university of groningen

faculty of science and engineering

Architecture and Implementation of a Gait Diagnosis System

Bachelor's Project Computing Science

June 2024

Author: Nikola Zivkovic
Student Number: S4796136
First supervisor: Dimka Karastoyanova
Second supervisor: Michel Medema
External supervisor: Dr. C. Greve

Abstract

This thesis addresses the urgent need for an improved gait diagnosis system at the University Medical Center Groningen (UMCG), where the current system's inefficiencies result in prolonged wait times and low patient throughput. The existing system's lack of industry-standard design principles further hampers its effectiveness in diagnosing gait disorders promptly. This research aims to enhance the system's architecture by integrating industry-standard design principles, efficient back-end technologies, and complex data processing techniques. Additionally, the project focuses on developing a user-friendly interface to facilitate seamless interaction for medical professionals. By automating and streamlining the diagnostic process, the new system aims to significantly reduce patient wait times while providing reliable and accurate diagnoses. The expected outcomes include substantial improvements in the system's efficiency, reliability, and user experience, ultimately benefiting both patients and healthcare practitioners at UMCG. This research not only contributes to the technical advancement of gait diagnosis systems but also aims to enhance the overall quality of patient care.

Contents

List of Figures

Acknowledgments

First and foremost, I would like to extend my heartfelt gratitude to my supervisors, Dimka Karastoyanova and Michel Medema, for their unwavering support throughout the project and for providing me with the opportunity to develop this software. I am also deeply grateful to our external supervisor, Dr. Christian Greve, for his availability to answer questions about the requirements and for facilitating visits to the gait diagnosis lab at UMCG. Additionally, I want to express my sincere thanks to my teammates, Emmanouil Kalostypis and Amr Rashad, for their collaboration and teamwork, which were vital in completing this project.

1 Introduction

Gait disorders and abnormalities affect approximately 15% of people by the age of 60 [13]. Additionally, more than 80% of people over 85 have a gait abnormality. Moreover, gait disorders and abnormalities also affect children through cerebral palsy, functional limb weakness [18], Phelan-McDermid Syndrome [5] and idiopathic scoliosis [14]. Over 123,000 [2] of the population in the Groningen province are aged 65 and above. This affects a large group of people and diagnosing gait deficiencies, abnormalities, or any gait-related trouble should be attended to as soon as possible. The importance of utilizing gait diagnosis is clinically useful [8] for patients who have impairment in their walking. Thus, the importance of providing the help people require is quintessential for their livelihood. Currently, the process in the University Medical Centre Groningen (UMCG) is a lengthy, laborious process that requires long waiting times for patients and a low patient throughput as a result.

Consequently, a gait diagnosis system can solve this problem within hospitals to reduce lengthy wait times, decrease labour and increase patient throughput. A gait diagnosis system is currently a work in progress in the medical field. There are prominent platform-based technologies [3] but a reliable, fast and elegant solution in the UMCG does not exist at the moment.

The objective of this paper is to provide UMCG with a prototype gait diagnosis system. This will allow the process of gait diagnosis to be sped up tremendously and allow for future projects to build upon the system. The thesis provides a very solid foundation for future work.

An efficient gait diagnosis is proposed by developing the complete architecture design and implementation of a system within the UMCG, alongside automation of the current systems to enhance and speed up current processes to ensure patients are getting diagnoses as accurately and fast as possible. Therefore, emphasizing the importance of the quality of work that this thesis produces. The research will utilize data provided by the UMCG and the current system to completely rebuild a new system. Later, this system could be utilized by other medical clinics and hospitals allowing us to develop and train the system to have better diagnoses. This leads to the following research question:
How can the architecture and implementation of a gait diagnosis system be designed for increased efficiency, incorporating industry-standard design principles and architectural patterns?

From this the following sub-research questions can be produced:

1. How can the integration of automated processes within the gait diagnosis system reduce patient wait times and increase patient throughput at UMCG?

2. What specific architectural patterns and design principles can be applied to the development of an automated gait diagnosis system to improve efficiency and accuracy?

Answering these research questions are expected to make crucial contributions to the development and architecture of a gait diagnosis system.

The factors mentioned above reveal the source of motivation behind developing a gait diagnosis system. Both patients and doctors at UMCG will benefit from a high-quality system that diagnoses patients and incorporates doctors' input as well for a diagnosis. More specifically, the thesis builds upon a gait diagnosis tool built by Dr. C. Greve from the UMCG, and a previous RUG student. The current system architecture lacks industry-standard design and architectural patterns which are required for a system with such importance. Due to the overall diagnosis process being quite slow and requiring manual labour, patients have quite long waiting times for appointments. Since walking is an important part of daily life for most people it would be quite important to provide an automated gait diagnosis system.

Given this, Section 2 describes in detail, gait cycles and phases as well as the current gait diagnosis process within the UMCG. Section 3 introduces all of the related work regarding Gait diagnosis and Gait diagnosis systems. Section 4 introduces the methodologies and approach that the thesis uses as well as outlines some requirements set by ourselves, the thesis supervisor and Dr. C. Greve. Section 5 introduces the architecture that is utilized for the system. Section 6 discusses the technology stack that is utilized for the system and the reasoning behind it. Section 7 is the implementation of the system and the individual contribution to the project. Section 8 discusses the results and prototype produced by the members of the project. Section 9 discusses the conclusion and takeaways from the thesis. Section 10 provides insights into future work. The references and appendices are the last part of the paper.

2 Gait Diagnosis Process within UMCG

2.1 Gait Analysis

A patient's gait is analyzed in the UMCG through the following: Speed of walking, stride length, rhythm, body mechanics, and Electromyography(EMG) of muscle activity. This system focuses on all of that apart from the EMG data. A patient's gait is described by a gait cycle. The gait cycle refers to the repetitive pattern of movements that happen during walking. It begins when the heel of one foot makes contact with the ground and concludes when that same heel touches the ground again. Within a gait cycle, there are gait phases called the stance phase and swing phase. The stance phase is the period of the gait cycle when the foot is on the ground and bearing body weight. The swing phase is the second phase of gait when the foot is free to move forward [6]. Here is a diagram representing a gait cycle and the respective phases:

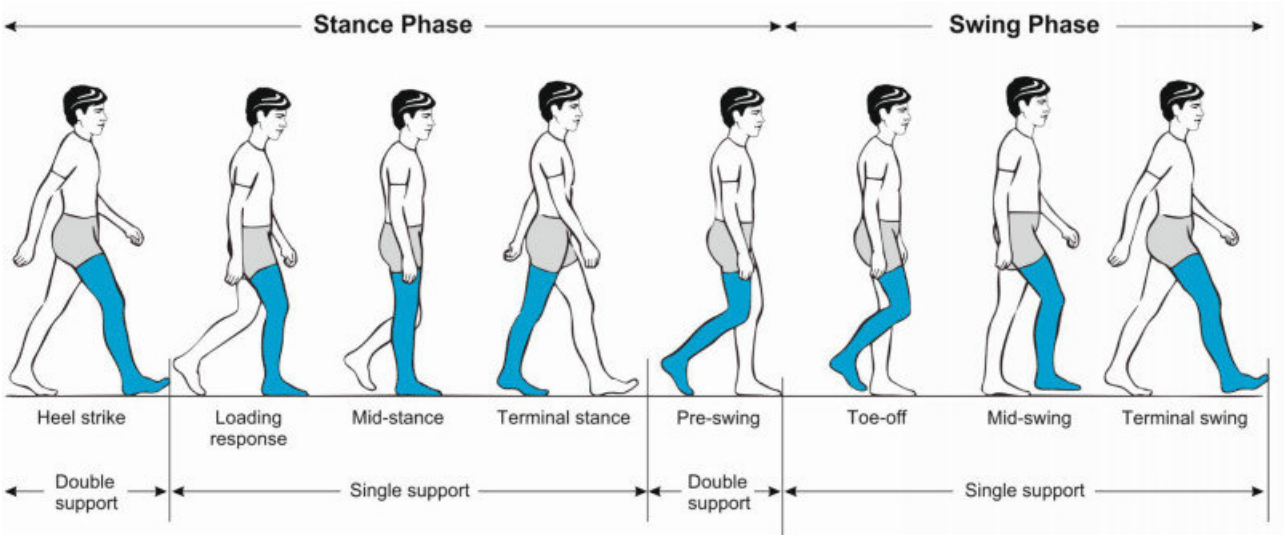

Figure 1: Gait Cycle and Phases Diagram [10].

2.2 Physical Assessment & Data Collection

The patient visits the facility in the UMCG to have Dr. C. Greve and other medical professionals conduct a physical assessment of the patient which is input onto an `.xlsx` file called LO in Dutch. Once the assessment is complete, the doctors attach optical markers to the patient from their hips down to work with the cameras and sensors placed around the room. Subsequently, the doctors use the platform in the UMCG to collect data on the patient's gait through their gait cycle and phases. The data collection process takes around 90 minutes to 120 minutes on a case-by-case basis. This can be described in the following diagrams:

Figure 2: Collection of patient gait data.

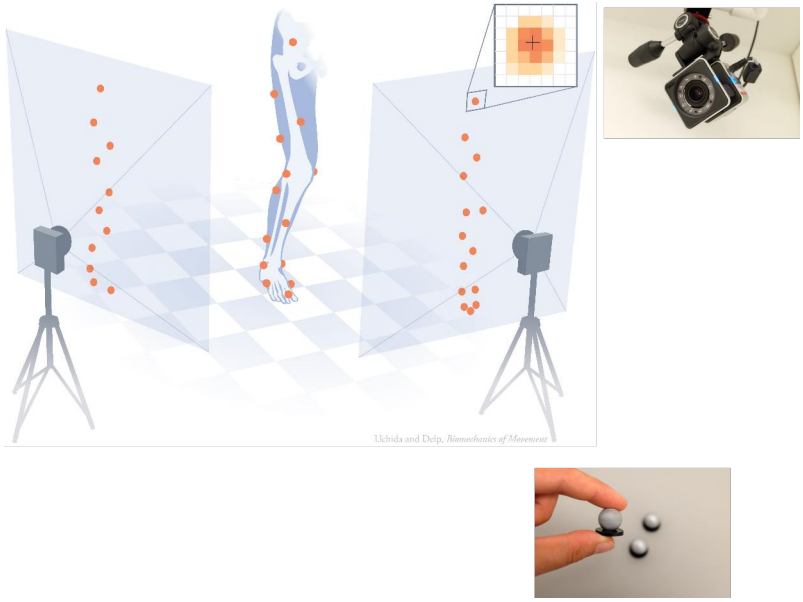

Figure 3: Sensors and optical markers diagram representing facility within the UMCG.

2.3 Data Analysis & Diagnosis

After the data has been gathered the doctors have to interpret it and analyse it to provide the patient with a diagnosis. This is all done manually by Dr. C. Greve and the other medical professionals at the UMCG. Using the physical examination data and going frame by frame in the `.c3d` file which is a standard file type utilized within most hospitals in the Netherlands for gait analysis, that contains all the information needed to read, display, and analyze 3D motion data with additional analogue data from force plates, electromyography and other sensors [15]. Using the `.c3d` file they can check the differentiation from a normal gait to produce a diagnosis. This data interpretation and analysis can take up to 60 to 120 minutes depending on how severe the gait abnormality is. This then concludes the diagnosis and the doctors can provide the patient with an extensive diagnosis. This is quite slow and is the main cause of delays for patients within the UMCG.

3 Related Work

3.1 Existing Work

A gait diagnosis system consists of analyzing a patient's gait to diagnose the patient with the reason why they may have difficulty walking. A gait difficulty tends to be caused in patients who have experienced a stroke, or Parkinson's disease or suffer from cerebral palsy [1, 11, 16]. Additionally, flat foot tends to be a common reason for trouble with gait [9]. Currently, there are a few gait diagnosis tools in order to help patients get a diagnosis of problems regarding their gait. The most relevant concerning the gait diagnosis process at the UMCG is, platform-based gait technologies [3] that utilize cameras, kinematic force pressure plates, sensors and optical markers to gather data on the gait of a patient. Apart from that, other solutions consist of in-shoe systems, a textile gait analysis platform and a cloud-based platform. Additionally, a recent paper on the interpretation of gait data in children with cerebral palsy produced a set of look-up tables to support the complicated process of understanding and processing gait data [17].

3.1.1 In-Shoe System

An In-Shoe system [4] consists of a sensor being attached to the patient's foot and based on this detects abnormalities in the patient's gait and performs a gait analysis in real-time via a smartphone. The system is more portable and suitable for daily use and plays an important role in monitoring the safety of the elderly. Its limitations are that it is difficult to set up initially and could be hard to distribute to patients. Additionally, it is quite different from a platform-based technology that is currently being utilized within the UMCG as it is built into a shoe. However, this is easier than having to commute to a hospital and to receive consultation. Here is a diagram of the in-shoe sensor:

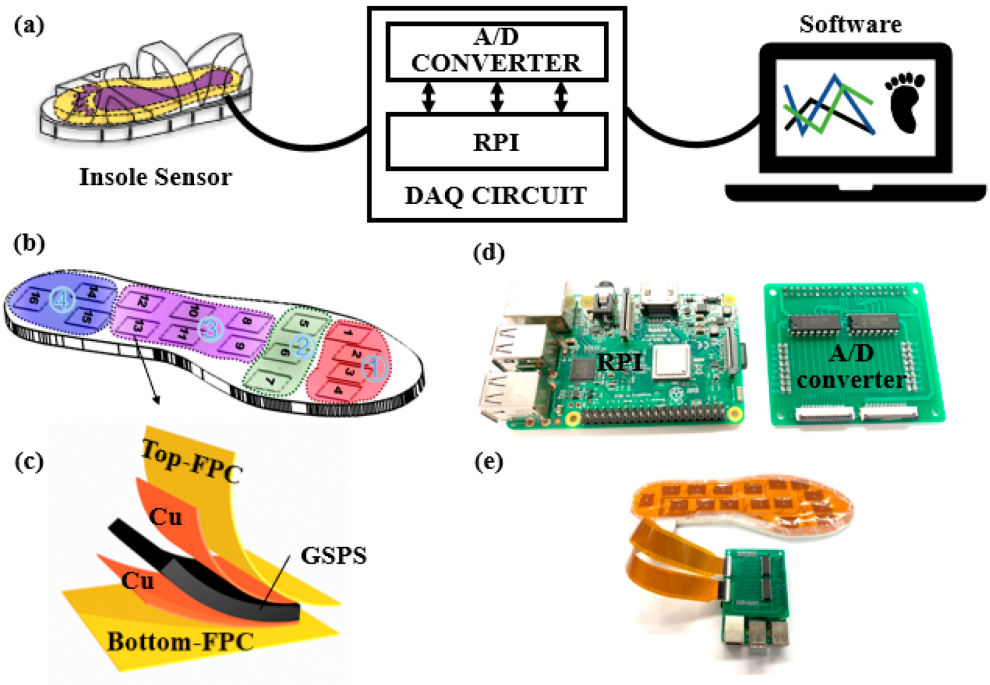

Figure 4: Diagram representing the in-shoe sensor [4].

3.1.2 Textile Gait Analysis Platform

Another finding discovered recently, towards the end of 2023, is a paper that was published proposing a "Robust and breathable all-textile gait analysis platform" [19] which produced a better platform-based gait analysis through an "ATPSA-LeNet system" and deep learning methods they were able to transform the plantar pressure into datasets through the different pressure sensor arrays built into the fabric using a "classifier level sensor fusion platform." Showing a brand new innovative way of utilizing a combination of neural networks and already existing platform-based gait analysis. The limitations of this textile platform are that it is quite expensive to set up and requires a deep understanding of the complex system utilized alongside it. Below is a diagram representing the platform:

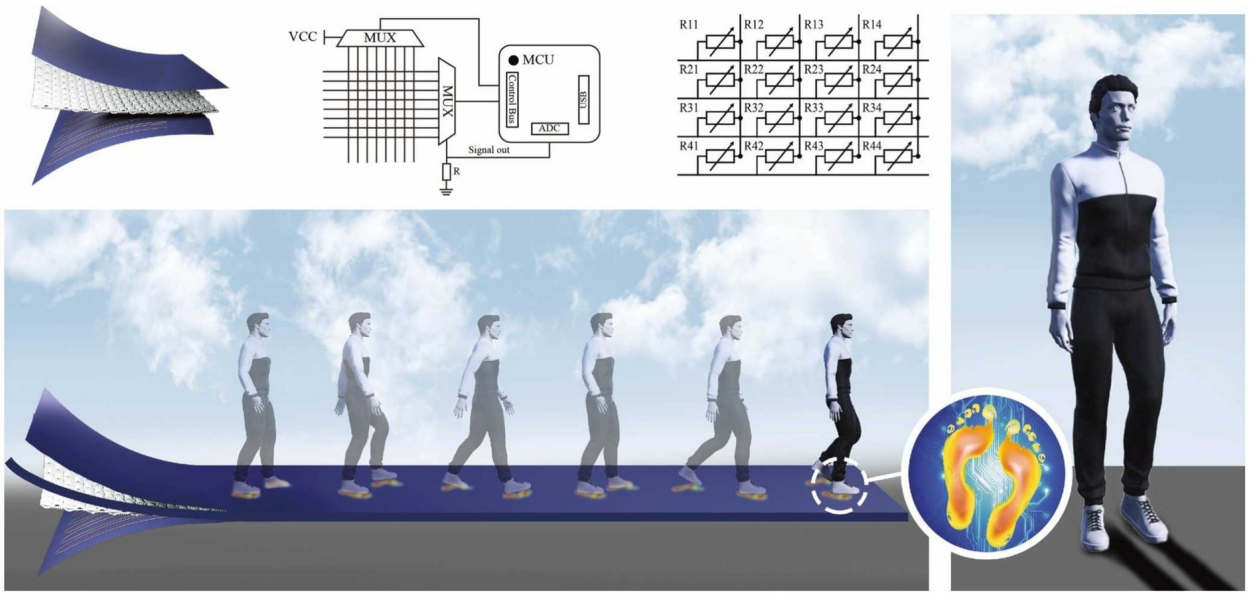

Figure 5: Robust and breathable all-textile gait analysis platform [19].

3.1.3 **Cloud Based Gait Analysis**

In very recent research, a paper was released suggesting a "cloud-based platform for comprehensive gait analysis" not requiring a force reading or pressure plate platform. This allows patients to send videos of themselves walking at home or being able to go to a clinic to get video captured at the facility. The main takeaway from this is that it is cheaper and easier to facilitate for the elderly as they would not need to go to the hospital if not required. A limitation is that it does not gather as much data as a platform-based gait analysis would be able to which could lead to a more inaccurate diagnosis. This is all uploaded to the platform producing a revolutionary way of performing gait analysis for patients and for doctors.[7]. Below is a diagram displaying the cloud-based platform:

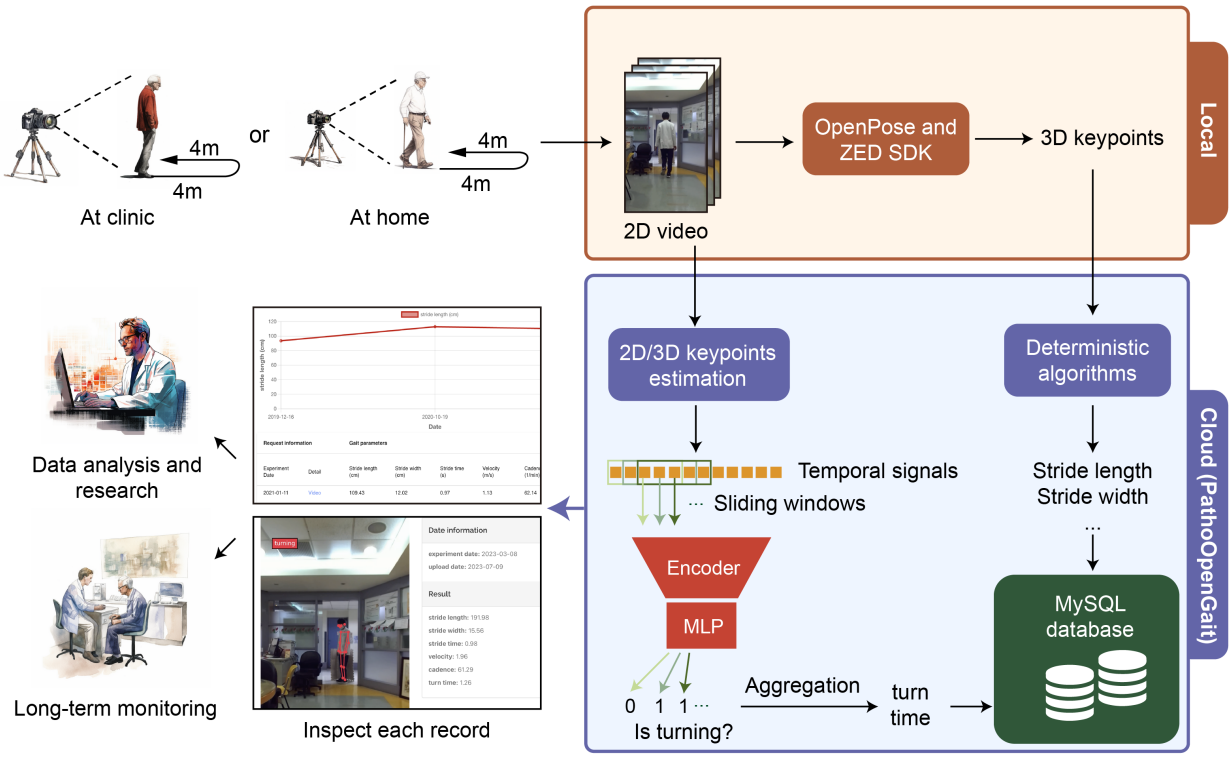

Figure 6: cloud-based platform for comprehensive gait analysis [7].

In conclusion, there are a multitude of different approaches and technologies available currently for gait analysis and there are many new and innovative ways being published yearly.

3.2 Current Work

A gait diagnosis tool currently exists within the UMCG and is presently being used. Dr. C. Greve and Adrian Segura Lorente both implemented a system for Gait Analysis to be utilized at the UMCG [12]. The UMCG utilizes a platform-based gait analysis where the platform is set up at a facility, shown in Figure 2, within the UMCG. This data is gathered using cameras, sensors, optical markers and pressure plates. This data is of the `.c3d` file type. The current system can be represented through this diagram:

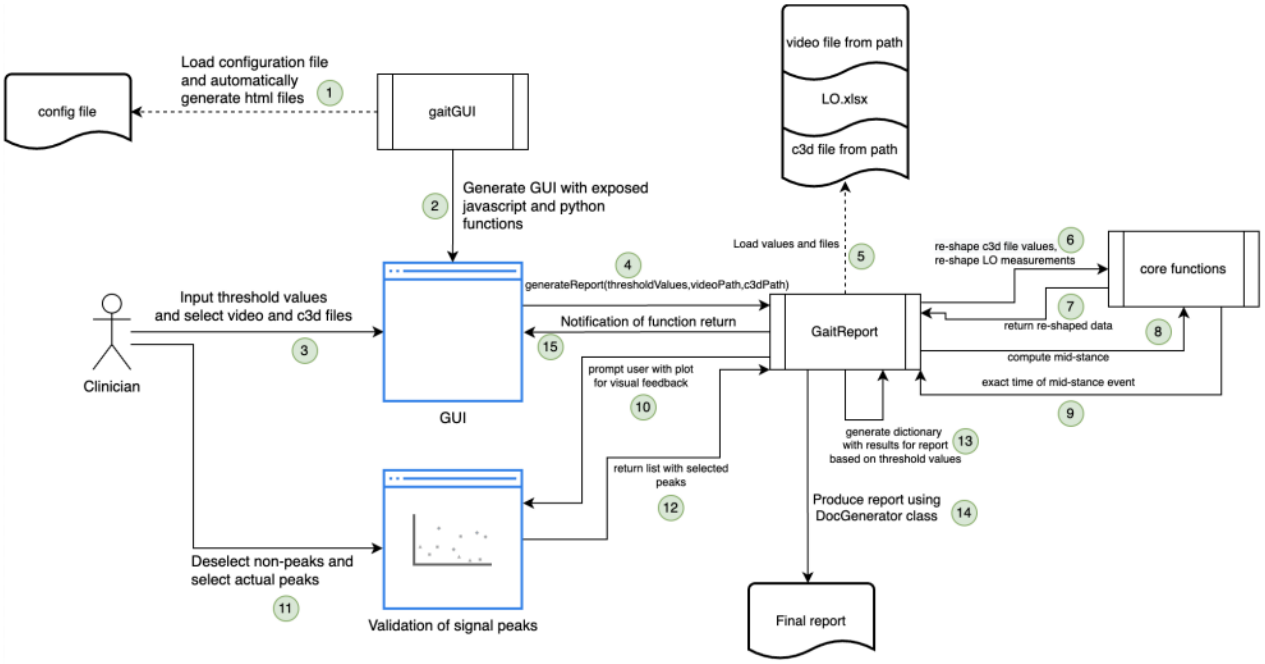

Figure 7: Diagram of the System overview [12]

Given this diagram, the current system consists of the following: A doctor has to upload a configuration file alongside input threshold values and select the corresponding video and c3d files. The video and `.c3d` files come from the sensors and optical markers. The angles of each foot at each point also need to be inputted. Following this, a report is generated displaying the data and images so that a relevant medical professional can analyze it and give a diagnosis. Currently, a large amount of the processes involve manual entry and many parts of it could be improved and automated to decrease waiting time. Additionally, the automation can allow us to give automated diagnoses that doctors would overlook to comment on the accuracy.

The honours project aimed to enhance the system by addressing the limitations of the current process at the UMCG. The existing tool was partially developed and had several issues, including hard-coded threshold values, broken functionality, and a lack of automation. Additionally, the lack of a Graphical User Interface (GUI) makes it quite hard for medical professionals to follow and to be able to get a diagnosis from the system.

The honours project also highlights the importance of restructuring the code to allow for better maintainability and extensibility. The system lacks automation. Moreover, the project identified further potential improvements, such as enhancing the robustness of the system and integrating additional features to support more comprehensive gait analysis. This alongside the decision trees provided to us by the experts at the UMCG is going to be of great help to produce a gait diagnosis system.

The goal of this research project is to automate and enhance the current processes that are being executed in the current gait diagnosis system. As mentioned in the report of the system, “At the moment, the main function still has hard-coded threshold values.” From this, we can gather that values are being hard-coded and the majority of the system requires improvement. The system does not utilize any architecture patterns and lacks coding practices. This can be problematic as it can cause issues regarding scalability, maintainability and reliability. Currently, gathering the data takes approximately 90-120 minutes and then processing and developing a diagnosis and report on this data takes 60-120 minutes. The processing of the data and producing the report is taking too long which leads to not that many patients having the opportunity to be examined.

In comparison to existing work in the gait analysis field, the facilities within the UMCG are very similar to platform-based gait analysis systems but the system itself is lacking in multiple aspects.

3.3 Search Terms

The results gathered above were done through two main approaches. Firstly a search of multiple combinations of keywords consisting of: ”gait”, ”gait diagnosis”, ”gait analysis”, ”force platform”, and ”platform-based”. The results of these queries were used on digital libraries such as SmartCat, World of Science and IEEE Explore provided as useful resources to gather relevant references.

4 Methods & Requirements

For this research, the UMCG provides a data set to be used for the development of the system. This data set will contain patients' data entailing: the session where the patient walks on the gait analysis platform and is provided to us through a `.c3d` file and a physical examination file called `LO`. Consequently, the current gait diagnosis tool is provided to us to work on the system's implementation and architecture. After a full comprehensive understanding of the current system, designing and implementing an architecture consisting of a front-end, back-end and data-processing model is required to produce a gait diagnosis system. Once appropriate frameworks are chosen, everything is implemented in one system, utilizing the EPIC database provided by the UMCG. With input from Dr. C. Greve, multiple runs and tests are conducted to ensure the diagnosis system functions as intended. Once this is complete, the prototype can be deemed as finished representing the new and improved system.

4.1 Approach

The methodology and approach that are used include an industry-standard front-end and back-end alongside a data processing module. The approach utilized for developing the gait diagnosis system is explained in the subsequent sections.

4.2 Data Processing & Analysis

The data processing and analysis is the largest part of the system. Extracting data from files, calculating different stances and comparing data to normative data.

4.2.1 Extracting Data from the `.c3d` Files

The UMCG provided .c3d files, a standard format for storing 3D motion data used in gait analysis. One file represents normal gait data, while another contains patient data. Using a data-extracting library for c3d files is most appropriate to gather all relevant data for a diagnosis.

4.2.2 Extracting Data from the Physical Examination (LO)

Data from physical examinations are provided in `.xlsx files`, detailing the patient's leg movement capabilities. The physical examination files can have different formats and data so, a Python script can dynamically parse this facilitating flexible data handling for the diagnostic process.

4.2.3 Stance Calculation

The different gait stances need to be identified and split in the data processing module.

4.2.4 Data Comparison

Joint angles during different gait phases need to be compared against normal ranges. Deviations of 8 degrees or more were flagged to identify gait abnormalities. 8 degrees is the agreed-upon value decided by Dr. C. Greve.

4.3 Back-end Development

Several endpoints are created including file upload and producing the diagnosis.

4.3.1 File upload endpoints

Endpoints for `.c3d` and `.xlsx` file uploads need to be implemented, ensuring secure and validated file handling. Uploaded files are saved in specific directories in order to be handled by the data processing module.

4.3.2 Diagnosis endpoint

A diagnosis produced by the data-processing module needs to be sent to the front-end to display.

4.4 Frontend Development

The front-end connects to the back-end to facilitate user interactions and display diagnosis results.

4.4.1 Dashboard

Users need to be able to upload `.c3d` and `.xlsx` files. The dashboard provides real-time feedback on file uploads and displays the gait analysis results in a user-friendly table format, ensuring easy interpretation by medical professionals as they are the system's intended users.

4.5 Requirements

For the thesis, requirements need to be set to help structure and plan what would have to be completed so that the system could be considered a success. Each requirement has a unique ID, denoted as `[R][numberR]-[F/NF]` where:

- `[R]`: Requirement
- `[numberR]`: the corresponding number of the requirement
- `F/NF`: meaning functional/non-functional

The requirements the planning produces are:

1. `[R-1-F]` The system should allow users to upload a .c3d file and a physical examination file
2. `[R-2-F]` The system should display the diagnosis for the respective patient
3. `[R-3-F]` The user should securely be able to login and logout using the appropriate credentials
4. `[R-4-F]` The system should produce a PDF representing the diagnosis report that the user could download
5. `[R-5-NF]` The system should be easy to use and understand for medical professionals
6. `[R-6-NF]` The system should display a diagnosis under 5 seconds after successful upload

During the development, Git will be used for source-code management. To avoid any serious loss of code due to hardware failure, the code will be regularly backed up to a GitHub repository. Since the data from the patients is quite sensitive and personal, the data will only be used internally to develop a gait diagnosis system.

5 Architecture

5.1 High-Level System Diagram & Process Steps

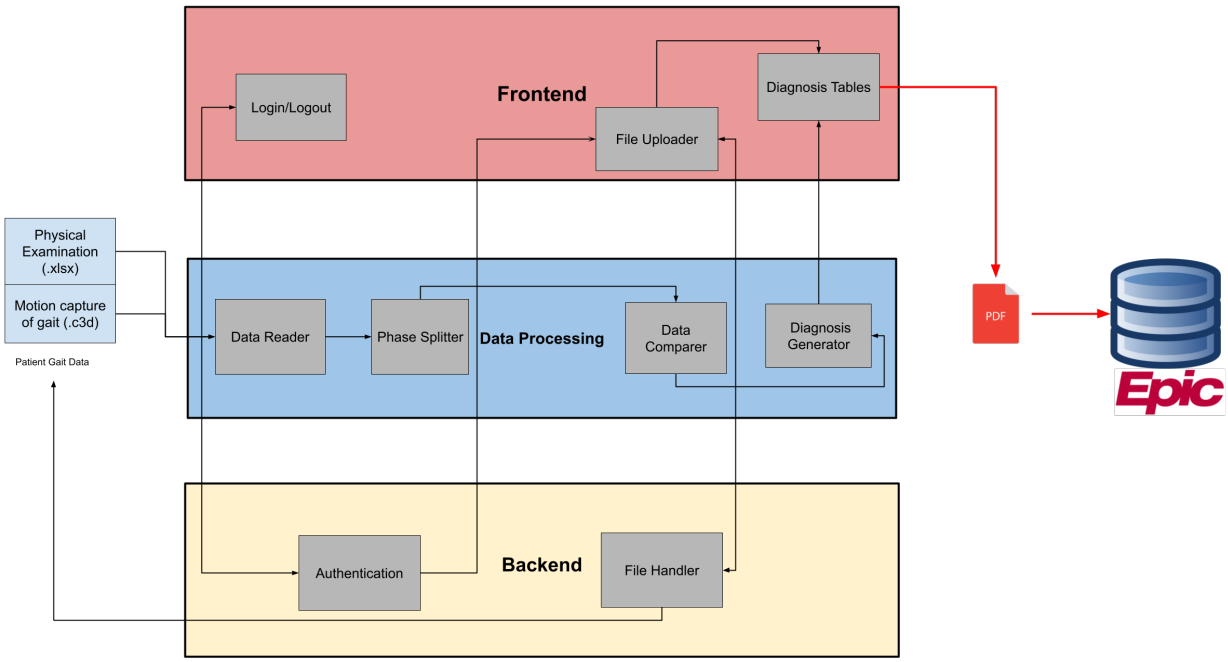

Figure 8: System Architecture Diagram entailing the different modules

The System Diagram 8 depicts the modular architecture that is employed throughout the Gait Diagnosis System. The architecture is split into three modules consisting of a front-end, back-end and data processing module each containing sub-components. A modularized architecture is the most appropriate for the project and requirements as it allows the system to have maintainability as each module can be independently maintained leading to easier debugging. Furthermore, it allows for scalability as additional features can be added as new modules or added to existing ones. It also allows for collaboration among the members developing the system to work on separate modules which is very suitable for this thesis. Lastly, it offers re-usability by isolating different functionalities into modules leading to the system becoming more robust. This leaves a strong and decoupled architecture that will allow others to contribute to the project and also make it scalable for future work.

The following sections describe in detail the different modules and the components within them:

5.2 Front-end Module

The front-end module of the gait diagnosis system serves as the user interface, facilitating interactions between the end-users, otherwise known as medical professionals, and the system. This module is crucial for ensuring a smooth and efficient workflow, as it handles user authentication, data input, and the presentation of diagnostic results. The front-end is composed of the following components:

5.2.1 Authentication Component

The authentication component takes the user input for login credentials and sends an HTTP request through a login endpoint to the back-end to verify that the credentials are correct. It receives a response from the back-end and if successful navigates the user to the file uploader component, otherwise known as the dashboard page of the system.

5.2.2 File-Uploader Component

The file uploader component facilitates the input of patient data into the system. This allows users to upload files containing physical examination data `.xlsx)` and motion capture data of gait `.c3d` to be sent to the back-end. Once this is successful, the data is sent to the data processing module to produce a diagnosis. Once the diagnosis is generated it is sent to the diagnosis tables component.

5.2.3 Diagnosis Tables Component

The diagnosis tables component displays the processed diagnostic information to the users. The component presents the results in an organized and comprehensible manner, making it easy for healthcare professionals to interpret and utilize the diagnostic outcomes.

5.3 Data Processing Module

The data processing module handles the core computational tasks required to read the data, split the data into the gait phases, compare it to normative data and generate diagnostic results. It includes the following components:

5.3.1 Data-Reader Component

The data-reader component reads and interprets the uploaded data files. It extracts relevant information from physical examination data `.xlsx` and motion capture data `.c3d` for further processing. The data is then sent to the phase splitter component.

5.3.2 Phase Splitter Component

The phase splitter component segments the motion capture data into different phases of gait shown in figure 1. Once complete the data is sent to be compared to normative data.

5.3.3 Data Comparer Component

This component compares the segmented gait data against normative gait data. The component identifies deviations that may indicate potential issues in the patient's gait. When all the data is compared it is sent to the diagnosis generator.

5.3.4 Diagnosis Generator Component

After receiving all the data and its comparisons a diagnosis is generated highlighting any detected abnormalities and highlighting if parts of the data is normal comparing to normative data. Once a diagnosis is created it is sent to the diagnosis tables component to be displayed.

5.4 Back-end Module

The back-end module manages the underlying system operations, ensuring secure data handling and authentication processes. The back-end performs these operations through endpoints. The module includes the following components:

5.4.1 Authentication Component

The authentication component ensures that only authorized users can access the system and its functionalities, maintaining the security and integrity of sensitive medical data. This is done through the login and logout endpoints.

5.4.2 File Handler Component

The file handler component is called upon by the file uploader component when the user uploads the patient data. This is done through the file upload endpoints for each file type. Once successful this is sent to the data processing module to produce a diagnosis.

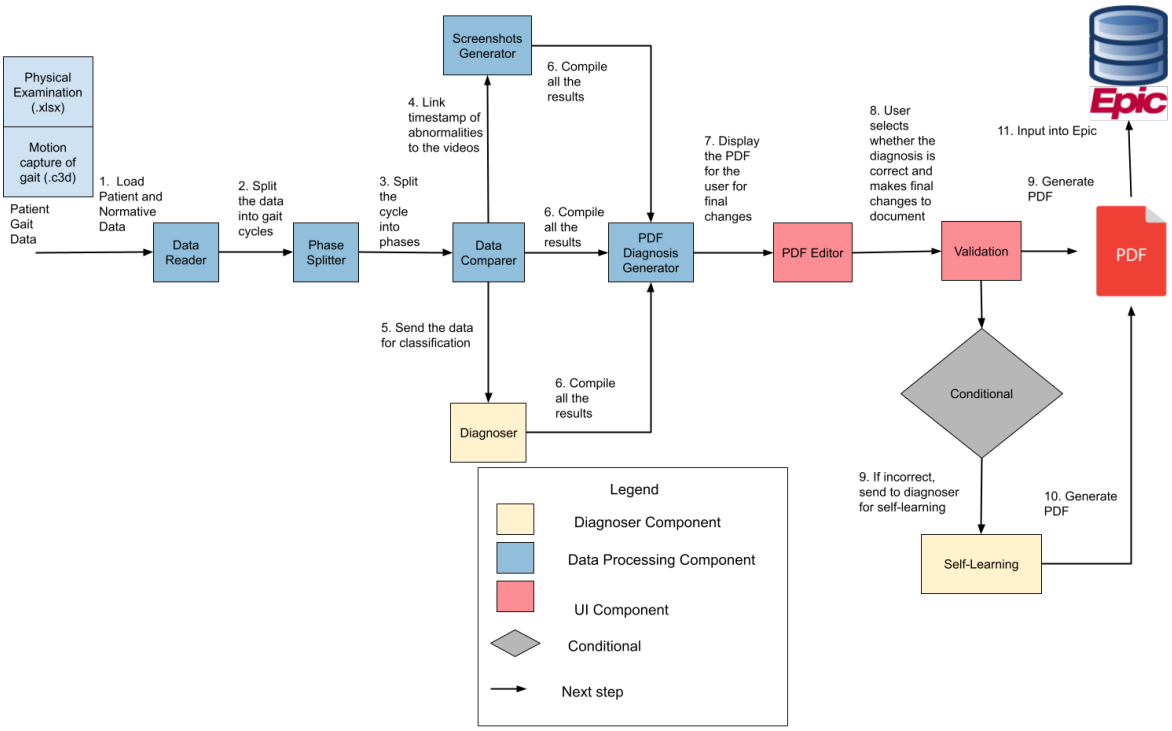

Figure 9: Flow diagram entailing the process of the system

The flow diagram goes more into depth explaining the whole process.
In steps one, two and three the patient data is loaded in alongside the normative data provided by the UMCG. This data is then split into its respective gait cycles and phases. The diagram 1 below explains how the system determines the split of the cycles.

In steps four, five and six the system links the timestamp of abnormalities to the videos whilst the data is sent for classification. Once this is complete the system compiles all of the results into the PDF Diagnosis Generator.

The fields from the PDF are displayed for the medical professionals to edit and alter to how they see fit. Once this is submitted it is passed to be validated in step eight where the user can select whether the diagnosis is correct and makes final changes to the document.

If the diagnosis is incorrect, the diagnosis system learns from this. So the system then sends the result to the self-learning module of the system. Which then generates the PDF, if the diagnosis is correct, the PDF is generated instantly. Once the diagnosis is complete it is sent to the EPIC database within the UMCG to have for future use and reference for the doctors and patients to have access to it. This then completes the process steps of the gait diagnosis system.

6 Technology Stack

The selection of the technology stack for the Gait Diagnosis System at the UMCG is crucial to achieving our goal of a rapid, reliable, and efficient diagnostic tool. Below is a detailed discussion of the chosen technologies and the rationale behind each choice.

6.1 Data Processing

We decided to use Python for the data processing of the gait diagnosis system. The rationale behind this is:

6.1.1 Robust Libraries

Python has a rich ecosystem of libraries and frameworks specifically geared towards data processing such as NumPy, pandas and ez3cd. These libraries offer robust functionality that simplifies handling and analyzing complex datasets such as the ones stored in 3cd files.

6.1.2 Ease of Use

Python's syntax and readability make it an accessible choice for us as it is easier to understand than most programming languages, and when looking at future work to be done on the project, makes it a good choice for future research done on gait diagnosis system. It also enables rapid development and prototyping making it a very suitable choice for us given the time frame.

6.1.3 Community and Support

Python has a vast and active community which ensures that we will have extensive support, regular updates and a multitude of resources and documentation for troubleshooting and debugging.

6.1.4 Machine Learning Integration

Python is the preferred language for machine learning and AI with powerful libraries such as TensorFlow, Keras, and scikit-learn. This is essential for incorporating AI into our data processing to enhance the accuracy and speed of gait analysis.

6.2 Backend Development

For the backend we decided to use Flask which is a backend framework in Python. The rationale behind this is:

6.2.1 Python Integration

Since our data processing is handled in Python, using Flask, a lightweight Python web framework, ensures seamless integration and reduces the overhead of context switching between different programming languages.

6.2.2 Flexibility & Simplicity

Flask is designed to be a simple and flexible framework. It provides the essentials without enforcing any particular project structure making it the ideal solution for the custom gait diagnosis system we are developing.

6.2.3 Extensibility

Flask supports extensions that can add functionality as needed such as database integration, form validation and authentication, allowing for a scalable and maintainable application.

6.2.4 Performance

Flask is lightweight and allows for fine-tuned control over the application, which is critical for ensuring the backend can handle real-time data processing demands efficiently.

6.3 Frontend Development

For the frontend we decided to use Reach which is a frontend framework in JavaScript. the rationale behind this is:

6.3.1 Component-Based Architecture

React's component-based architecture allows for the development of reusable UI components, ensuring consistency and maintainability across the application.

6.3.2 Performance

React efficiently updates and renders components as data changes, which is crucial for creating a responsive and interactive user interface for the gait diagnosis system.

6.3.3 Compatibility with Flask

React works well with Falsk, as the frontend and backend can communicate seamlessly via RESTful APIs. This compatibility ensures smooth data flow and interaction between the client-side and server-side components.

6.3.4 Developer Tools and Ecosystem

React's robust developer tools and active ecosystem provide a wide range of libraries, extensions and plugins allowing us to create a highly functional and user-friendly interface.

6.4 Integration of Technologies

The integration of these technologies forms a cohesive and efficient stack tailored to the specific requirements of our gait diagnosis system. Python's strength in data processing is leveraged over other programming languages to handle and analyze gait data effectively and efficiently. Flask is an excellent choice for the backbone of the application and provides a reliable and flexible environment for backend operations. React enhances the user experience by delivering a dynamic and responsive front end. Together, these technologies enable the development of a system that is both technically robust and user-centric, ultimately reducing wait times for patients and providing medical professionals with a powerful diagnostic tool which is the penultimate objective of this project.

7 Implementation

This section includes the implementation details of the system including the methods and approach used to develop the system as well as discussing the distribution amongst the members of the project.

7.1 Data Processing and Analysis

The initial stage of the process involved extracting diverse data essential for formulating the diagnosis.

7.1.1 Extracting Data from the `.c3d` Files (Function 2)

The UMCG provides the research project with `.c3d` files, a standard format widely used in bio-mechanics and motion capture to store 3D motion data. One of these `.c3d` files act as the 'Normal Data', representing an ideal gait cycle stored in the software for future comparisons. The second `.c3d` file contains the patient's data collected while walking on the platform.

To begin analyzing the data from these `.c3d` files, it is necessary to decompose them, as each file typically includes several sections: a Header section, a Parameter section, a 3D Point Data section, an Analog Data section, and an Event Data section. The function `readc3d` 2 is used to extract important data such as labels, knee, hip, and ankle angles, as well as details like frame-rate, starting and ending frames. These extracted parameters are essential for subsequent data analysis.

For handling the `c3d` files, the main library utilized is `ezc3d` which provides built-in functions for opening files and extracting information from them.

7.1.2 Extracting Data from the Physical Examination(LO) (Function 1)

Another stage in the diagnostic process involved parsing the data provided by UMCG, which includes the results of the physical examination conducted on patients before walking on the platform. This examination assesses the patient's ability to extend various parts of their legs, measured in degrees. The data was structured in a single column within a standardized `.xlsx` file. To handle this data effectively, a Python script was developed to dynamically allocate two arrays: one for body part headers and another for corresponding degrees of extension. This dynamic approach ensures flexibility, allowing doctors to examine without strictly adhering to a predefined format, as the software manages data collection dynamically.

7.1.3 Stance Calculation

A crucial aspect of the diagnostic process involved developing software capable of accurately identifying the gait phases in a patient's walk, as defined in Figure 1. These predefined

phases describe specific stages of the gait cycle and serve as benchmarks for comparing deviations between patient data and normal walking patterns. The four computed phases include Heel strike, Loading response, Mid-stance, and Terminal stance, collectively spanning one complete gait cycle for a patient. Each phase's occurrences were noted for both the left and right foot.

The `.c3d` file of the patient contains valuable data regarding the moments when the feet make contact with and lose contact from the ground. These events can be extracted using functions provided by the `ezc3d` library, facilitating precise calculation of the gait phases.

7.1.3.1 **Heel Strike**

Heel strike, also known as initial contact, marks the moment when the heel first touches the ground at the onset of the stance phase in the gait cycle. This phase is critical as it initiates weight-bearing and sets the foot's alignment for subsequent movements. The timing of this event is easily determined by extracting data from the `.c3d` file.

7.1.3.2 **Loading Response** (Function 12)

Following heel strike, the loading response extends until the opposite foot lifts off the ground. During this phase, the body absorbs the impact of initial contact and begins transferring weight to the leading leg. Knee flexion helps in shock absorption, while foot pronation adapts to the ground surface. To identify this phase, we defined it as occurring ten frames before the opposite foot loses contact with the ground.

7.1.4 **Mid-Stance** (Function 10)

Mid-stance occurs when the body's weight is directly over the supporting foot, crucial for balance and stability. The foot serves as a stable platform, supporting the body weight efficiently. Proper alignment and muscle coordination during mid-stance facilitate energy transfer and prepare for propulsion. The `findMidStance` 10 function uses motion capture data to estimate this phase by analyzing ankle and knee coordinates.

Mid-stance is calculated through the following method:

$$\begin{aligned}
\text{Distance 1} &: d_1 = \text{Left Knee} - \text{Left Ankle} \quad \text{(for Right Foot)} \\
&: d_1 = \text{Right Knee} - \text{Right Ankle} \quad \text{(for Left Foot)} \\
\text{Distance 2} &: d_2 = \text{Right Knee} - \text{Left Knee} \\
\text{Mid-Stance} &: \text{Establish mid-stance where } |d_1 - d_2| \text{ is minimized.}
\end{aligned}$$

It identifies foot strike and foot-off events for both feet, converting these events into frame times based on sampling frequency.

7.1.3.4 **Terminal Stance** (Function 11)

Terminal stance begins as the supporting foot's heel starts to lift off the ground and ends when the opposite foot contacts the ground. During this phase, the body shifts forward over the stance leg, and the ankle, knee, and hip extend to propel the body forward. Similar to the loading response, terminal stance is estimated to occur ten frames before the opposite foot strikes the ground.

7.1.5 **Data Comparison** (Functions 5, 6)

Gait abnormalities were identified by analyzing hip, ankle, and knee angles during different phases of the gait cycle. For each phase, heel-strike, loading response, mid-stance, and terminal stance the system computes joint angles and compares them against normal ranges. Therefore, rather than reaching a single diagnostic conclusion, the system provides a comprehensive specification of all detected diagnoses for each leg per gait cycle as each part of the cycle is compared to normative data. Significant impairments are flagged by deviations of eight degrees or more, in alignment with clinical standards and the practices of other hospitals, to accurately identify potential gait abnormalities. This systematic approach focused on critical joint movements ensured thorough evaluation and diagnosis of gait impairments.

7.1.6 **Diagnosis** (Functions 7, 9, 8)

The software integrated data from physical examinations, detailing movement restrictions and angles of free extension for each body part, into the diagnosis process. Angles recorded during the examination are correlated with software-detected abnormalities. By linking clinical findings with motion capture data, the system offers a comprehensive diagnostic view. This holistic approach enhanced diagnostic accuracy and reliability by combining insights from both physical examination results and motion analysis. Integration was achieved programmatically by extracting and dynamically storing examination data, and comparing it with motion capture results based on specific body part labels and motion types (flexion or extension).

7.2 Back-end Development

Flask, a back-end framework in Python, is used to build the back-end. It comprises several endpoints crucial for data management, diagnostic result delivery, and authentication.

7.2.1 File upload (Endpoints 3 & 4)

File upload functionality, protected by authentication, allows uploading of `.c3d` and `.xlsx` files. The endpoint for `.c3d` files ensures uploads are saved in `uploads/c3d`, validating file type to accept only `.c3d` files. Similarly, the `.xlsx` upload endpoint saves files in `uploads/xlsx` and restricts uploads to `.xlsx` format. Since this endpoint is of the `HTTP` Request `POST` it sends back the uploaded files as separate `JSON` Objects.

7.2.2 Authentication (Endpoints 1 & 2)

User authentication is managed through the `/login` and `/logout` endpoints, which handle user login sessions securely. Authentication credentials are validated against an in-memory user store.

7.2.3 CORS and Security

Cross-Origin Resource Sharing (CORS) is configured to allow secure communication with the front-end. Additional security measures are implemented to protect endpoints and manage user sessions effectively.

7.2.4 Error Handling

Error handling in the back-end of the Gait Diagnosis System is crucial to ensure robustness and reliability when handling various operations, such as file uploads and data processing. The implementation uses try-except blocks to catch and manage exceptions that may occur during runtime and are displayed in the logs of the back-end if there are any.

7.2.5 Diagnosis (Endpoint 5)

Upon receiving the `.xlsx` file, the back-end saves it and triggers the process function for data analysis based on the `.c3d` file and `.xlsx`. This function utilizes data from physical examination spreadsheets and `.c3d` files, processing them via the `dataProcessing` component to generate diagnostic results returned to the front-end in `JSON` format.

7.3 Front-end Development

The front-end connects to the back-end to be able to allow the user to log in, and upload a `.c3d` file and a `.xlsx` file and receive a diagnoses response from the back-end and data processing module displayed in a table for readability.

7.3.1 Component-Based Architecture

React is used as the front-end framework making use of a component-based architecture to allow for scalability, re-usability and ease of maintenance. Scalability is a priority in this system because developing a well-designed architecture and codebase not only ensures the system can handle increased loads but also makes it easier for others to build upon and continue the work that has been completed.

7.3.2 Authentication

The user is prompted to a login page that only medical professionals will have access to. Once logged in the Authentication is checked to see if login was successful from the back-end endpoint. If it was then the user is navigated to the dashboard where they are prompted to upload a `.c3d` file and `.xlsx` file from the physical examination. A logout button is also provided to allow users to securely exit the application. Clicking the button triggers a request to the back-end to terminate the user session and redirects them to the login page. This ensures proper session management and only allows access to authorized personnel only. The login page is represented by the following figure:

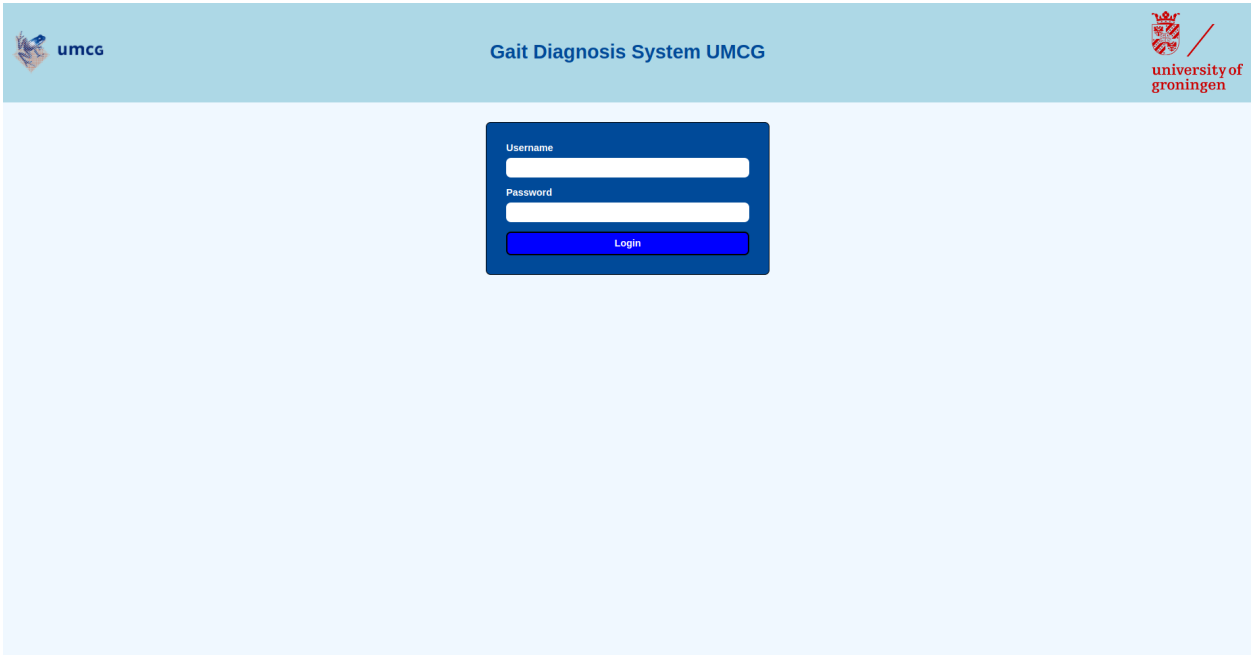

Figure 10: Login Page

7.3.3 **Dashboard**

The dashboard is the main page of the project and front-end. After login is successful the users are directed to the dashboard. Aforementioned there are buttons to upload a `.c3d` file and `.xlsx` file from the physical examination. These buttons display an input dialogue for the users to select files from their local system. After that, the files are uploaded to the back-end via the designated endpoints. The front-end provides real-time feedback, showing success messages for successful uploads and error messages for any upload issues. Once the upload is successful, the files are sent to the data-processing component, a diagnosis is created and sent to the back-end which then sends the results of the gait analysis to be displayed on the front-end. The front-end parses this data and presents it in a user-friendly format in the form of a table, making it easy for medical professionals to interpret the results. This is done using dynamic rendering based on `JSON` Objects returned from the endpoint. Below is a figure displaying the Dashboard page:

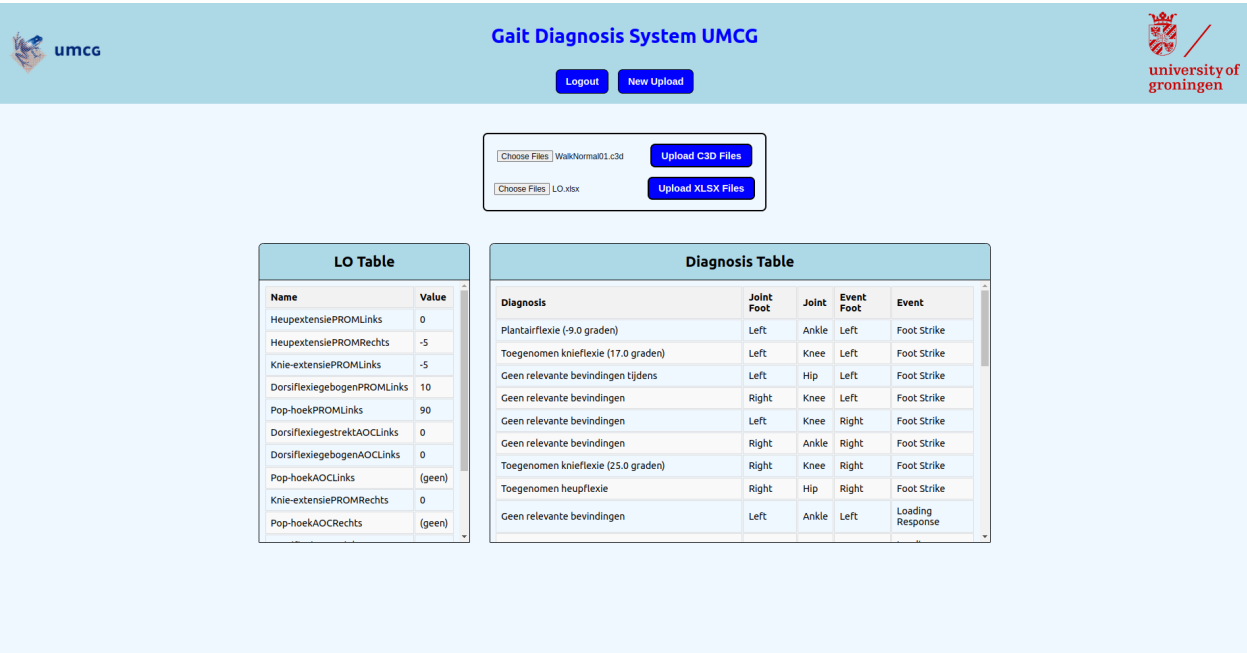

Figure 11: Dashboard Page

7.4 Individual Contribution

For clarity, my contributions primarily involved implementing the front end and developing the file-upload endpoints 3 & 4 and the diagnosis endpoint 5 on the back-end. The front-end was completed entirely by me, developing the authentication component, file uploader and diagnosis tables component. The back-end work was a collaborative effort between Emmanouil and myself. I concentrated on establishing a smooth integration between the front-end and back-end, ensuring that the file upload and diagnosis endpoints operated seamlessly. Meanwhile, Amr and Emmanouil were responsible for the data processing components. Our group made a concerted effort to evenly distribute the workload to avoid any disparity. The architecture diagram below illustrates my contributions, highlighted within the red boxes:

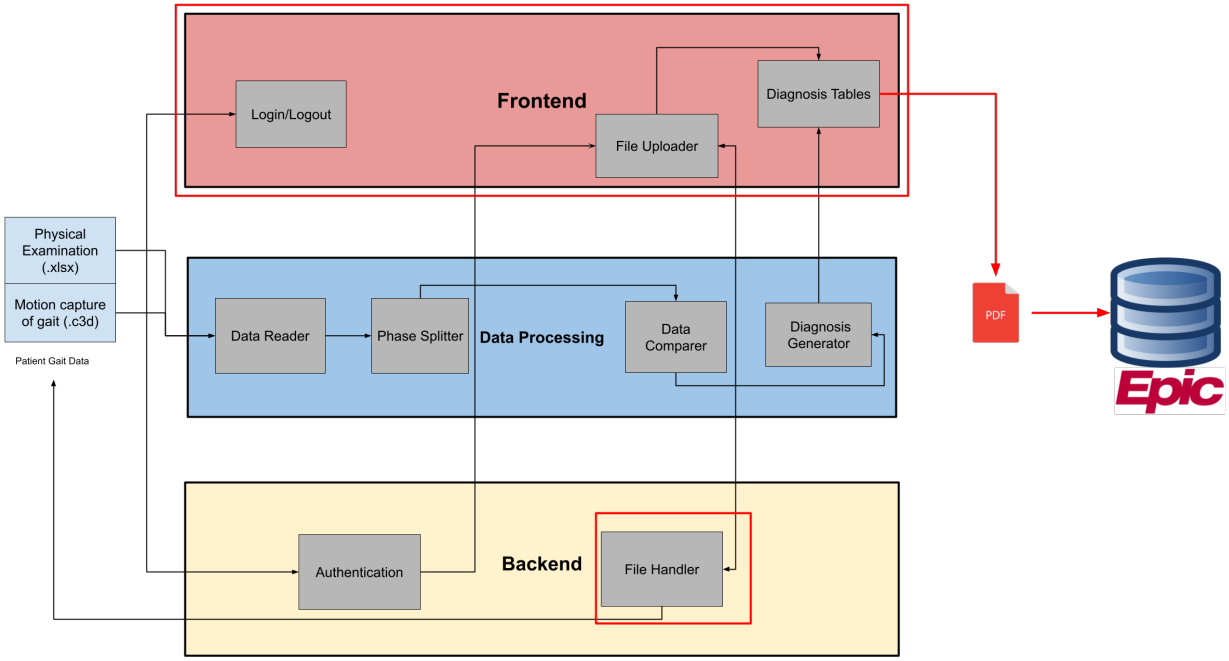

Figure 12: Individual Contribution Diagram

8 Results and Discussion

This section entails which requirements were fulfilled as well as aspects that were not able to be completed.

8.1 Results

A prototype of a gait diagnosis system was successfully developed that contains an architecture incorporating a front-end, back-end, and data-processing model utilizing industry standards and well-documented, reliable technologies. Additionally, a user interface for visualization of the system is designed for easy usage by medical professionals 11. Furthermore, the automated diagnosis feature is integrated into the system providing rapid and accurate diagnoses. Lastly, documentation on the architecture through diagrams 8, 9 and code documentation in the code base. In relation to the set requirements this means that five out of 6 were achieved and can be represented in the following table:

Requirement	Description	Fulfilled
[R-1-F]	The system should allow users to upload a .c3d file and a physical examination file (.xlsx).	✓
[R-2-F]	The system should display the diagnosis for the respective patient	✓
[R-3-F]	The user should securely be able to login and logout using the appropriate credentials	✓
[R-4-F]	The system should produce a PDF representing the diagnosis report that the user could download	✗
[R-5-NF]	The system should be easy to use and understand for medical professionals	✓
[R-6-NF]	The system should display a diagnosis under 5 seconds after successful upload	✓

8.1.1 Prototype of the System

The development of the system followed industry standards for creating the architecture and developing a prototype for medical professionals to use. The smooth integration of the front-end and back-end with quick responsiveness is evident. This is evident through using compatible technologies, modular architecture and RESTful endpoints. Alongside this, the data-processing module integrated well with the Flask back-end since both were in Python. This common language facilitated seamless communication and data handling between components, contributing to the reliability and quality of the gait diagnosis system.

8.1.2 User Interface

The user interface is clear and minimalistic in design to allow for ease of use by medical professionals. This can be shown by the multiple frames of the front-end displaying the user experience through Figures 10 and 11.

8.1.3 Automation of Gait Diagnosis 3

As mentioned previously, the vital requirement for the system is automating the diagnosis. Alongside this, the process dramatically reduces the time required to produce a diagnosis, cutting it down from 60-120 minutes to under five seconds. Through comparison to normative data, the accuracy and precision of the diagnosis are of a high quality and standard, in comparison to existing work, as each part of the cycle is compared. This allows for the utmost accuracy of the system as well as being able to produce a diagnosis rapidly. The ability to specify multiple diagnoses for each leg in a single gait cycle provides a nuanced and comprehensive understanding of the patient's gait impairments. This is particularly helpful in clinical practice, as patients often can have impairments in only one leg or specific parts of that leg. Consequently, the software allows for a precise identification of gait impairments, focusing on the specific body part affected.

8.1.4 Comprehensive Documentation

Documentation is vital for encapsulating this system. The quality and clarity are crucial to allow others to contribute to it after this thesis. An architecture diagram 8 and a flow diagram 9 concretely explain each step of the process and the extensive comments in the code C help others quickly understand the system we produced and can continue our work for the benefit of medical professionals and the UMCG.

8.2 Discussion

8.2.1 Comparison with Previous Work

Compared to previous work completed in UMCG regarding the gait diagnosis system, the system that was produced shows substantial improvements in multiple aspects. Firstly, the use of industry-standard architecture and a modularized design marks a notable enhancement over the previous code, which lacked both. This improvement enables other researchers to expand on our system and helps the UMCG continue providing automated gait diagnoses.

Another area of major improvement is providing a User Interface (UI). The interactive and responsive UI, due to using React, along with straightforward and simplistic design 11 allows for medical professionals ease of use and quick adaptability to produce various diagnosis results.

Furthermore, one aspect that the previous code did not address is linking the motion data to the physical examination, enabling us to give multiple diagnoses for each leg alongside

having each part of the cycle compared to normative data. This adds a level of accuracy, in comparison to the work produced previously, as it allows doctors to be able to see a more detailed representation of the abnormality at each joint, foot and gait phase. This allows medical professionals to have a more accurate understanding of the difficulty patients can have, leading to doctors being able to provide more effective treatment planning.

Subsequently, the system produced through this bachelor's thesis provides a user-friendly UI, reliable diagnoses and a scalable and extensible codebase for future work allowing for the automation of the gait diagnosis system to be utilized by the UMCG leading to a decrease in wait-time for patients, increase in speed of doctor's diagnoses and treatment plans, therefore, causing an increase in patient throughput.

8.2.2 **Limitations**

Although the system was of great standard and quality, there were limitations due to time constraints mainly. The main limitation was not implementing the self-improving AI component in the diagnostic process. Despite this feature being intended originally, it was not finished within the allotted time. Consequently, the user of the system is not able to edit the diagnosis displayed since there is no self-diagnoser. Incorporating AI-driven self-improvement mechanisms could enable the system to learn from each diagnosis, thereby continuously enhancing its accuracy and minimizing the occurrence of false positives and negatives. This can be best represented through a diagram 13 depicting the ideal architecture to be utilized for future projects including all previous modules and components from Figure 8 but including a form editor for a doctor's input and a self-learning component in the back-end to incorporate with new diagnoses. This diagram is represented below:

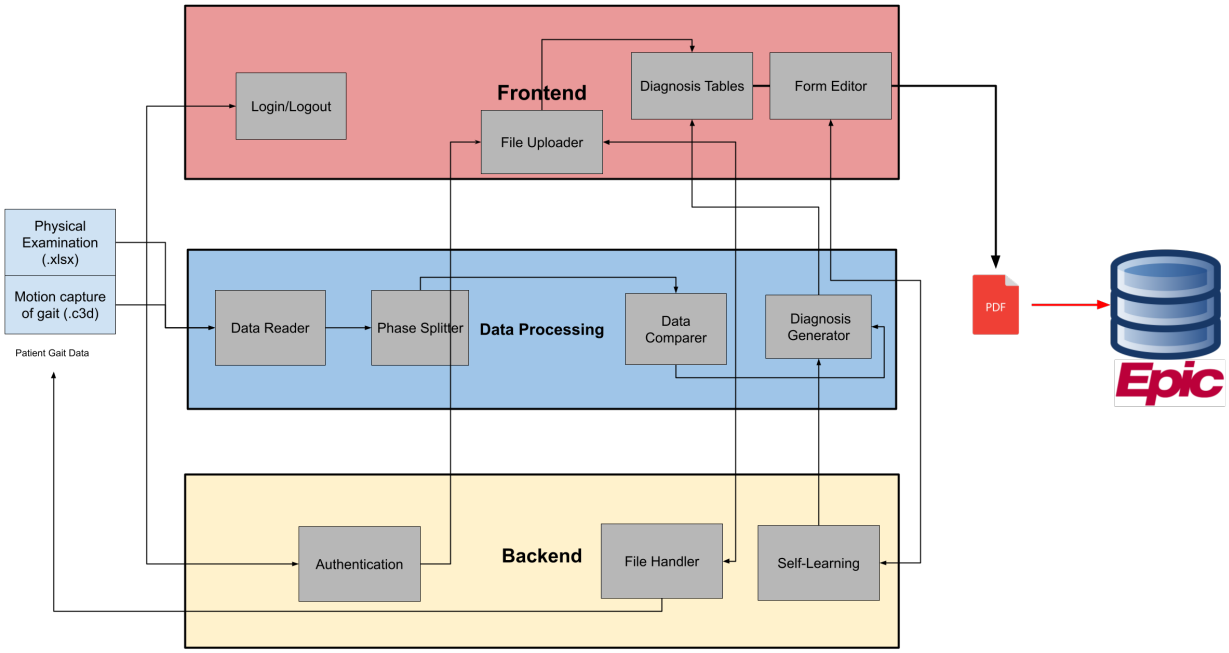

Figure 13: The ideal system architecture diagram incorporating new components.

Another aspect that was not implemented was not generating a PDF based on the diagnosis which led to not integrating with the EPIC database part of the UMCG. This was mainly due to wanting to run tests through the UMCG first before allowing the results of the diagnosis system to be saved on the UMCG database.

9 Conclusion

Gait disorders and abnormalities significantly affect individuals across various age groups, with a considerable prevalence among the elderly and those with specific medical conditions. Diagnosing these gait issues promptly is crucial for patient well-being. Currently, the gait diagnosis process at the University Medical Center Groningen (UMCG) is inefficient, characterized by long waiting times and a low patient throughput. This inefficiency is largely due to the manual and laborious nature of the current system. Motivated by the need to enhance the efficiency and reliability of gait diagnosis, this thesis aimed to develop a prototype gait diagnosis tool that restructures the existing system architecture and automates processes to improve speed and accuracy.

Going back to the research question, **"How can the architecture and implementation of a gait diagnosis system be designed for increased efficiency, incorporating industry-standard design principles and architectural patterns?"**, the system was structured using industry-standard design principles and well-documented technologies for both front-end and back-end. This design resulted in a more efficient and reliable system, capable of handling the complexities of gait analysis by providing multiple diagnoses for different gait phases and producing results in under 5 seconds.

The development of the Gait Diagnosis System for UMCG significantly improved the existing system by addressing its limitations. The project created a user-friendly front-end interface that simplifies interactions for medical professionals, offering real-time feedback and accessible diagnostic results. Extensive documentation was produced, detailing the system architecture, implementation, and usage instructions, which will aid future research and development. Moreover, the automation of data processing and diagnosis streamlined the workflow, reducing the time required for data analysis, thereby decreasing patient wait times and enhancing efficiency.

In conclusion, this project has laid a strong foundation for the development of a gait diagnosis system at UMCG. The improvements in system architecture, data processing, and user interface design have demonstrated the potential for significant advancements in the field of gait analysis. By addressing current limitations and exploring future directions, the Gait Diagnosis System can continue to evolve, providing even greater benefits to patients and medical professionals alike. This thesis not only contributes to the existing body of knowledge but also sets the stage for ongoing innovation and development in gait diagnosis technology.

10 Future Work

Future iterations of the system should address the limitations mentioned previously.

10.1 Integrating Self-Learning AI Model

One area for future work is the implementation of machine learning models used for gait analysis. While this project has laid the groundwork for integrating AI into the diagnosis process, there is potential to improve the accuracy and reliability of these models. Future research should focus on the self-learning component to be able to better deal with data that are outliers (extreme cases) as well as considering doctors' inputs which is the most important since they are the most knowledgeable about gait diagnosis in the field. This would ultimately lead to better clinical outcomes in terms of an accurate gait diagnosis.

10.2 User Interface

Additionally, linking to the above sub-section the front-end should allow the user to edit fields of the displayed diagnosis to train the AI model to produce better and more precise diagnoses. This can be done by providing an input field to adjust the threshold value 5 which is the normative value that the data is compared to. Since other hospitals or other medical professionals might use another value that they see fit. Furthermore, the UI can provide fields which would edit the `JSON` object and then re-render the table based on the changes.

10.3 Uploading diagnosis to EPIC (UMCG Database)

Integration with the EPIC database was not achieved as the system would still need to be thoroughly tested before the diagnoses are uploaded to the database to send to clients after consultation. This is important for future researchers to keep in mind as connecting to EPIC should be relatively straightforward due to the component-based design and overall architecture of the system.

10.4 Testing

One crucial aspect of any diagnostic system is testing. In the context of this thesis, testing should be conducted in collaboration with medical professionals to ensure reliability and accuracy. Doctors should independently diagnose patient gait issues and compare these to the system's automated diagnoses, aiming for over 90% accuracy to consider the system reliable. An iterative testing process should be implemented, gathering feedback from medical professionals to continuously refine and improve the AI model. Performance should be evaluated across different gait phases and conditions to ensure consistent accuracy and reliability. Validation studies with a diverse patient population will help ensure the system's diagnoses are generalized across various demographics and gait disorders.

References

[1] Birol Balaban and Fatih Tok. Gait disturbances in patients with stroke. *PM R: The Journal of Injury, Function, and Rehabilitation*, 6(7):635–642, 2014. Epub 2014 Jan 19.

[2] Thomas Brinkhoff. City population - groningen, netherlands. https://www.citypopulation.de/en/netherlands/admin/, 2023. Data collected with the Central Bureau for Statistics in the Netherlands.

[3] Gema Chamorro-Moriana, Antonio José Moreno, and José Luis Sevillano. Technology-based feedback and its efficacy in improving gait parameters in patients with abnormal gait: A systematic review. *Sensors*, 18(1), 2018.

[4] Tianrui Cui, Le Yang, Xiaolin Han, Jiandong Xu, Yi Yang, and Tianling Ren. A low-cost, portable, and wireless in-shoe system based on a flexible porous graphene pressure sensor. *Materials*, 14(21), 2021.

[5] Yitzchak Frank, Tess Levy, Reymundo Lozano, Kate Friedman, Slayton Underwood, Ana Kostic, Hannah Walker, and Alexander Kolevzon. Gait abnormalities in children with phelan-mcdermid syndrome. *Journal of Child Neurology*, 38(13-14):665–671, 2023. PMID: 37849292.

[6] Roberto Grujičić and Dimitrios Mytilinaios. Gait cycle. *Kenhub*, November 2023. Last reviewed: November 28, 2023.

[7] Ming-Yang Ho, Ming-Che Kuo, Ciao-Sin Chen, Ruey-Meei Wu, Ching-Chi Chuang, Chi-Sheng Shih, and Yufeng Jane Tseng. Pathological gait analysis with an open-source cloud-enabled platform empowered by semi-supervised learning-pathoopengait. *IEEE journal of biomedical and health informatics*, 28(2):1066–1077, 2024.

[8] Yohanes Hutabarat, Daisuke Owaki, and Mitsuhiro Hayashibe. Quantitative gait assessment with feature-rich diversity using two imu sensors. *IEEE Transactions on Medical Robotics and Bionics*, 2(4), 2020.

[9] M.T. Karimi, R.B. Tahmasebi, B. Satvati, and F. Fatoye. Influence of foot insole on the gait performance in subjects with flat foot disorder. *Journal of Mechanics in Medicine and Biology*, 19(6):1950050, 2019.

[10] Min-Jung Kim, Ji-Hun Han, Woo-Chul Shin, and Youn-Sik Hong. Gait pattern identification using gait features. *Electronics*, 13(10), 2024.

[11] Zihan Li, Jun Liu, Xinxin Miao, Shaoyun Ge, Jun Shen, Shaohua Jin, Zhengxue Gu, Yongfeng Jia, Kezhong Zhang, Jianwei Wang, and Min Wang. Reorganization of structural brain networks in parkinson's disease with postural instability/gait difficulty. *Neuroscience Letters*, 827:137736, 2024.

[12] Adrian Segura Lorente. Gait analysis tool: Improvement and optimization. RUG Honor's Project, 2022. Supervisors: Dr. Dimka Karastoyanova and Dr. Christian Greve.

[13] Merck Manual. Gait disorders in older adults, 2023. Accessed 2/22/2023.

[14] S. Pesenti, L. Garcia, V. Pomero, G. Authier, and C. Boulay. Gait abnormalities in adolescent idiopathic scoliosis patients: Do curvature amplitude matter? *Gait & Posture*, 97:399, 2022.

[15] Motion Lab Systems. C3d file format. http://www.c3d.org, 2023. Accessed: 2023-01-12.

[16] Marjolein M. van der Krogt, Han Houdijk, Koen Wishaupt, Kim van Hutten, Sarah Dekker, and Annemieke I. Buizer. Development of a core set of gait features and their potential underlying impairments to assist gait data interpretation in children with cerebral palsy. *Frontiers in Human Neuroscience*, 16, 2022.

[17] Marjolein M. van der Krogt, Han Houdijk, Koen Wishaupt, Kim van Hutten, Sarah Dekker, and Annemieke I. Buizer. Development of a core set of gait features and their potential underlying impairments to assist gait data interpretation in children with cerebral palsy. *Frontiers in Human Neuroscience*, 16, 2022.

[18] E. West and U. Shah. Diagnosis of functional weakness and functional gait disorders in children and adolescents. *Seminars in Pediatric Neurology*, 41, 2022.

[19] Miaomiao Zhao, Hui Xu, Weibing Zhong, Xiaojuan Ming, Mufang Li, Xinrong Hu, Kangyu Jia, and Dong Wang. Robust and breathable all-textile gait analysis platform based on lenet convolutional neural networks and embroidery technique. *Sensors and Actuators: A. Physical*, 360, 2023.

A Link to Project Gait Diagnosis Repository

https://github.com/akr115/RUG-Gait-Diagnosis-System

B Backend Endpoints

```
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')
    if username in users and users[username] == password:
        session['username'] = username
        return jsonify({"success": True, "message": "Login successful"}), 200
    else:
        return jsonify({"success": False, "message": "Invalid username or password"}), 401
```

Listing 1: Login endpoint

```
@app.route('/logout')
def logout():
    session.pop('username', None)
    flash('You have been logged out', 'info')
    return redirect(url_for('index'))
```

Listing 2: Logout endpoint

```python
@app.route('/upload/c3d', methods=['POST'])
def upload_c3d_files():
    if 'files' not in request.files:
        return jsonify({"error": "No file part"}), 400

    files = request.files.getlist('files')
    if not files:
        return jsonify({"error": "No selected files"}), 400

    try:
        filenames = []
        for file in files:
            if file and file.filename.endswith('.c3d'):
                filename = secure_filename(file.filename)
                file.save(os.path.join(UPLOAD_FOLDER_C3D, filename))
                filenames.append(filename)
        return jsonify({"message": "C3D files uploaded successfully!", "
    filenames": filenames}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

Listing 3: Upload c3d file endpoint

```python
@app.route('/upload/xlsx', methods=['POST'])
def upload_xlsx_files():
    if 'files' not in request.files:
        return jsonify({"error": "No file part"}), 400

    files = request.files.getlist('files')
    if not files:
        return jsonify({"error": "No selected files"}), 400

    try:
        filenames = []
        for file in files:
            if file and file.filename.endswith('.xlsx'):
                filename = secure_filename(file.filename)
                file.save(os.path.join(UPLOAD_FOLDER_XLSX, filename))
                filenames.append(filename)
        return jsonify({"message": "XLSX files uploaded successfully!", "
    filenames": filenames}), 200
    except Exception as e:
        print("yo")
        return jsonify({"error": str(e)}), 500
```

Listing 4: Upload xlsx file endpoint

```python
@app.route('/diagnose', methods=['POST'])
def diagnose_endpoint():
    if 'file' not in request.files:
        return jsonify({"error": "No file part"}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({"error": "No selected file"}), 400

    if file and file.filename.endswith('.xlsx'):
        filename = secure_filename(file.filename)
        filepath = os.path.join(UPLOAD_FOLDER_XLSX, filename)
        file.save(filepath)
        results, lo = process()
        results = results.to_json()
        lo = lo.to_json()
        return jsonify(results, lo), 200
    else:
        return jsonify({"error": "Invalid file type"}), 400
```

Listing 5: Diagnose endpoint

C Data Processing Functions

Algorithm 1 Read XLSX File

Require: File path to XLSX file
Ensure: DataFrame containing LO evaluation
  Input: `file_path`
  Output: DataFrame `df`
  function READXLSX(file_path)
    Read XLSX file into `df`
    Initialize `variable_names` and `variable_values`
    for each set of 3 rows in `df` **do**
      if name cell is not empty **then**
        Append name to `variable_names`
        Append value to `variable_values`
      end if
    end for
    Create DataFrame `df` with `variable_names` and `variable_values`
    Remove whitespace from `df`
    return `df`
  end function

Algorithm 2 Read C3D File

Require: File path to C3D file
Ensure: DataFrame containing events of gait cycles and joint angles
  Input: `file_path`
  Output: `globalEvents, LAngles, RAngles, firstFrame, frameRate`
  function READC3D(file_path)
    Read C3D file into `c3d_file`
    Extract labels and data points from `c3d_file`
    Initialize `data` dictionary with labels
    for each indicator and frame **do**
      for each dimension **do**
        Append data to `data` dictionary
      end for
    end for
    Create DataFrame `df` from `data`
    Extract joint angles for left and right legs into `LAngles` and `RAngles`
    Create DataFrame `globalEvents` with contexts, labels, and times
    Extract midstance, terminal stance, and loading response using helper functions
    Create DataFrames for each event
    Concatenate event DataFrames into `globalEvents`
    return `globalEvents, LAngles, RAngles, firstFrame, frameRate`
  end function

Algorithm 3 Trim Global Events

Require: DataFrame with global events
Ensure: Trimmed DataFrame with necessary events
  Input: `globalEvents`
  Output: Trimmed DataFrame `filtered_globalEvents`
  function TRIMGLOBALS(globalEvents)
    Initialize counters for events
    for each row in `globalEvents` **do**
      Increment event counters based on event label
      if at least 2 gait cycles for each event **then**
        Save index of the last event
        Break loop
      end if
    end for
    Slice `globalEvents` up to the last event index
    return `filtered_globalEvents`
  end function

Algorithm 4 Calculate Indices

Require: Event data and frame information
Ensure: Indices for normal and current events
Input: `event, event_normal, frame_rate_normal, frame_rate, first_frame_normal, first_frame`
Output: `index_frame_normal, index_frame`
function CALCULATEINDICES(event, event_normal, frame_rate_normal, frame_rate, first_frame_normal, first_frame)
`index_frame_normal` ← `frame_rate_normal` × `event_normal['times']` − `first_frame_normal`
`index_frame` ← `frame_rate` × `event['times']` − `first_frame`
return `index_frame_normal, index_frame`
end function

Algorithm 5 Calculate Differences

Require: Joint angles and thresholds
Ensure: Differences and angles between current and normal joint angles
Input: `ankleAngles, kneeAngles, hipAngles, ankleAnglesNormal, kneeAnglesNormal, hipAnglesNormal, frameNormal, frame, threshold`
Output: `result, angles`
function CALCULATEDIFFERENCES(ankleAngles, kneeAngles, hipAngles, ankleAnglesNormal, kneeAnglesNormal, hipAnglesNormal, frameNormal, frame, threshold)
Initialize `diff_array` and `result`
Extract `angles` from current angles
Append differences to `diff_array`
for each difference in `diff_array` **do**
if difference $\leq -threshold$ **then**
Append -1 to `result`
else if difference $\geq threshold$ **then**
Append 1 to `result`
else
Append 0 to `result`
end if
end for
return `result, angles`
end function

Algorithm 6 Compare Joint Angles

Require: Global events and joint angles
Ensure: DataFrame with comparison results
Input: `global_events, global_events_normal, LAngles, RAngles, LAnglesNormal, RAnglesNormal, first_frame_normal, first_frame, frame_rate_normal, frame_rate, threshold`
Output: DataFrame `df`
function COMPAREJOINTANGLES(global_events, global_events_normal, LAngles, RAngles, LAnglesNormal, RAnglesNormal, first_frame_normal, first_frame, frame_rate_normal, frame_rate, threshold)
Initialize `data_labels` and `data`
Extract joint angles for left and right legs from `LAngles` and `RAngles`
for each `global_event` in `global_events` **do**
Find corresponding `global_event_normal`
`index_frame_normal, index_frame` ← CALCULATEINDICES(global_event, global_event_normal, frame_rate_normal, frame_rate, first_frame_normal, first_frame)
`result_l, angles_l` ← CALCULATEDIFFERENCES(LAnkleAngles, LKneeAngles, LHipAngles, LAnkleAnglesNormal, LKneeAnglesNormal, LHipAnglesNormal, index_frame_normal, index_frame, threshold)
`result_r, angles_r` ← CALCULATEDIFFERENCES(RAnkleAngles, RKneeAngles, RHipAngles, RAnkleAnglesNormal, RKneeAnglesNormal, RHipAnglesNormal, index_frame_normal, index_frame, threshold)
Append results to `data`
end for
Convert `data` into DataFrame `df`
return `df`
end function

Algorithm 7 Diagnose Ankle

Require: Data and side of the foot
Ensure: Diagnosis results and list of relevant variables
  Input: `data, side`
  Output: DataFrame of results, list of relevant variables
  function DIAGNOSE_ANKLE(data, side)
    Filter `data` for stance based on `side`
    Initialize `results` and `lo_variables` as empty lists
    for each row in `df_stance` **do**
      if row.Event == 'Foot Strike' **then**
        if ankle angle is -1 **then**
          Append diagnosis for plantarflexion to `results`
          Extend `lo_variables` based on `side`
        else
          Append no relevant finding to `results`
        end if
      end if
    end for
    return DataFrame of `results`, `lo_variables`
  end function

Algorithm 8 Diagnose Knee

Require: Data and side of the body
Ensure: Diagnosis results and list of relevant variables
  Input: `data, side`
  Output: DataFrame of results, list of relevant variables
  function DIAGNOSE_KNEE(data, side)
    Filter `data` for stance based on `side`
    Initialize `results` and `lo_variables` as empty lists
    for each row in `df_stance` **do**
      if row.Event == 'Foot Strike' **then**
        if knee condition **then**
          Append diagnosis for knee condition to `results`
          Extend `lo_variables` based on `side`
        else
          Append no relevant finding to `results`
        end if
      end if
    end for
    return DataFrame of `results`, `lo_variables`
  end function

Algorithm 9 Diagnose Hip

Require: Data and side of the body
Ensure: Diagnosis results and list of relevant variables
  Input: `data, side`
  Output: DataFrame of results, list of relevant variables
  function DIAGNOSE_HIP(data, side)
    Filter `data` for stance based on `side`
    Initialize `results` and `lo_variables` as empty lists
    for each row in `df_stance` **do**
      if row.Event in ['Foot Strike', 'Loading Response'] **then**
        if hip condition **then**
          Append diagnosis for hip condition to `results`
          Extend `lo_variables` based on `side`
        else
          Append no relevant finding to `results`
        end if
      end if
    end for
    return DataFrame of `results`, `lo_variables`
  end function

Algorithm 10 Find Mid Stance

Require: Marker positions, sampling frequency, times, contexts, events
Ensure: Midstance frames for right and left legs
Input: `markersWithLabels, Fs_MarkerPositions, times, contexts, events`
Output: `midStance_right, midStance_left`
function FINDMIDSTANCE(markersWithLabels, Fs_MarkerPositions, times, contexts, events)
Initialize `evnt_ic_right, evnt_ic_left` to `1e10`
Initialize `evnt_footoff_right, evnt_footoff_left` to `-1`
Extract marker positions for left ankle (`LANK`), left knee (`LKNE`), right ankle (`RANK`), right knee (`RKNE`)
for each event in `events` **do**
if event is 'Foot Strike' and context is 'Right' **then**
`evnt_ic_right` ← `times[i]`
else if event is 'Foot Off' and context is 'Right' and `times[i] > evnt_ic_right` **then**
`evnt_footoff_right` ← `times[i]`
else if event is 'Foot Strike' and context is 'Left' **then**
`evnt_ic_left` ← `times[i]`
else if event is 'Foot Off' and context is 'Left' and `times[i] > evnt_ic_left` **then**
`evnt_footoff_left` ← `times[i]`
end if
end for
Convert times to frames using `Fs_MarkerPositions`
Calculate `d1_d2_right` as the absolute difference between distances `d1` and `d2`
`midStance_right` ← frame with minimum `d1_d2_right`
Calculate `d1_d2_left` similarly
`midStance_left` ← frame with minimum `d1_d2_left`
return `midStance_right, midStance_left`
end function

Algorithm 11 Find Terminal Stance

Require: Sampling frequency, times, contexts, events
Ensure: Terminal stance frames for right and left legs
  Input: `Fs_MarkerPositions, times, contexts, events`
  Output: `evnt_terminalStance_right, evnt_terminalStance_left`
  function FINDTERMINALSTANCE(Fs_MarkerPositions, times, contexts, events)
    Initialize lists `evnt_footstrike_right, evnt_footstrike_left`
    Initialize lists `evnt_terminalStance_right, evnt_terminalStance_left`
    Set `frame_estimation` to 10
    for each event in `events` **do**
      if event is 'Foot Strike' and context is 'Right' **then**
        Append `times[i]` to `evnt_footstrike_right`
      else if event is 'Foot Strike' and context is 'Left' **then**
        Append `times[i]` to `evnt_footstrike_left`
      end if
    end for
    for each footstrike in `evnt_footstrike_right` **do**
      Append footstrike time minus `frame_estimation / Fs_MarkerPositions` to `evnt_terminalStance_left`
    end for
    for each footstrike in `evnt_footstrike_left` **do**
      Append footstrike time minus `frame_estimation / Fs_MarkerPositions` to `evnt_terminalStance_right`
    end for
    return `evnt_terminalStance_right, evnt_terminalStance_left`
  end function

Algorithm 12 Find Loading Response

Require: Sampling frequency, times, contexts, events
Ensure: Loading response frames for right and left legs
  Input: `Fs_MarkerPositions, times, contexts, events`
  Output: `evnt_loading_resp_right, evnt_loading_resp_left`
  function FINDLOADINGRESPONSE(Fs_MarkerPositions, times, contexts, events)
    Initialize lists `evnt_footoff_right, evnt_footoff_left`
    Initialize lists `evnt_loading_resp_right, evnt_loading_resp_left`
    Set `frame_estimation` to 10
    for each event in `events` **do**
      if event is 'Foot Off' and context is 'Right' **then**
        Append `times[i]` to `evnt_footoff_right`
      else if event is 'Foot Off' and context is 'Left' **then**
        Append `times[i]` to `evnt_footoff_left`
      end if
    end for
    for each footoff in `evnt_footoff_right` **do**
      Append footoff time minus `frame_estimation / Fs_MarkerPositions` to `evnt_loading_resp_left`
    end for
    for each footoff in `evnt_footoff_left` **do**
      Append footoff time minus `frame_estimation / Fs_MarkerPositions` to `evnt_loading_resp_right`
    end for
    return `evnt_loading_resp_right, evnt_loading_resp_left`
  end function