viewport(size);
        });
        it(`allows the user to select different ground textures (screen size: ${size})`,()=>{
            checkDropdownText("#"+id.Training.GroundTexture, groundValuesList);
        });
        it(`allows the user to select different sky textures (screen size: ${size})`,()=>{
            checkDropdownText("#"+id.Training.SkyTexture, skyValuesList);
        });
    });
});

describe('Interactions settings',()=>{
    screenSizes.forEach((size)=>{
        beforeEach(()=>{
            cy.viewport(size);
        });
        it(`allows the user to enable and disable the colours task (screen size: ${size})`,()=>{
            checkDropdownText("#"+id.Training.CogNonCog, coloursTask);
        });
        it(`allows the user to set the speed for the colours task (screen size: ${size})`,()=>{
            checkDropdownText("#"+id.Training.DistractorSpeed, colourTaskSpeed);
        });
        it(`allows the user to set the time for the test (screen size: ${size})`,()=>{
            checkDropdownText("#"+id.Training.TestTime, testTime);
        });
    });
});

describe('Interactions settings',()=>{
    screenSizes.forEach((size)=>{
        beforeEach(()=>{
            cy.viewport(size);
        });
        it(`allows the user to select different ground textures (screen size: ${size})`,()=>{
            checkDropdownText("#"+id.Training.GroundTexture, groundValuesList);
        });
        it(`allows the user to select different sky textures (screen size: ${size})`,()=>{
            checkDropdownText("#"+id.Training.SkyTexture, skyValuesList);
```

```
        });
    });
});

/* Compares the dropdown text againt those in the provided list */
function checkDropdownText(id,textList){
    //Cycle through each dropdown option and verify the text
    var i = 0;
    cy.get(id+'>option').each((el)=>{
        expect(el.text()).to.eq(textList[i]);
        i++;
    });
}

const buttonEmph = 'bg-gradient-to-b from-yellow-400 to-yellow-500';
const buttonNonEmph = 'bg-gradient-to-b from-yellow-200 to-yellow-300';

describe('mobile switch displays button',()=>{
    context('iphone-x',()=>{
        beforeEach(()=>{
            cy.viewport('iphone-x');
        });
        it(`should switch to the three display (screen size: iphone-x)`,()=>{
            cy.get("#"+id.Training.VideoSwitchDisplay).click();
            cy.get("#"+id.Training.VideoFeed).should('have.class','invisible');
            cy.get("#"+id.Training.GameWindow).should('not.have.class','invisible');
        });
        it(`should switch back to the video display (screen size: iphone-x)`,()=>{
            cy.get("#"+id.Training.ThreeSwitchDisplay).click();
            cy.get("#"+id.Training.VideoFeed).should('not.have.class','invisible');
            cy.get("#"+id.Training.GameWindow).should('have.class','invisible');
        });
    });
});

describe('mobile menu open & close button (video window)',()=>{
    context('iphone-x',()=>{
        it(`should open the settings menu (screen size: iphone-x)`,()=>{
            cy.viewport('iphone-x');
            cy.get("#"+id.Training.VideoGCSToggle).click();
            cy.get("#"+id.Training.GCS).should('not.have.class','invisible');
            cy.get("#"+id.Training.CloseGCS).click();
            cy.get("#"+id.Training.GCS).should('have.class','invisible');
        });
    });
});

describe('mobile menu open & close button (three window)',()=>{
    context('iphone-x',()=>{
```

```
        it(`should open the settings menu (screen size: iphone-x)`,()=>{
            cy.viewport('iphone-x');
            cy.get("#"+id.Training.VideoSwitchDisplay).click();
            cy.get("#"+id.Training.ThreeGCSToggle).click();
            cy.get("#"+id.Training.GCS).should('not.have.class','invisible');
            cy.get("#"+id.Training.CloseGCS).click();
            cy.get("#"+id.Training.GCS).should('have.class','invisible');
            cy.get("#"+id.Training.ThreeSwitchDisplay).click();
        });
    });
});


describe('Settings navbar',()=>{
    screenSizes.forEach((size)=>{
        it(`should only display the avatar settings when the avatar button is clicked (screen size: ${size})`,()=>{
            //Set viewport size and visit training page
            cy.viewport(size);

            if(size == 'iphone-x'){
                cy.get("#"+id.Training.VideoGCSToggle).click();
            }
            cy.get("#"+id.Training.WorldButton).click();
            cy.get("#"+id.Training.AvatarButton).click();
            cy.get("#"+id.Training.AvatarSettings).should('not.have.class','hidden');
            cy.get("#"+id.Training.WorldSettings).should('have.class','hidden');
            cy.get("#"+id.Training.InteractionsSettings).should('have.class','hidden');
        });
        it(`should emphasize the avatar button when the avatar button is clicked (screen size: ${size})`,()=>{
            cy.viewport(size);
            cy.get("#"+id.Training.AvatarButton).should('have.class',buttonEmph);
        });
        it(`should de-emphasize the world button when the avatar button is clicked (screen size: ${size})`,()=>{
            cy.viewport(size);
            cy.get("#"+id.Training.WorldButton).should('have.class',buttonNonEmph);
        });
        it(`should de-emphasize the interactions button when the avatar button is clicked (screen size: ${size})`,()=>{
            cy.viewport(size);
            cy.get("#"+id.Training.InteractionsButton).should('have.class',buttonNonEmph);
        });
        it(`should only display the interactions settings when the avatar button is clicked (screen size: ${size})`,()=>{
            //Set viewport size and visit training page
            cy.viewport(size);
```

```
            cy.get("#"+id.Training.InteractionsButton).click();
            cy.get("#"+id.Training.AvatarSettings).should('have.class','hidden');
            cy.get("#"+id.Training.WorldSettings).should('have.class','hidden');
            cy.get("#"+id.Training.InteractionsSettings).should('not.have.class','hidden');
        });
        it(`should de-emphasize the avatar button when the interactions button is clicked (screen size: ${size})`,()=>{
            cy.viewport(size);
            cy.get("#"+id.Training.AvatarButton).should('have.class',buttonNonEmph);
        });
        it(`should de-emphasize the world button when the interactions button is clicked (screen size: ${size})`,()=>{
            cy.viewport(size);
            cy.get("#"+id.Training.WorldButton).should('have.class',buttonNonEmph);
        });
        it(`should emphasize the interactions button when the interactions button is clicked (screen size: ${size})`,()=>{
            cy.viewport(size);
            cy.get("#"+id.Training.InteractionsButton).should('have.class',buttonEmph);
        });
        it(`should only display the world settings when the world button is clicked (screen size: ${size})`,()=>{
            cy.viewport(size);
            cy.get("#"+id.Training.WorldButton).click();
            cy.get("#"+id.Training.AvatarSettings).should('have.class','hidden');
            cy.get("#"+id.Training.InteractionsSettings).should('have.class','hidden');
            cy.get("#"+id.Training.WorldSettings).should('not.have.class','hidden');
        });
        it(`should emphasize the world button when the world button is clicked (screen size: ${size})`,()=>{
            cy.viewport(size);
            cy.get("#"+id.Training.WorldButton).should('have.class',buttonEmph);
        });
        it(`should de-emphasize the avatar button when the world button is clicked (screen size: ${size})`,()=>{
            cy.viewport(size);
            cy.get("#"+id.Training.AvatarButton).should('have.class',buttonNonEmph);
            if(size == 'iphone-x'){
                cy.get("#"+id.Training.CloseGCS).click();
            }
        });
    });
});
```

```
describe('Metrics panel',()=>{
    screenSizes.forEach((size)=>{
        it(`should be invisible when not fullscreen (screen size: ${size})`,()
            =>{
            cy.viewport(size);
            cy.get("#"+id.Training.MetricsDisplay).should('have.class','
                invisible');
        });
    });
});
```

D Study Documents

D.1 Study Consent Form

The study consent form is included on the two pages following this one.

Evaluating the Accuracy of Web-Based Gait Analysis

1 Information About the Project

This project is focused on developing a low-cost web-based VR application that analyses and improves movement. The application uses a single RGB camera to capture video footage of the subject walking on a treadmill.

The application uses the video footage to detect the positions of the subject's joints in 3D space. Using this position data, it builds a stick figure on a screen in front of the patient that mirrors their movements in real-time. It also performs a more detailed analysis of their movement after they finish walking. This detailed analysis could be used by clinicians in the near future to diagnose movement problems and disorders.

This study will assess the viability of the VR application for gait analysis against a gold standard motion capture system (Optitrack).

2 Before the Test

Before the test begins, the test operator will assign you a patient code (this will also be sent to you by email). You will then enter some personal details (patient code, date of birth, height, weight and gender). You can opt out of any of these details (apart from the patient code).

Please try to wear tighter fitting trousers such as joggers or jeggings, as this makes placing the Optitrack markers much easier.

3 Testing Process

3.1 Marker Placement

Before starting, the session operator will ask if you are comfortable with them placing adhesive markers on top of your clothing on your hips, knees and ankles. Alternatively, the session operator can tell you where to place the markers yourself.

3.2 Rounds

The test consists of "rounds", each 1 minute in length. There are nine rounds in total performed at different speeds. Before the testing rounds, there will also be one warm-up round. This is so you can get used to the setup.

You can ask to take a break at any point during the test. If you require this, follow the instructions for stopping as specified below.

3.3 Test Instructions

1. The session operator will ask you to step on the treadmill. Remain stationary, holding onto the support beams.
2. Look at the screen and say "READY" when you wish to begin.
3. The treadmill will start. Keep holding the rails until you feel comfortable walking with no support. The session operator will begin the round.
4. When the round is over you can choose to take a break or carry onto the next round.
5. If you want to stop the treadmill, say "STOP" and hold onto the support rails until it has stopped moving completely. You will also be asked to do this at the end of the 15 rounds.

4 After the Test

When you have completed all 15 rounds of testing, the test operator will ask you to complete a survey about the testing process and the walking environment.

5 Data Protection

By signing this form, you agree that this study can store and use your data indefinitely unless you request it's removal. All data is stored in compliance with The University of York's data protection regulations.

6 Consent

If you have read all of the information above and are happy to proceed, please sign and date below.

Name: ______________________________

Signature: ______________________________

Date: ______________________________

E Web App Validation

E.1 GET_SWING_STANCE() FUNCTION

Below is the code for the **get_swing_stance()** MATLAB function.

```
function [swing_times,stance_times,stride_lengths] = get_swing_stance(maxima,minima,data,data_opposite,frame_rate)
%calculates swing/stance times and stride length using a list of the maxima
%and minima for both legs

    max_size = size(maxima,1);
    min_size = size(minima,1);
    max_counter = 1;
    min_counter = 1;
    swing_times = [];
    stance_times = [];
    stride_lengths = [];

    %while we are within the sizes of both arrays
    while (max_counter < max_size) && (min_counter < min_size)
        %set current events and next events
        heel_strike = maxima(max_counter);
        heel_raise = minima(min_counter);
        heel_strike_next = maxima(max_counter+1);
        heel_raise_next = minima(min_counter+1);

        %if the current heel raise occurs after the heel strike
        if heel_raise > heel_strike
            if heel_strike_next > heel_raise
                %stance time is from current heel strike to current heel raise
                stance = (heel_raise - heel_strike)/frame_rate;
                stance_times =  [stance_times stance];

            end
            max_counter = max_counter + 1;
        else
            if heel_raise_next > heel_strike
                %record stride length
                stride_length = data(heel_strike)-data(heel_raise);

                %swing time is from current heel raise to current heel strike
                swing = (heel_strike - heel_raise)/frame_rate;
                swing_times =  [swing_times swing];
                stride_lengths = [stride_lengths stride_length];
            end
            min_counter = min_counter + 1;
```

```
        end
    end
end
```

E.2 GET_DOUBLE_SUPPORT() FUNCTION

Below is the code for the **get_double_support()** MATLAB function.

```
function [double_times] = get_double_support(maxima,minima_other_leg,
    frame_rate)
%calculates double support times using a list of the maxima and minima for
    both legs

    max_size = size(maxima,1);
    min_size = size(minima_other_leg,1);
    max_counter = 1;
    min_counter = 1;
    double_times = [];

    %while we are within the sizes of both arrays
    while (max_counter < max_size) && (min_counter < min_size)
        %set current events and next events
        heel_strike = maxima(max_counter);
        heel_raise = minima_other_leg(min_counter);
        heel_strike_next = maxima(max_counter+1);

        %if the current heel raise occurs after the heel strike
        if heel_raise > heel_strike
            if heel_strike_next > heel_raise
                %record double support times
                double_support = (heel_raise - heel_strike)/frame_rate;
                double_times = [double_times double_support];

            end
            max_counter = max_counter + 1;
        else
            min_counter = min_counter + 1;
        end
    end
end
```

E.3 GET_METRICS() FUNCTION

Below is the code for the **get_metrics()** MATLAB function.

```
function [metrics,swing_times_l,stance_times_l] = get_metrics(ankle_l,ankle_r)
%calculates the gait metrics needed for the study

    %set frame rate and get number of frames
    frame_rate = 100;
    frames = size(ankle_l,1);

    %plot ankle z locations on a graph
    time = linspace(0,frames-1,frames);

    %get step frequency
    fundamental_l = get_fundamental(ankle_l,frame_rate);
    fundamental_r = get_fundamental(ankle_r,frame_rate);
    cadence_l = 1/(fundamental_l);
    cadence_r = 1/(fundamental_r);
    cadence = cadence_l + cadence_r

    %calculate mean and std dev for both ankles
    % mean_l = mean(ankle_l);
    % mean_r = mean(ankle_r);
    rms_l = rms(ankle_l)
    rms_r = rms(ankle_r)

    %find maxima in the data
    [peaks_l,locs_l] = findpeaks(ankle_l,'MinPeakDistance',(fundamental_l*frame_rate)*0.75,'MinPeakProminence',rms_l*0.75);
    [peaks_r,locs_r] = findpeaks(ankle_r,'MinPeakDistance',(fundamental_r*frame_rate)*0.75,'MinPeakProminence',rms_r*0.75);

    %find minima in the data
    [min_l,locs_min_l] = findpeaks(-ankle_l,'MinPeakDistance',(fundamental_l*frame_rate)*0.75,'MinPeakProminence',rms_l*0.75);
    [min_r,locs_min_r] = findpeaks(-ankle_r,'MinPeakDistance',(fundamental_r*frame_rate)*0.75,'MinPeakProminence',rms_r*0.75);
    min_l = -min_l;
    min_r = -min_r;

    %get swing, stance and single support (stance of opposite foot) times
    [swing_times_l,stance_times_l,stride_lengths_l] = get_swing_stance(locs_l,locs_min_l,ankle_l,ankle_r,frame_rate);
    single_times_r = stance_times_l;
    [swing_times_r,stance_times_r,stride_lengths_r] = get_swing_stance(locs_r,locs_min_r,ankle_r,ankle_l,frame_rate);
    single_times_l = stance_times_r;


    %get double support times
```

```
    double_times_l = get_double_support(locs_l,locs_min_r,frame_rate);
    double_times_r = get_double_support(locs_r,locs_min_l,frame_rate);

    %get average stride lengths
    stride_l = mean(stride_lengths_l);
    stride_r = mean(stride_lengths_r);

    %get average times
    swing_l = mean(swing_times_l);
    swing_r = mean(swing_times_r);
    stance_l = mean(stance_times_l);
    stance_r = mean(stance_times_r);
    double_l = mean(double_times_l);
    double_r = mean(double_times_r);
    single_l = mean(single_times_l);
    single_r = mean(single_times_r);

    %get ratio
    swing_ratio_l = swing_l/(swing_l + stance_l)*100;
    swing_ratio_r = swing_r/(swing_r + stance_r)*100;
    stance_ratio_l = stance_l/(swing_l + stance_l)*100;
    stance_ratio_r = stance_r/(swing_r + stance_r)*100;
    double_ratio_l = double_l/(double_l + single_l)*100;
    double_ratio_r = double_r/(double_r + single_r)*100;
    single_ratio_l = single_l/(double_l + single_l)*100;
    single_ratio_r = single_r/(double_r + single_r)*100;

    %get number of steps for each leg
    steps_l = size(peaks_l,1)-1;
    steps_r = size(peaks_r,1)-1;

    %get distance walked from stride lengths
    distance_l = sum(stride_lengths_l);
    distance_r = sum(stride_lengths_r);

    %calculate speed from distance values
    speed = (distance_l + distance_r)/(frames/frame_rate);

    %save gait metrics into a structure and return it
    metrics = struct(...
        'steps_left', steps_l,...
        'steps_right', steps_r,...
        'cadence_left_steps_s', cadence_l,...
        'cadence_right_steps_s', cadence_r,...
        'cadence_steps_s', cadence, ...
        'swing_time_left_s', swing_l, ...
        'swing_time_right_s', swing_r, ...
        'stance_time_left_s', stance_l, ...
        'stance_time_right_s', stance_r, ...
        'single_time_left_s', single_l, ...
        'single_time_right_s', single_r, ...
        'double_time_left_s', double_l, ...
        'double_time_right_s', double_r, ...
        'swing_time_left_ratio', swing_ratio_l, ...
```

```
        'swing_time_right_ratio', swing_ratio_r, ...
        'stance_time_left_ratio', stance_ratio_l, ...
        'stance_time_right_ratio', stance_ratio_r, ...
        'single_time_left_ratio', single_ratio_l, ...
        'single_time_right_ratio', single_ratio_r, ...
        'double_time_left_ratio', double_ratio_l, ...
        'double_time_right_ratio', double_ratio_r, ...
        'stride_length_left_m', stride_l, ...
        'stride_length_right_m', stride_r, ...
        'distance_left_m', distance_l, ...
        'distance_right_m', distance_r, ...
        'speed_ms', speed ...
    );

    %plot data with maxima and minima labeled and display metrics matrix
    plot(time,ankle_l,locs_l,peaks_l,'ob',locs_min_l,min_l,'ob',time,ankle_r,
        locs_r,peaks_r,'or',locs_min_r,min_r,'or');
end
```

E.4 GET_FUNDAMENTAL() FUNCTION

Below is the code for the **get_fundamental()** MATLAB function.

```
function [fundamental] = get_fundamental(data,fs)
% get_fundamental calculates the fundamental frequency of the step data

    len = length(data);

    % calculate the one-sided fourier transform of the data
    f_trans = abs(fft(data)/len);
    f_trans_os = f_trans(1:floor(len/2)+1);
    f_trans_os(2:end-1) = 2*f_trans_os(2:end-1);
    f_trans_os(1) = 0;

    % plot the fourier transform and return the fundamental frequency (secs
        per step)
    f = fs*(0:(len/2))/len;
    [peaks,locs] = max(f_trans_os);
    plot(f,f_trans_os,f(locs),peaks,'ob');
    fundamental = 1/f(locs);
end
```

Bibliography

[1] P. Mahlknecht, S. Kiechl, B. R. Bloem, J. Willeit, C. Scherfler, A. Gasperi, G. Rungger, W. Poewe, and K. Seppi, “Prevalence and burden of gait disorders in elderly men and women aged 60-97 years: a population-based study,” *PLoS One*, vol. 8, no. 7, p. e69627, Jul. 2013.

[2] S. Perera, K. V. Patel, C. Rosano, S. M. Rubin, S. Satterfield, T. Harris, K. Ensrud, E. Orwoll, C. G. Lee, J. M. Chandler, A. B. Newman, J. A. Cauley, J. M. Guralnik, L. Ferrucci, and S. A. Studenski, “Gait Speed Predicts Incident Disability: A Pooled Analysis,” *The Journals of Gerontology: Series A*, vol. 71, no. 1, pp. 63–71, 08 2015. [Online]. Available: https://doi.org/10.1093/gerona/glv126

[3] P. Mahlknecht, S. Kiechl, B. R. Bloem, J. Willeit, C. Scherfler, A. Gasperi, G. Rungger, W. Poewe, and K. Seppi, “Prevalence and burden of gait disorders in elderly men and women aged 60-97 years: a population-based study,” *PLoS One*, vol. 8, no. 7, p. e69627, Jul. 2013.

[4] J. Verghese, A. F. Ambrose, R. B. Lipton, and C. Wang, “Neurological gait abnormalities and risk of falls in older adults,” *Journal of Neurology*, vol. 257, no. 3, pp. 392–398, Mar 2010. [Online]. Available: https://doi.org/10.1007/s00415-009-5332-y

[5] A. T. Booth, A. I. Buizer, J. Harlaar, F. Steenbrink, and M. M. van der Krogt, “Immediate effects of immersive biofeedback on gait in children with cerebral palsy,” *Archives of Physical Medicine and Rehabilitation*, vol. 100, no. 4, pp. 598–605, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0003999318314497

[6] L. Y. Liu, S. Sangani, K. K. Patterson, J. Fung, and A. Lamontagne, “Real-time avatar-based feedback to enhance the symmetry of spatiotemporal parameters after stroke: Instantaneous effects of different avatar views,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 28, no. 4, pp. 878–887, 2020.

[7] E. de Villiers, T. Stone, N.-W. Wang, V. Sarangi, A. Pelah, and N. Shenker, “Virtual environment rehabilitation for patients with motor neglect trial (vermont): A single-center randomized controlled feasibility trial,” *Brain Sciences*, vol. 11, no. 4, 2021. [Online]. Available: https://www.mdpi.com/2076-3425/11/4/464

[8] D. Richmond, “Motion capture games and a social platform for clinical diagnostics, rehabilitation and exercise,” Master’s thesis, University of York, 2021.

[9] Laboratory for Investigative Virtual Environments (LIVE), “Stepsense clinic,” 2022.

[10] J. McKeown, “Stromoapp: A mobile gait and body symmetry analysis application using video pose estimation.” Master’s thesis, University of York, 2020.

[11] Our World In Data, "Covid-19 data explorer - our world in data." [Online]. Available: https://ourworldindata.org/explorers/coronavirus-data-explorer?facet=none&Metric=Confirmed+cases&Interval=7-day+rolling+average&Relative+to+Population=false&Color+by+test+positivity=false&country=~OWID_WRL

[12] W. Pirker and R. Katzenschlager, "Gait disorders in adults and the elderly: A clinical guide," *Wiener klinische Wochenschrift*, vol. 129, p. 2, 10 2016.

[13] S. Mihcin, S. Çıklaçandır, M. Koçak, and A. Tosun, "Wearable motion capture system evaluation for biomechanical studies for hip joints," *Journal of Biomechanical Engineering*, vol. 143, 12 2020.

[14] M. Pau, F. Corona, R. Pili, C. Casula, F. Sors, T. Agostini, G. Cossu, M. Guicciardi, and M. Murgia, "Effects of physical rehabilitation integrated with rhythmic auditory stimulation on spatio-temporal and kinematic parameters of gait in parkinson's disease," *Frontiers in Neurology*, vol. 7, 07 2016.

[15] AMTI, "Bms400600," 2022. [Online]. Available: https://www.amti.biz/product/bms400600/

[16] A. Gouelle and P. Roscher, "How do we use gait analysis?" last Accessed 05 April 2022. [Online]. Available: https://lermagazine.com/cover_story/gait-and-balance-academy-how-do-we-use-gait-analysis-to-measure-walking-consistency

[17] University of Delaware, "Rehabilitating knees," last Accessed 05 April 2022. [Online]. Available: https://engr.udel.edu/news/2019/04/rehabilitating-knees/

[18] "Virtual reality." [Online]. Available: https://en.wikipedia.org/wiki/Virtual_reality

[19] G. Jouvet, G. Cordonnier, B. Kim, M. Lüthi, A. Vieli, and A. Aschwanden, "Deep learning speeds up ice flow modelling by several orders of magnitude," *Journal of Glaciology*, p. 4, 12 2021.

[20] M. Yani, S. Irawan, and C. Setianingsih, "Application of transfer learning using convolutional neural network method for early detection of terry's nail," *Journal of Physics: Conference Series*, vol. 1201, p. 3, 05 2019.

[21] B. Ramsundar and B. Zadeh Reza, "Chapter 4. fully connected deep networks." [Online]. Available: https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html

[22] InData Labs, "Pose estimation for fitness and physical therapy application." [Online]. Available: https://indatalabs.com/resources/human-activity-recognition-fitness-app

[23] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," 2018. [Online]. Available: https://arxiv.org/abs/1812.08008

[24] Microsoft, "Azure kinect dk." [Online]. Available: https://www.microsoft.com/en-us/d/azure-kinect-dk/8pp5vxmd9nhq

[25] D. Mehta, O. Sotnychenko, F. Mueller, W. Xu, M. Elgharib, P. Fua, H.-P. Seidel, H. Rhodin, G. Pons-Moll, and C. Theobalt, "XNect: Real-time multi-person 3D motion capture with a single RGB camera," vol. 39, no. 4, July 2020. [Online]. Available: http://gvv.mpi-inf.mpg.de/projects/XNect/

[26] R. A. Güler, N. Neverova, and I. Kokkinos, "Densepose: Dense human pose estimation in the wild," *CoRR*, vol. abs/1802.00434, 2018. [Online]. Available: http://arxiv.org/abs/1802.00434

[27] Apple Inc., "Monitor your heart rate with apple watch." [Online]. Available: https://support.apple.com/en-us/HT204666

[28] A. Booth, M. van der Krogt, A. Buizer, F. Steenbrink, and J. Harlaar, "The validity and usability of an eight marker model for avatar-based biofeedback gait training," *Clinical Biomechanics*, vol. 70, pp. 146–152, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0268003319300191

[29] mrdoob, "three.js examples." [Online]. Available: https://threejs.org/examples/#webgl_animation_keyframes

[30] NaturalPoint, "Optitrack - flex 3 - an affordable motion capture camera," 2022. [Online]. Available: https://optitrack.com/cameras/flex-3/

[31] Scrum.org, "What is scrum?" 2022. [Online]. Available: https://www.scrum.org/resources/what-is-scrum

[32] Mike Cox, "Woman walking on a treadmill in a chicago gym," 2021, last Accessed 30 December 2022. [Online]. Available: https://unsplash.com/photos/06EpjZiMz_E

[33] MathWorks, "Linear or rank correlation," last Accessed 18 April 2023. [Online]. Available: https://uk.mathworks.com/help/stats/corr.html

[34] mrdoob, "How to create vr content." [Online]. Available: https://threejs.org/docs/index.html?q=vr#manual/en/introduction/How-to-create-VR-content

[35] J. Patience, K. S. P. Lai, E. Russell, A. Vasudev, M. Montero-Odasso, and A. M. Burhan, "Relationship between mood, thinking, and walking: A systematic review examining depressive symptoms, executive function, and gait," *The American Journal of Geriatric Psychiatry*, vol. 27, no. 12, pp. 1375–1383, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1064748119304361

[36] E. F. Ogawa, L. Shi, J. F. Bean, J. M. Hausdorff, Z. Dong, B. Manor, R. R. McLean, and S. G. Leveille, "Chronic pain characteristics and gait in older adults: The mobilize boston study ii," *Archives of Physical Medicine and Rehabilitation*, vol. 101, no. 3, pp. 418–425, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0003999319313061

[37] K. Spaniolas, J. D. Cheng, M. L. Gestring, A. Sangosanya, N. A. Stassen, and P. E. Bankey, "Ground level falls are associated with significant mortality in elderly patients," *Journal of Trauma and Acute Care Surgery*, vol. 69, no. 4, 2010. [Online]. Available: https://journals.lww.com/jtrauma/Fulltext/2010/10000/Ground_Level_Falls_Are_Associated_With_Significant.14.aspx

[38] World Health Organization, "Parkinson disease," 2022, last Accessed 10 October 2022. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/parkinson-disease

[39] M. E. Morris, R. Iansek, T. A. Matyas, and J. J. Summers, "The pathogenesis of gait hypokinesia in parkinson's disease," *Brain*, vol. 117 (Pt 5), pp. 1169–1181, Oct. 1994.

[40] B. R. Bloem, J. M. Hausdorff, J. E. Visser, and N. Giladi, "Falls and freezing of gait in parkinson's disease: A review of two interconnected, episodic phenomena," *Movement Disorders*, vol. 19, no. 8, pp. 871–884, 2004. [Online]. Available: https://movementdisorders.onlinelibrary.wiley.com/doi/abs/10.1002/mds.20115

[41] J. G. Nutt, B. R. Bloem, N. Giladi, M. Hallett, F. B. Horak, and A. Nieuwboer, "Freezing of gait: moving forward on a mysterious clinical phenomenon," *Lancet Neurol*, vol. 10, no. 8, pp. 734–744, Aug. 2011.

[42] Parkinson's UK, "Treatments and therapies for parkinson's." [Online]. Available: https://www.parkinsons.org.uk/information-and-support/treatments-and-therapies-parkinsons

[43] S. G. Reich and J. M. Savitt, "Parkinson's disease," *Med. Clin. North Am.*, vol. 103, no. 2, pp. 337–350, Mar. 2019.

[44] P.-L. Wu, M. Lee, and T.-T. Huang, "Effectiveness of physical activity on patients with depression and parkinson's disease: A systematic review," *PLoS One*, vol. 12, no. 7, p. e0181515, Jul. 2017.

[45] NHS, "Nhs england » musculoskeletal," last Accessed 10 October 2022. [Online]. Available: https://www.england.nhs.uk/elective-care-transformation/best-practice-solutions/musculoskeletal/

[46] World Health Organization, "Musculoskeletal health," 2022, last Accessed 10 October 2022. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/musculoskeletal-conditions

[47] Office for Health Improvement and Disparities, "Musculoskeletal health: applying all our health," last Accessed 30 December 2022. [Online]. Available: https://www.gov.uk/government/publications/musculoskeletal-health-applying-all-our-health/musculoskeletal-health-applying-all-our-health#:~:text=There\%20are\%203\%20groups\%20of,a\%20fall\%20from\%20standing\%20height

[48] K. R. Kaufman, C. Hughes, B. F. Morrey, M. Morrey, and K.-N. An, "Gait characteristics of patients with knee osteoarthritis," *Journal of Biomechanics*, vol. 34, no. 7, pp. 907–915, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021929001000367

[49] R. J. Weiss, P. Wretenberg, A. Stark, K. Palmblad, P. Larsson, L. Gröndal, and E. Broström, "Gait pattern in rheumatoid arthritis," *Gait & Posture*, vol. 28, no. 2, pp. 229–234, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0966636207003001

[50] National Multiple Sclerosis Society, "What is ms?" 2022. [Online]. Available: https://www.nationalmssociety.org/What-is-MS

[51] ——, "Ms signs & symptoms," 2022. [Online]. Available: https://www.nationalmssociety.org/Symptoms-Diagnosis/MS-Symptoms#section-0

[52] C. L. Martin, B. A. Phillips, T. J. Kilpatrick, H. Butzkueven, N. Tubridy, E. McDonald, and M. P. Galea, "Gait and balance impairment in early multiple sclerosis in the absence of clinical disability," *Multiple Sclerosis Journal*, vol. 12, no. 5, pp. 620–628, 2006, pMID: 17086909. [Online]. Available: https://doi.org/10.1177/1352458506070658

[53] U. Givon, G. Zeilig, and A. Achiron, "Gait analysis in multiple sclerosis: Characterization of temporal–spatial parameters using gaitrite functional ambulation system," *Gait & Posture*, vol. 29, no. 1, pp. 138–142, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0966636208002099

[54] S. J. Crenshaw, T. D. Royer, J. G. Richards, and D. J. Hudson, "Gait variability in people with multiple sclerosis," *Multiple Sclerosis Journal*, vol. 12, no. 5, pp. 613–619, 2006, pMID: 17086908. [Online]. Available: https://doi.org/10.1177/1352458505070609

[55] National Multiple Sclerosis Society, "Comprehensive care," 2022. [Online]. Available: https://www.nationalmssociety.org/Treating-MS/Comprehensive-Care

[56] Centers for Disease Control and Prevention, "What is cerebral palsy?" 2022. [Online]. Available: https://www.cdc.gov/ncbddd/cp/facts.html

[57] S. Armand, G. Decoulon, and A. Bonnefoy-Mazure, "Gait analysis in children with cerebral palsy," *EFORT Open Rev*, vol. 1, no. 12, pp. 448–460, Dec. 2016.

[58] National Institute for Neurological Disorders and Stroke, "Cerebral palsy," 2022. [Online]. Available: https://www.ninds.nih.gov/health-information/disorders/cerebral-palsy

[59] UK Health Security Agency, "Covid-19: epidemiology, virology and clinical features," 2020, last Accessed 30 March 2022. [Online]. Available: https://www.gov.uk/government/publications/wuhan-novel-coronavirus-background-information/wuhan-novel-coronavirus-epidemiology-virology-and-clinical-features

[60] World Health Organization, "Archived: Who timeline - covid-19," 2020, last Accessed 30 March 2022. [Online]. Available: https://www.who.int/news/item/27-04-2020-who-timeline---covid-19

[61] ——, "Coronavirus disease (covid-19)," 2020, last Accessed 30 March 2022. [Online]. Available: https://www.who.int/health-topics/coronavirus

[62] Lisa Maragakis, "Coronavirus diagnosis: What should i expect?" 2022, last Accessed 30 March 2022. [Online]. Available: https://www.hopkinsmedicine.org/health/conditions-and-diseases/coronavirus/diagnosed-with-covid-19-what-to-expect

[63] World Health Organization, "A clinical case definition of post covid-19 condition by a delphi consensus, 6 october 2021," 2021, last Accessed 30 March 2022. [Online]. Available: https://www.who.int/publications/i/item/WHO-2019-nCoV-Post_COVID-19_condition-Clinical_case_definition-2021.1

[64] F. Pistoia, R. Ornello, P. Sucapane, C. Marini, and S. Sacco, "Symptoms of gait and coordination impairment in a patient with covid-19 interstitial pneumonia," *Neurological Sciences*, vol. 42, no. 8, pp. 3083–3086, Aug 2021. [Online]. Available: https://doi.org/10.1007/s10072-021-05341-9

[65] H. Keklicek, H. Selçuk, İlke Kurt, S. Ulukaya, and G. Öztürk, "Individuals with a covid-19 history exhibit asymmetric gait patterns despite full recovery," *Journal of Biomechanics*, vol. 137, p. 111098, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021929022001518

[66] NHS, "Treatments for coronavirus (covid-19)," 2022, last Accessed 4 November 2022. [Online]. Available: https://www.nhs.uk/conditions/coronavirus-covid-19/self-care-and-treatments-for-coronavirus/treatments-for-coronavirus/

[67] Asthma + Lung UK, "Long covid care in the nhs," 2021, last Accessed 4 November 2022. [Online]. Available: https://www.blf.org.uk/support-for-you/long-covid/long-covid-care-in-the-nhs

[68] G. Rawal, S. Yadav, and R. Kumar, "Post-intensive care syndrome: an overview," *J Transl Int Med*, vol. 5, no. 2, pp. 90–92, 2017.

[69] M. E. Tinetti and C. S. Williams, "Falls, injuries due to falls, and the risk of admission to a nursing home," *New England Journal of Medicine*, vol. 337, no. 18, pp. 1279–1284, 1997, pMID: 9345078. [Online]. Available: https://doi.org/10.1056/NEJM199710303371806

[70] Boston Orthotics and Prosthetics, "Terminology of human walking," last Accessed 01 April 2022. [Online]. Available: https://www.bostonoandp.com/Customer-Content/www/CMS/files/GaitTerminology.pdf

[71] J. Hamill and M. Kathleen, *Biomechanical Basis of Human Movement Second Edition*. Lippincot Williams & Wilkins, 2003.

[72] T. Uchida, "Biomechanics of movement | lecture 7.4: Inverse kinematics: From marker locations to joint angles," 2022. [Online]. Available: https://youtu.be/R9DRhOy93RE

[73] ——, "Biomechanics of movement | lecture 8.1: Inverse dynamics: What is a joint moment?" 2022. [Online]. Available: https://youtu.be/zrsQIGK7Gys

[74] M. Doherty, J. Dacre, P. Dieppe, and M. Snaith, "The 'gals' locomotor screen." *Annals of the Rheumatic Diseases*, vol. 51, no. 10, pp. 1165–1169, 1992.

[75] M. E. Tinetti, T. Franklin Williams, and R. Mayewski, "Fall risk index for elderly patients based on number of chronic disabilities," *The American Journal of Medicine*, vol. 80, no. 3, pp. 429–434, 1986.

[76] S. Mathias, U. Nayak, and B. Isaacs, "Balance in elderly patients: the" get-up and go" test." *Archives of physical medicine and rehabilitation*, vol. 67, no. 6, pp. 387–389, 1986.

[77] NHS, "Physiotherapy," 2017, last Accessed 1 November 2022. [Online]. Available: https://www.nhs.uk/conditions/physiotherapy/

[78] S. Arambulo, "Difference between active and passive physiotherapy," last Accessed 1 November 2022. [Online]. Available: https://www.squareonephysio.ca/blog/active-passive-physiotherapy-mississauga

[79] H. E. Lowood, "virtual reality," last Accessed 06 April 2022. [Online]. Available: https://www.britannica.com/technology/virtual-reality

[80] HTC Corporation, "Products." [Online]. Available: https://www.vive.com/uk/product/

[81] Oculus, "Meta quest 2." [Online]. Available: https://store.facebook.com/gb/quest/products/quest-2/

[82] Valve Corporation, "Valve index." [Online]. Available: https://store.steampowered.com/valveindex

[83] Sony Interactive Entertainment, "Playstation vr." [Online]. Available: https://www.playstation.com/en-gb/ps-vr/

[84] D. Corbetta, F. Imeri, and R. Gatti, "Rehabilitation that incorporates virtual reality is more effective than standard rehabilitation for improving walking speed, balance and mobility after stroke: a systematic review," *Journal of Physiotherapy*, vol. 61, no. 3, pp. 117–124, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1836955315000569

[85] G. D. Yilmaz Yelvar, Y. Çırak, M. Dalkılınç, Y. Parlak Demir, Z. Guner, and A. Boydak, "Is physiotherapy integrated virtual walking effective on pain, function, and kinesiophobia in patients with non-specific low-back pain? randomised controlled trial," *European Spine Journal*, vol. 26, no. 2, pp. 538–545, Feb 2017. [Online]. Available: https://doi.org/10.1007/s00586-016-4892-7

[86] V. Zwass, "neural network," last Accessed 29 June 2022. [Online]. Available: https://www.britannica.com/technology/neural-network

[87] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann, "Mediapipe: A framework for building perception pipelines," 2019. [Online]. Available: https://arxiv.org/abs/1906.08172

[88] Google, "Solutions." [Online]. Available: https://google.github.io/mediapipe/solutions/solutions.html

[89] ——, "Mediapipe pose." [Online]. Available: https://google.github.io/mediapipe/solutions/pose

[90] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann, "Blazepose: On-device real-time body pose tracking," *CoRR*, vol. abs/2006.10204, 2020. [Online]. Available: https://arxiv.org/abs/2006.10204

[91] MediaPipe, "Mediapipe holistic." [Online]. Available: https://google.github.io/mediapipe/solutions/holistic.html

[92] Microsoft, "Azure kinect dk." [Online]. Available: https://azure.microsoft.com/en-us/services/kinect-dk/#overview

[93] Z. Liu, "3d skeletal tracking on azure kinect." [Online]. Available: https://www.microsoft.com/en-us/research/uploads/prod/2020/01/AKBTSDK.pdf

[94] R. Filkov, "Azure kinect examples for unity." [Online]. Available: https://assetstore.unity.com/packages/tools/integration/azure-kinect-examples-for-unity-149700

[95] The Editors of Encyclopaedia Britannica, "biofeedback," last Accessed 22 April 2022. [Online]. Available: https://www.britannica.com/science/biofeedback

[96] Vercel Inc., "Next.js by vercel." [Online]. Available: https://nextjs.org/

[97] Meta Platforms, Inc., "React - a javascript library for building user interfaces." [Online]. Available: https://nextjs.org/

[98] Vercel Inc., "What is next.js?" [Online]. Available: https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs

[99] Tailwind Labs Inc., "Tailwind css - rapidly build modern websites without ever leaving your html." [Online]. Available: https://tailwindcss.com/

[100] ——, "Flex." [Online]. Available: https://v2.tailwindcss.com/docs/flex

[101] ——, "Grid." [Online]. Available: https://v2.tailwindcss.com/docs/display#grid

[102] mrdoob, "Three.js - javascript 3d library." [Online]. Available: https://threejs.org/

[103] NaturalPoint, “Optitrack - motion capture systems,” 2022. [Online]. Available: https://optitrack.com/

[104] Vicon Motion Systems Ltd, “Vicon | award winning motion capture systems.” [Online]. Available: https://www.vicon.com/

[105] R. Baker, “Gait analysis methods in rehabilitation,” *Journal of NeuroEngineering and Rehabilitation*, vol. 3, no. 1, p. 4, Mar 2006. [Online]. Available: https://doi.org/10.1186/1743-0003-3-4

[106] NaturalPoint, “Optitrack - support - tracking tools,” 2022. [Online]. Available: https://optitrack.com/support/software/tracking-tools.html

[107] ——, “Optitrack - support - flexv100,” 2022. [Online]. Available: https://optitrack.com/support/hardware/flex-v100.html

[108] ——, “V100-r2 data sheet,” 2012. [Online]. Available: https://d111srqycjesc9.cloudfront.net/V100-R2\%20Data\%20Sheet.pdf

[109] Atlassian, “Jira | issue & project tracking software | atlassian,” 2022. [Online]. Available: https://www.atlassian.com/software/jira

[110] Pallets, “Welcome to flask - flask documentation (2.2x),” 2022. [Online]. Available: https://flask.palletsprojects.com/en/2.2.x/

[111] Mozilla Corporation, “Blob,” 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Blob#:~:text=The\%20Blob\%20object\%20represents\%20a,in\%20a\%20JavaScript\%2Dnative\%20format.

[112] ——, “Mediarecorder - web apis|mdn,” 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder

[113] MediaPipe, “Mediapipe - pose.” [Online]. Available: https://codepen.io/mediapipe/pen/jOMbvxw

[114] mrdoob, “Raycaster.” [Online]. Available: https://threejs.org/docs/?q=ray#api/en/core/Raycaster