HALMSTAD UNIVERSITY

Bachelor of science in computer engineering, 180 credits

BACHELOR THESIS

Cloud-based Mobile System for Free-Living Gait Analysis

System component : Server architecture

Hampus Carlsson, Marcus Kärrman

Bachelor thesis, 15 credits

Halmstad 2017-05-09

Abstract

Progress in the fields of wearable sensor technologies together with specialized analysis algorithms has enabled systems for gait analysis outside labs. An example of a wearable sensor is the accelerometer embedded in a typical smartphone. The goal was to propose a system design capable of hosting existing gait analysis algorithms in a cloud environment, and tailor the design as to deliver fast results with the ambition of reaching near real-time.

The project identified a set of enabling technologies by examining existing systems for gait analysis; the technologies included cloud computing and WebSockets. The final system design is a hierarchical composition starting with a Linux VM running Node.js, which in turn connects to a database and hosts instances of the MatLab runtime. The results show the feasibility of mobile cloud based free-living gait analysis. The architectural design provides a solution to the critical problem of enabling existing algorithms to run in a cloud environment; and shows how the graphical output of the native algorithm could be accurately reproduced in a web browser. The system can process a chunk of 1300 data points under 3 seconds for a client streaming at 128 Hz, while simultaneously streaming the real time signal.

Preface

What follows is a bachelor thesis written in the spring of 2017. The system is a product of collaboration between people from different fields and countless hours of coding, sometimes into the wee hours of night. Our special thanks goes out to the supervision and mentorship offered to us by Siddhartha Khandelwal and João Bentes.

Hampus & Marcus

The MAREA Gait Team

Siddhartha Khandelwal, João Bentes, Hampus Carlsson, Marcus Kärrman, Tim Svensson

Contents

Chapter 1

Background

1.1 Introduction

Gait is the pattern by which a person walks.

Gait is centered around three main components: locomotion, balance and ability to adapt to the environment. In the human body this is achieved through a balance between various interacting neuronal and musculoskeletal systems[1]. Effects of dysfunction in any of these interacting systems would appear in gait, thus stating the importance of gait analysis. For instance, a neuro-physiological disease like Parkinson's disease (PD) will have measurable effects on gait even at an early stage in the disease's progression[2].

A gait cycle is described as the time between successive foot contacts of the same limb. In one gait cycle there are 2 steps and 1 stride. A step is defined as the time between the heel strike of one foot and the heel strike of the other. A stride is completed after two successive heel strikes on the same foot[3]. Two important events in a normal gait cycle are the heel strike and toe off. They are essential in estimating stride and step parameters and that is why being able to detect these events is of great importance in any gait analysis application(s).

Gait analysis is normally performed in clinical gait labs equipped with various sensing modalities. Labs equipped with motion capture systems and force plate equipment offer very rich data but suffers from not being able to monitor subjects during longer periods of time[4]. Recent advancements in MEMS tech[5], has significantly improved gait analysis by providing wearable sensing technologies and ambulatory systems[6]. These wearable technologies are generally based on inertial sensors such as accelerometers, gyroscopes and magnetometers.

Tests confined to labs and dependent on supervision are suffering from not being able to observe patients during longer periods of time in daily life; they are also expensive to perform. The efforts made into tackling both the problems of cost and observation time has yielded: (i) better algorithms for analyzing gait in different environments[7]; (ii) mobile applications which uses the embedded sensors of smartphones[8]; (iii) remote processing which makes gait analysis available for more people.

There is demand for a framework for performing gait analysis in real-world environments. Some general purpose platforms for performing scientific experiments such as e-Science central[9] have been proposed along the years. However, there remains a challenge regarding integrating applications reliant on a more rich runtime environment, which will typically be the case for gait analysis algorithms. One of the fundamental challenges in such a project entails getting these

algorithms to run in a cloud environment. Making a specialized platform for gait analysis will allow for a more tailored architecture and targeted optimization.

1.2 Problem Statement

1. Can a cloud based gait analysis platform enable existing indoor systems for gait analysis to be applicable in an outdoor environment?
2. Can cloud based gait analysis platform enable real-time or near real-time results?

1.3 Goal

Purpose a system design for a cloud based gait analysis platform focused on near real-time results. Implement and test the proposed system design. Evaluate the system in terms of performance and scalability.

1.3.1 Key Features of the Proposed Platform

The platform will consist of a cloud server, or at least a cloud component (there might be a need to run multiple servers to get the desired performance). There will also be a mobile application which can communicate with the cloud component. The mobile app which is using data primarily from its embedded accelerometer, will be the collector of data in the system.

The platform can perform gait analysis on the data submitted to it from its clients. During development gait analysis will be done using an algorithm which performs detection of heel strike and toe off. The algorithm runs in MatLab.

The mobile app collects data and has the means to transmit that data to the server. Both the mobile and the server component is responsible for maintaining data integrity and ensure the quality of the output.

Interested parties given that they have some required level of access can subscribe to the output of a given process. A process here is a gait algorithm running for an input of data potentially streamed from a mobile app.

As a whole the platform handles the collecting, processing and presentation of the data and the corresponding processed data.

1.4 Limitations

This project is strictly concerned with the development of the **server side of the system**. The proposed mobile app, acting as a sensor gateway is a project of its own. Moreover, testing of the platform in terms of scalability and performance will be done using simulated clients only.

The platform will only be tested with one gait analysis algorithm. The prototype will only be designed to handle data from triaxial accelerometers.

Testing will be done under optimal network conditions; that is, testing will not be done with clients connected via mobile network.

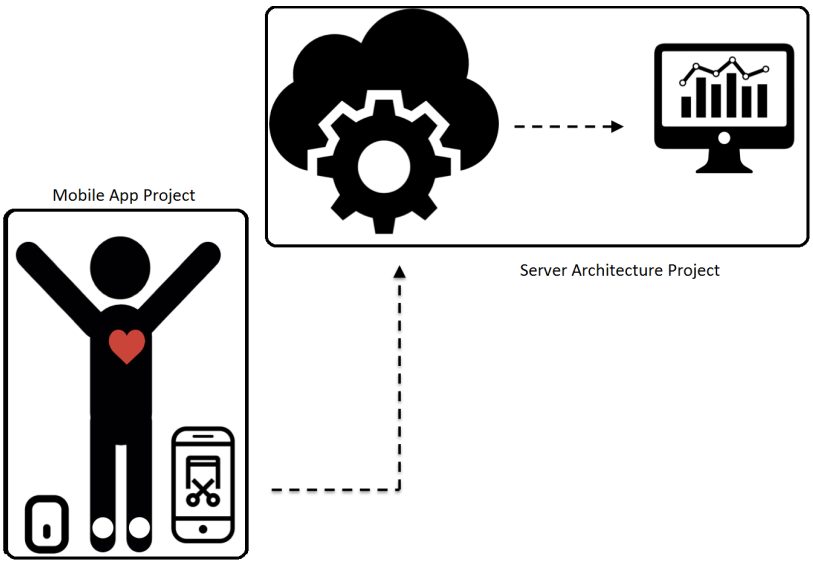


Figure 1.1: Separation between the mobile application project and the server architecture project.

In terms of security, work will be limited to suggestions of future improvements for increased security and protection of integrity (section 7.5). Implementation of security features should be implemented after feasibility of the fundamental components (section 1.3.1) of the suggested platform have been verified.

1.4.1 Project Separation and Assumptions

The project of developing the server architecture starts from the stream of data reaching the server (Figure 1.1). The stream of data is produced from inertial sensors and transmitted via the mobile app (the client). Packets from the client are assumed arrive in "correct" order meaning in the same order as they were sent; in addition packet loss is assumed to be at 0%.

If more than one sensor is streaming simultaneously from one client the assumption made by the server is that these signals are synchronized in time. From an implementation standpoint this means that if there are two streams from one client s_a and s_b the assumption is $t(s_a(i)) \approx t(s_b(i))$ where $t(x)$ is the time of capture of x.

If the client disconnects at any point during data streaming that session is viewed as finished by the server; Any initialization process (appendix B) would have to be done all over again for the client to continue streaming.

A connection between the server and the client is always initialized by the client; the server should, at any time be available for an incoming request to start streaming from a client.

1.4.2 Real-time Definition

For adding substance to the concept of real-time the definition from the book: *Programming real-time computer systems*[10] is used; a real-time system is a system which "controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that time"

1.5 Related work

According to Weiss et al.[11] and Din et al.[12] performing gait analysis in highly controlled settings is suboptimal. This is said to be because of two main reasons. The first being the white-coat effect during clinical experiments causing a bias in the collected data itself. The second is that the subject is not observed during longer periods of time and hence there is lack of data for long-term gait analysis.

Din et al[12] writes: "Free-living gait is naturalistically dual task because of the distractions, environmental obstacles, and task complexities that limit attentional compensation; while conversely attentional control is optimised during scripted gait tests in the laboratory."

For anyone wanting to conduct gait analysis of subjects in daily life there are three main components that must be in place. First, the subject(s) must have access to a mobile sensor, for instance a wearable accelerometer. Second, there needs to be a way to collect the data from the subject and send it to the concerned party, the researcher. Third, the data must be analyzed by the concerned party and "the results should readily available to the subjects". While the first part, i.e. distributing the sensors to subjects for use at home, is a manageable task, the second and third is considered more problematic. As elaborated in [13], after the sensor has been worn at home by a subject, the data must be sent back to the researcher via some file sharing service like Dropbox™. The researcher then has to run the processing algorithm on each and every dataset received. In their example, a monitor period of a single patient for 7 days will result in 250 mb of data. This would then have to be processed, and would take around 20 minutes for each data set to finish. In this instance processing is done in MatLab™.

The e-Science central[9] and BodyCloud[14] are both cloud based systems for data analysis aimed at satisfying similar demands but for different target groups. The e-Sicene central is a platform designed for researchers with the stated aim to: "store, share and analyse their data, and for developers to create new scientific services and applications". BodyCloud is a system built around being able to handle real-time data streams from body sensor networks(BSN[15]). The system addresses processing of real-time data, secure transmission- and storage of data and customizable ways of data presentation. Figure 1.2 is showing a generic BSN setup.

The e-Science central allows for researchers to create workflows by combining applications that either already exists on the platform, or uploading their own applications to the platform. The service is interfaced via a web browser or API, although the API only lets you run a pre-existent workflow. BodyCloud enables, in a similar way to the e-Science central, for the specification and uploading of algorithms and workflows to the platform, in addition it also integrates existing data mining systems.

Din et al.[13] acknowledges that platforms like the e-Science central (and by extension also BodyCloud) have at least the potential to alleviate the task of large scale, unsupervised gait analysis experiments of subjects in their home environment; they then proceed by highlighting a critical problem: in order for the service to be useful it must be able to run what is typically algorithms too sophisticated in terms of libraries and dependencies to be easily deployed.

Moving away from the idea of a general purpose research platform, the work of Pan et. al. [8] introduces PD Dr. This work narrows the focus and concentrates on patients suffering from Parkinson's disease. Realizing the value in monitoring subjects or patients in their daily life, the work of Pan presents a platform for gait assessment that allows the patient to conduct

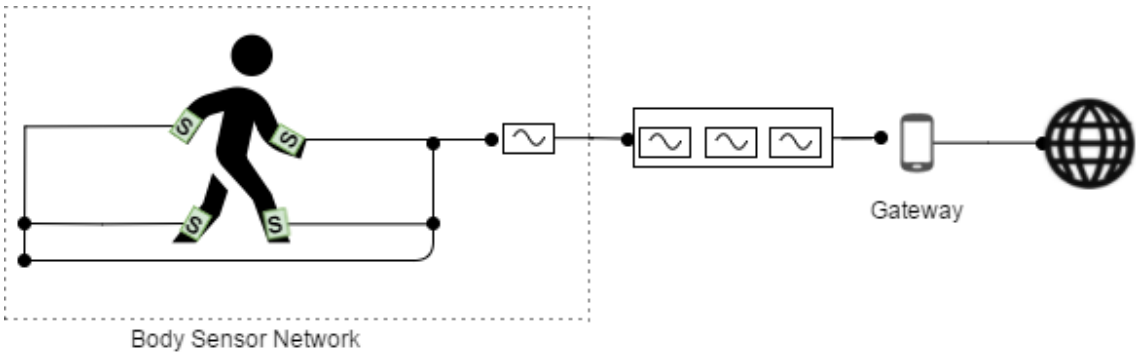


Figure 1.2: A person wearing a set of sensors (S). Together the sensors make up a body sensor network[15]. The network connects to the Internet via a gateway, in this case a smartphone.

tests at home using an app which they call PD Dr. The app presents the patient with a set of tests and instructions on how to go about in performing them. The app uses the embedded accelerometer of the phone to collect data. After a test is completed, the data is sent to a cloud server for processing. The resulting output is sent back to the client and can also be accessed by a doctor via a database. The cloud component of PD Dr is modular in design and is argued to scalable. But according to the authors, the system is not designed to perform long term continuous monitoring of subjects.

The approaches on the literature indicate that to get the better results from gait analysis, subjects need to be monitored in their natural environment and ideally also during longer periods of time. Understanding that studies on large scale becomes resource intensive due to the amount of manual labor needed there is demand for a platform to overcome these problems.

Making use of personal devices such as smartphones helps in further lowering of thresholds for large scale gait analysis, this is demonstrated with the mobile application and cloud service PD Dr. While PD Dr allows for 'at home' gait assessment it lacks the general purpose of the e-Science central and long term monitoring capabilities.

We see that in all of the above examples there is no real mention of optimization of delivering output with short delay. The idea of not only collecting data in real-time, but also analyzing it and getting results back in real-time appears yet unexplored; even though there might be applications of gait analysis which would benefit greatly from real-time feedback. Additionally, no solutions are presented for running existing algorithms in a cloud environment.

Chapter 2

Theory

In this section existing methods of gait analysis are summarized and common features among the subset aimed at gait analysis outside labs are highlighted. Existing technologies that could allow similar systems to work in a cloud based environment are presented.

2.1 Common Methods for Gait Analysis

There exists today many different systems and methods for performing quantitative assessment of human gait. These systems range from dedicated gait labs[16], to systems based on small wearable inertial sensors[17]; while the former provides rich data and is considered the gold standard, the later offers mobility and lower cost.

A paper by Muro-de-la-Herran et al.[18] from 2014 reviewed 3 different approaches to human gait analysis namely: image processing, floor sensors and wearable sensors. They describe how different methods of gait analysis allow for extraction of different gait parameters. Some examples of gait parameters are: *cadence*, *stance time* and *swing time*; extraction of gait parameters enables quantitative assessment of gait, examples of which are *symmetry* and *normality*.

The following is a short list describing different types of sensors and what gait parameters could be extracted using them, elaborated in[18].

Inertial sensors have been used to detect steps, determine stride length and assess symmetry in gait. They are typically small and wearable. Accelerometers and gyroscopes are both inertial sensors.

Pressure sensors can be embedded into insoles to measure ground reaction force with strong correlation to clinical control measurements. Pressure sensors is one of the sensing modalities that can be used for gait phase detection.

EMG sensors (electromyogram) can be used to measure muscle contraction by placing electrodes on the body of a subject. EMG sensors belong to the class of wearable sensors.

2.2 Existing Gait Analysis Systems

Five systems for gait analysis are here examined for their characteristics in terms of how sensor data is produced, handled and in what domain they operate; this is to better understand which technologies are required to enable similar systems to be compatible with a cloud based platform.

(A) The GaitShoe
In a study by Stacy J et al.[19] a wireless wearable system aimed at gait analysis outside of the traditional motion laboratories was developed. The developed system, which could be attached to the shoes of the subject, was equipped with inertial- and pressure sensors; The data collected by the system was wirelessly transmitted to a base station nearby; the data was transmitted at a rate of 75 Hz.

(B) A pressure sensitive floor
The study "Your Floor Knows Where You Are: Sensing and Acquisition of Movement Data" by Philipp Leusmann et al. [20] explore the possibility of electronic health-care using pressure plates in a home environment. A floor covering surface was constructed using multiple Arduino Mega microcontrollers connected to pressure sensors; the network of sensors transferred data to a host computer where it was processed and analyzed by a software developed in Java. The system was evaluated using accuracy of step detection as a metric.

(C) Gait symmetry and normality assessment using inertial sensors
A study by Anna et al.[17] showed how gait symmetry and normality could be assessed using wearable inertial sensors. The sensors used in their experiment were 3 Shimmer® sensors attached to the body of the subject; the sensors were sampling at a rate of 128 Hz and data was stored at the sensor node. Processing of the signal data was later done in MATLAB.

(D) S.J.M. Kinetic gait analysis using a low-cost insole
An insole equipped with 12 pressure sensors was developed as a system for clinical as well as at home gait assessment[21]. Data from the embedded sensors was streamed at a rate of 118 Hz to a receiver plugged into a nearby laptop. Data collected by the insole was later processed in MATLAB.

(E) BTS Gait Lab
The BTS gait lab[16] is a commercial system for clinical gait analysis. The system is equipped with 8 infrared cameras, 6 pressure sensitive plates and 8 EMG probes. Systems like these are often considered to be the gold standard and are commonly used as reference when evaluating other gait analysis systems.

Systems for Free-living Gait Analysis
Systems **A**, **C** and **D** are all systems aimed at enabling gait analysis outside the lab environment. Common features among these system are that they are wearable, work at high sampling rates and use specialized software to process the collected data. The studies in which they are presented all stretch the importance of enabling gait analysis of subjects in a more natural environment.

By enabling systems like **A**, **C** and **D** to be further separated in terms of where data is collected and where data is processed by using enabling technologies such as cloud computing, the ability to monitor subjects in natural environments may improve.

2.3 Enabling Technologies

Remote processing
Systems **A**, **C** and **D** all operate in close proximity to their receiving nodes; incorporating such systems into a more geographically independent platform entails the ability to do remote

processing. By utilizing the capabilities of cloud computing, this could potentially be achieved. The concept of cloud computing is presented in section 2.3.1.

Data transmission
Systems **A**, **C** and **D** indicate that working at a high sampling rate has positive effects on the quality of data; among these systems the lowest sampling rate is 75 Hz. Allowing them to work over the Internet means finding communication protocols with the ability to transmit data at such rates while still providing high data integrity; in section 2.3.2 such a protocol is presented.

Web frameworks
At the receiving end of the data stream from the wearable systems data must be handled in a structured way as to be able to deal with many concurrent sensor systems connected to the platform. For real-time gait applications the transmission rate or *streaming rate* will be close to the sampling rate of the system. In section 2.3.3 a modern framework for web development is presented.

To estimate the total number of incoming packets from a given number of clients, Eq 2.1 is used. C is the set of all clients connected to the server and $S = \{s_i\}$ is the family of sensors indexed by C where $s_i \triangleq [(f_0, f_1, ..f_n)^T]$ f_n is the streaming rate of that sensor. The number of packets p received by the server every second when there are 10 connected clients, each wearing 2 sensors streaming at 128 Hz is 2560.

$$p = \sum_{i=1}^{|C|} \sum_{k=1}^{|s_i|} s_i(k) \tag{2.1}$$

Signal processing runtime environments
Systems **C** and **D** both explicitly state their use of specialized runtime environments for processing of sensor data. Moreover, algorithms such as the one presented in *"Gait event detection in real-world envi-ronment for long-term applications"* [22] which enable gait analysis in more varied environments still relies on these specialized tools. Section 2.3.4 briefly elaborates on how MatLab can be used as an integrated component in a cloud based web system.

2.3.1 Intro to Cloud Computing

Cloud computing is a way of hosting server programs, web servers and data storage on remote servers which are maintained by third-party data centers. Cloud hosting companies provides its customers with resources like RAM, CPU power and storage for leasing on-demand[23]. In Figure 2.1 a simple example of how cloud computing could be used is illustrated. For example, an application which demands hosting capacity for its users, could instead of purchasing a physical server, deploy the application on a cloud hosting platform; these are more adaptable to increase in amount of traffic from users.
Cloud providers typically offer flexibility in terms of CPU, RAM, bandwidth, storage and location; meaning that a service can be upgraded to fit increase, or decrease in demand. These upgrades can be done fast in the cloud, instead of upgrading a server with physical components. The model commonly used by cloud providers is "pay as you go" which works as such that having 1000 servers running 1 hour costs the same as having 1 server running 1000 hours[24].

Virtual machines (VMs) are offered by most of the cloud providers on the market. Customers are essentially renting a segment of a server from the host with specific RAM, CPU power and

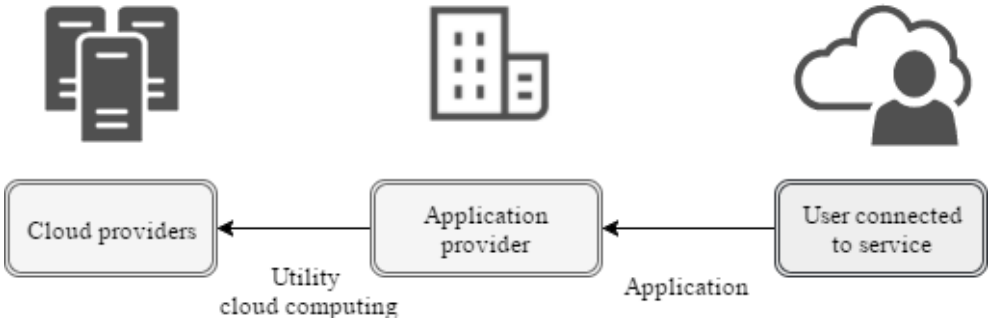


Figure 2.1: The cloud and the customer

storage of the customers choice and includes the option of choosing what operating system the server should run. Accessing the virtual machine is done through the Secure Shell protocol (SSH)[25] that encrypts the messages between user/server and intend to provide a secure channel over an unsecured network. SSH enables a channel for communication and remote access of the virtual machine.

There are many competing cloud providers, focusing on different types of needs of the clients. One subset of the cloud computing is the IaaS (Infrastructure as a service) which is the cloud-service model for providing virtual machines. This report will only briefly present two of the market leading IaaS service providers.

Google Cloud™ offers a free trial with $300 start credits that the new user can spend however they want over the coming 90 days. However there are restrictions, only 8 cores (or Virtual CPUs) can run simultaneously. Google™ offers a set of different virtual machines tailored for different purposes, some of them listed here:

- Standard machines - suitable for tasks that have a balance of CPU and memory needs.
- High-memory machines - suitable for tasks with heavy RAM usage
- High-CPU machines - suitable for tasks with heavy CPU usage

Amazon Web Services™ (AWS) is the leader in market shares in cloud computing,[26] they also offer free tier virtual machines for 12 months [27]. The instance that is included in the Amazon™ free tier is the t2.micro instance which have a single Intel Xeon CPU and 1GB of memory[28].

2.3.2 The WebSocket Protocol

The WebSocket protocol is a web oriented communication protocol built on top of TCP (Transmission Control Protocol)[29]. The guarantee of package delivery and ordering is inherited from the TCP protocol[30], on top of which it is defined.

Websocket was standardized in 2011 by the IETF(Internet Engineering Task Force) as RFC 6455 [29]. It provides reduction of network traffic and latency in comparison to the previous solutions like the HTTP protocol by minimizing the size of the package header.

Figure 2.2: The relationship between the Server and clients C_1 and C_0 from the example.

In the WebSocket protocol a connection is initiated with a handshake[1] which consists of an HTTP upgrade request that is performed by the client, this request can be accepted or rejected by the server. After the upgrade request has been accepted a bi-directional channel between the server and the client is opened.

WebSocket packets can be sent in one or more frames[2], each packet consists of a minimal header and a payload, the minimal header is key in making the protocol fast. WebSocket is a message based protocol; but the fact that there is no limit within the protocol itself to inhibit infinite fragmentation of data being sent, it is feasible to make it behave like a streaming protocol.

The following is an example aiming to show the utility in the WebSocket protocol compared more traditional ways of web communication: Consider a server and two clients, C_0 and C_1. The clients are each connected to the server but not to each other (figure 2.2). C_0 is a weather station and sends data regarding the state of the weather to the server over the Internet. C_1 wants to display graphs showing how the weather changes over time. When there is new data from C_0, C_1 wants to add this to the graphs as soon as possible. The data coming from C_0 does not necessarily arrive with a fixed interval. The problem for C_1 is to know when new data has arrived from C_0.

Two solutions are here proposed: HTTP long polling and WebSocket based communication. HTTP long polling is when the party wanting data as soon at it arrives (in this case C_1) sends a HTTP GET request to the server, the server then *holds* the request until there is data to send back. In this particular example, the communication flow using HTTP long polling could look as follows: C_1 send a request to the server for new weather data. There is none at this time so the server holds the request until C_0 has sent a new batch of data. The GET request from C_1 is now "loaded" with the new data and sent back. And so the process repeats.

Solving the same problem using WebSockets allows data from C_0 to be pushed directly to C_1 upon arrival without the transmission of each packet be preceded by a GET request from C_1 (given that the server and C_1 has a previously established connection).

A study by V. Pimentel and B. G. Nickerson[33] has compared the average latency for real-time communication over the Internet using long polling and WebSockets. In that particular experimental setup, on average, the performance between the two are close in terms of one-way latency. The measurements was done is sessions of 5 minutes and data was sent at a 4-Hz rate.

2.3.3 Node.js

Node.js or node, is "an asynchronous *event driven* JavaScript runtime"[34]. It is a server side framework running on top of the Google V8 JavaScript engine. Node.js does not adhere to the

[1] "In information technology, telecommunications, and related fields, handshaking is an automated process of negotiation that dynamically sets parameters of a communications channel established between two entities before normal communication over the channel begins"[31]

[2] "A frame is a digital data transmission unit in computer networking and telecommunication"[32]

common multi-thread approach to concurrency.

As explained in [35], node makes use of asynchronous I/O which is a solution to the same type of problem solved by multithreading, namely reducing the penalty of blocking procedures. In a multithreaded system, idle time due to waiting for some I/O operation, will be filled by work on another thread via a context switch. The asynchronous event driven architecture of node allows it to be run in a single non-blocking thread where I/O-contingent procedures will make use of callbacks or promises for continued execution. This works well within the JavaScript language due to it supporting higher order functions i.e. functions that take other functions as parameters.

For each instance of node, there is only one call stack. Synchronous calls will be placed directly on the call stack, however, I/O tasks uses the low-level C/C++ API provided by the V8 engine. Function calls which uses the API will not end up directly on the call stack; they will be sent to the API. When an I/O request completes, the associated callback is placed in the callback queue. When the call stack is empty, the *event loop* will put the next queued callback on the stack, see Figure 2.3.

For utilizing more of the potential of the hardware, for instance using more than one core, node provides support for child processes. A child process can be either a node program itself or it can be some other program, like a python script. This is done via the native modules spawn[36] and fork[37], fork being the special case used for instantiating node scripts; forked processes will have their own instance of the V8 engine.

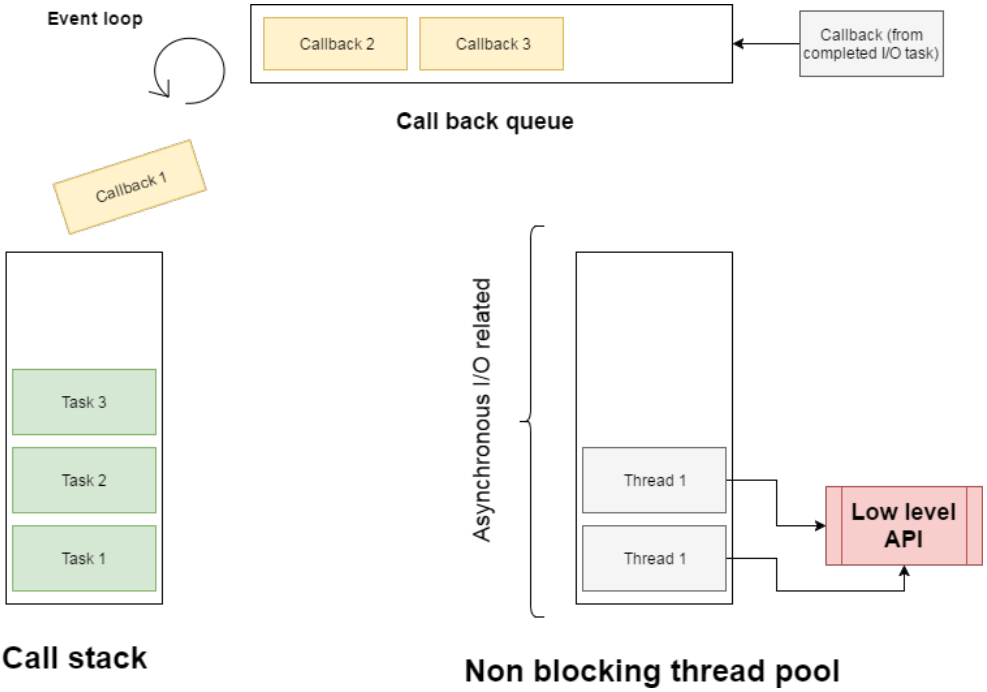


Figure 2.3: Node.js workflow. The call stack handles all synchronous calls in order. Asynchronous tasks such as I/O is not put immediately on the call stack but are handled by the low level API. When asynchronous tasks complete the corresponding callback is put in the call back queue. When the call stack is empty, callbacks are de-queued and put on the stack.

2.3.4 MatLab and the MatLab Engine

MatLab (Matrix laboratory) is an environment and a programming language for manipulating matrices and doing a wide range of various of mathematical operations, as well as simulations.

The algorithm used for gait signal analysis in this instance of the platform is written in MatLab, it is therefore a requirement for the cloud server to support the MatLab runtime. This report will not go into details regarding the algorithm, for further reading see: "Gait Event Detection in Real-World Environment for Long-Term Applications: Incorporating Domain Knowledge into Time-Frequency Analysis" by Siddhartha Khandelwal and Nicholas Wickström [22]

Functions written in MatLab can be executed by other processes with the use of "MatLab engine". The MatLab engine loads the specified function(s) into the runtime and pipes the output back to the parent process. At the time of writing, the MatLab engine has support for Java, Python C/C++ and Fortran[38][39]. The MatLab engine is included in the desktop version of the software but does not require a graphical user interface.

Chapter 3

Method

3.1 Project model

The project of developing the gait platform contained within 2 sub-projects, one being this one, the cloud server; the other one being the mobile component. The mobile component was the gateway of the wearable sensors of a given client. The 2 components, the server and the client, are mutually dependent and has to agree on the ways of communicating with each other.

The 2 projects worked together using Scrum which is an agile software development methodology. The development team(s) worked in sprints of two weeks (on some cases a "quick" sprint was planned for a single week). By the end of every week, a meeting was held among the developers. On the meetings the state of progress was discussed and feedback was given; meetings coinciding with a completed sprint would involve evaluation and planning of the next sprint.

In the version of Scrum which was adopted, a sprint was composed of a set of tasks, where each task was defined by the team at the beginning of the sprint. A task contained a brief description, a list of dependencies and a test description. Ideally, the tester of a task was not the developer of the task.

The platform was developed from the bottom up; starting with fundamental features like connection between *publishers* and *subscribers*. A publisher was a client on the platform who was sending data for processing. A subscriber was a client on the platform who received processed and unprocessed data. Building from the bottom up made it easier to identify problems at an earlier stage in development than would been possible taking a top down approach.

Early discussions regarding system architecture explored the micro service architecture[40]. Briefly, the micro service architecture is aimed at separation of concerns in distributed systems. The micro service approach advocates splitting system components into sub-modules, opening up for internal communication via for instance a REST[41] API. This was argued to be a scalable approach[42] but has also been criticized for being labor intensive to maintain[43].

The micro service architecture was among the members of this project viewed as a good fit for the task at hand. However, due to time constraints a more pragmatic (bottom up) route had to be taken so as to develop a proof of concept prototype; testing of the prototype would then reveal information about what would be the ideal architecture for the gait platform.

3.2 Frameworks, Protocols, Tools and Design Pattern

This subsection elaborates on the initial decisions surrounding the choice of: design pattern for the software architecture, web framework, communication protocol and database. The decisions were made in the order that they are presented.

3.2.1 Web Framework

The choice of web framework was made from a selection of the most popular frameworks for web development at the time[44]. A list of preferred features was used to narrow the field down to one.

The list of options consisted of Node.js (section 2.3.3), ASP .NET[45] and Django[46] for python. ASP .NET is a web application framework developed primarily by Microsoft. The framework had rich functionality in the domain of web and used threads as a concurrency model.

Django is a web framework written in python. Django did not handle requests directly but instead used the Web Server Gateway Interface(WSGI). WSGI was an interface for web applications written in python and acted as the bridge between the web server and the framework.

Features of interest

Linux	The free tier VM options of both AWS and Google were limited instances running Linux distributions (Ubuntu).
MatLab engine	Support for the MatLab engine, either directly or indirectly was needed to run the processing algorithm.
Prior experience	Prototyping could be made faster by opting for a framework with which the project members had prior experience.
Asynchronicity	An asynchronous model based on child processes instead of threads could remove some of the concerns regarding protection of shared resources; as an added benefit the use of child processes would advocate a more modular design which could enable easier scaling.

Evaluation with regards to the features of interest revealed Node.js to be the better choice, see Table 3.1.

3.2.2 Real-time Protocol

As clients on the platform would typically send data way beyond the 4-Hz rate that was used in the HTTP long polling vs. WebSockets experiment[33], a streaming based protocol was deemed more fitting. A continuous stream of data from the sensors worn by the client had to be transmitted shortly after the time of collection, without this real-time could not be achieved.

The WebSocket protocol provided the key feature of allowing for data streaming between a client and the server; being lightweight in terms of header size would also help limit the mobile data usage of a client. The WebSocket protocol was therefore the most compelling choice at the time[1] and was integrated into the platform via socket.io[49].

[1]The protocol MQTT[47] was discovered later to be a potentially better fit for the platform having similar qualities as WebSockets and also offering secure communication vi MQTT-S[48]

3.2.3 Database

For long term storage of sensor data, some kind of database had to be integrated into the system.

MongoDB[50] is a NoSQL database and an open-source document oriented database; this meant that any JSON(JavaScript Object Notation) object could be inserted straight into the database as there were no strict schemas to follow.

MySQL[51] is an open-source relational database management system (RDBMS) that was widely used by big companies like Facebook and Google. The schemas were strictly defined and tables were connected via a relational structure.

PostgreSQL[52] is an object-relational database (ORDBMS); it had a similar structure to MySQL but in addition it also had an object-oriented database model. What this meant was that it supported objects, classes and inheritance, similar to the object-oriented programming languages.

Features of interest

Dynamic Schemas	A priority since the project used a bottom up approach and the structure of tables in the database could change during the project's progression.
Json	The fact that Node.js was used, meant that all the server side code would be written in JavaScript and objects in default were in JSON format. It was therefore beneficial to have a database in the same format.
Auto sharding	Enabled a kind of load balancing where the user could choose a shard key[53] which determined how the data should be distributed among multiple servers.
Rich querying	A feature that allows for the combination of smaller queries into more complex functions for retrieving data, joining tables etc.
SSL	Having a secure channel between the server and the database was a crucial feature for the project's future development.

After evaluation (Table 3.2) MongoDB and PostgreSQL got the same score but MongoDB was favored due to easier integration with the chosen web framework.

3.2.4 Design Pattern

System attributes that followed from choice of technologies and desired structure are listed below; Table 3.3 is a mapping between the system attributes and established design patterns. The combination of design patterns would become the starting point for the system design.

System attributes

Event handling (EH)	Implementations of message based protocols like WebSocket[29] and MQTT[47] in the Node.js framework handled messages in terms of events and event handlers.
IPC handling (IPCH)	Node.js used child processes communicating using inter-process communication (IPC); the design pattern had to address the issue of handling the IPC data streams.

Clear abstractions (CA) The system would be composed of a set of different entities, for example the entities generating sensor data and entities for receiving data; these entities had to relate to each other such as to provide a clear abstraction of which generating entities were related to which receiving entities.

The following is a subset of the common architectural styles presented in the book "An Introduction to Software Architecture"[54].

Pipes and filters (PF)
Pipes and filters is an architectural style where each component of the system can receive- and output data. The connectors between two such components are called pipes. This style enables a component to start streaming data on its output channels before the corresponding input has been received in full; this is enabled through the use of filters, which are independent structures defined to enable piecewise handling of input data.

Data abstraction and object-oriented organization (OO)
Object-oriented organization is a set of ideas around software architectural design which is widely adopted. The advantages of this style of architecture comes from being able to abstractly, represent collections of entities and their operations as one coherent object.

One disadvantage of this style is that for objects to communicate, the locations of the object must be known; hence the systems must provide ever higher level abstractions to act as a reference medium so as to allow for this functionality.

Implicit Invocation (II)
"The idea behind implicit invocation is that instead of invoking a procedure directly, a component can announce (or broadcast) one or more events"[54].

By adopting the style of implicit invocation system components are given more autonomy which enables them to change independently to the event source. This style is well suited for use in distributed systems as components can evolve independently without disrupting the pre-existing structure.

Table 3.1: Comparing some of the alternatives for web frameworks, among the three alternatives Node.js got the highest score.

Web framework	Linux support	MatLab engine support	Prior experience	Asynchronicity
Node.js	1	1	1	1
ASP .NET	0	1	1	0
Python (Django)	1	1	0	1

Table 3.2: Comparing some of the alternatives for choice of database. MongoDB and PostgreSQL got the same score but MongoDB was favored due to easier integration with the chosen web framework.

Database	Dynamic Schemas	Json	Auto sharding	Rich Querying	SSL
MongoDB	1	1	1	0	1
MySQL	0	0	0	1	1
PostgreSQL	1	1	0	1	1

Table 3.3: Mapping of systems attributes to architectural design patterns.

Event handling	**II**
IPC handling	**PF**
Clear abstractions	**OO**

Chapter 4

System Design

The server architecture was composed of a hierarchy of child processes; this to increase performance when more than one client was streaming data to the server at a time. The hierarchical structure is composed (Figure 4.1) of three layers: the WebSocket layer, the session.js layer and the processing layer.

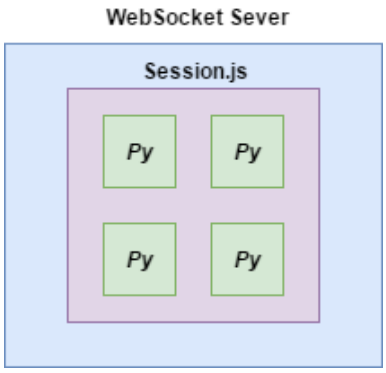


Figure 4.1: The three layers of child processes in the server architecture. The WebSocket layer, session layer, and processing layer(marked "Py" for python). The WebSocket layer communicates directly only with the session layer; in turn the session layer is the only process in direct communication with the processing layer.

4.1 Publishers, Subscribers and Sessions

A client streaming data from a sensor was given the name **publisher**; each publisher was assigned one **session**. The session is an instance of the aptly named session.js and served four main purposes:

1. *Buffer the incoming data points.* The session was being piped all the data points from its publisher; the data points were pushed to a buffer awaiting processing.

2. *Manage algorithm parameters.* Different processing algorithms would have different requirements regarding input parameters. For instance, the algorithm used under development required sampling frequency as an input argument; it also required a data vector of at least 1300 in length to be able to do the processing. The required amount of data points to perform a round of processing was given the name **minimum chunk**.

3. *Load the processing scripts.* The session is the parent of the child process(s) doing the actual data processing. It was the task of the session to detect when the buffer had

enough data and thereafter load the child process with the data points from the buffer and other input parameters.

4. *Handle process output.* When the processing of a set of data points had finished, the results were piped back up to the session. The results was then stored in the database; in addition to storing, the results were also piped up to the WebSocket server layer which in turn would send the results to the **subscribers** of that session.

Subscribers was the name given to the clients which did not produce data, but instead desired to display the data from a publisher. A subscriber would receive not only the results from a publisher's input stream but also the data stream itself in *real-time*. Figure 4.2a is the system architecture as it was when the basic components *publishers*, *subscribers* and *sessions* were in place.

4.2 Python and the MatLab engine

The session could in theory support any child process, or chain of child processes for hosting a given gait processing algorithm; the minimum requirement was that it could be instantiated by either a Spawn[36] or a Fork [37].

To bridge the gap between the MatLab engine and the Node.js, a python script (spawned from the session), was used as an intermediary between the two run time environments. In this particular instantiation of the platform the python child process managed the following:

1. *Read chunk from parent.* The python process would be started and placed in a state of waiting for a chunk from the parent session via stdin.

2. *Load MatLab engine.* Once a chunk had been read it was passed on to the MatLab engine.

3. *Collect MatLab engine output.* Once the processing algorithm had finished, the output would be collected and piped back up to the parent session.

The python child process can be seen as p_{icn} in Figure 4.2.

4.3 Channels for Publishers

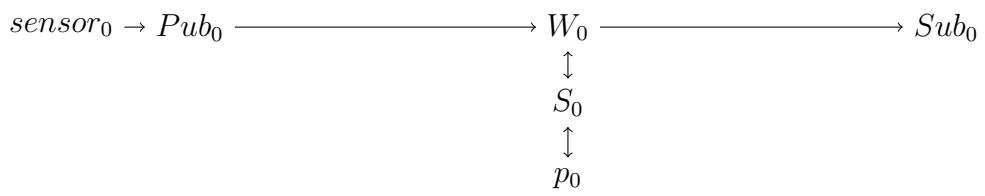
To allow for a publisher to stream data from more than one sensor at the time, channels were introduced to the system, see Figure 4.2b. Without channels a publisher was only mapped to one session; via the introduction of channels, a publisher was now mapped to *a set of sessions*.

Subscribers were still connected to the sessions. A publisher could be subscribed to using the publisher-identifier and a channel-identifier.

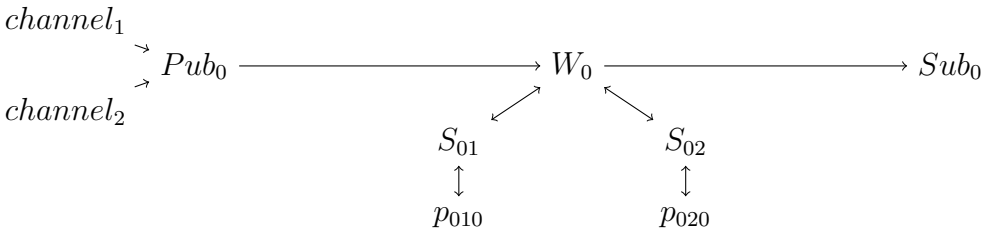
Channels were the feature that could potentially enable gait analysis applications that would depend on *symmetry of the limbs*, this because such analysis would require a subject to wear more than one sensor at a time.

4.4 Improving upon Performance

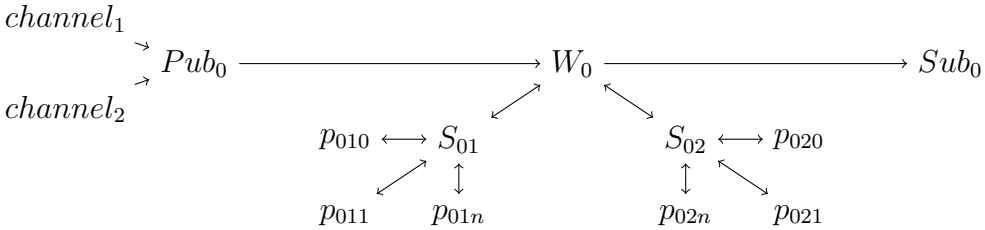
The time between a measurement being received, to being piped to a session, used in gait analysis and being stored as a result in the database was used as an overall metric of platform performance.



(a) Basic system design. Only one sensor per publisher Pub with only one python instance p running in the session S at a time. No queuing systems in place.



(b) Channels added to the system architecture allowing for the publisher Pub to stream from more than one sensor at a time.



(c) Final version of the system architecture supporting multiple sensors for publishers and parallel chunk processing. The python processes p_{icn} can be queued and re-queued after a chunk has been processed.

Figure 4.2: The evolving system architecture. Subfigures (a), (b) and (c) shows how the system grew from one sensor per publisher Pub_i to sensors being represented by channels $channel_c$. Every channel gets its own session S_{ic}, which maintains the python children p_{icn}.

It was recognized that time would be spent "in" one of the following three:

1. *Time to reach minimum chunk*, defined as the time in seconds to reach required amount of measurements to perform a round of processing.

2. *Mean chunk procession time*, defined as the average time for a round of processing to complete, for a set number of simultaneously streaming publishers connected to the platform.

3. *WebSocket server overhead*, defined as the time not spent in (1) or (2)

(1) was believed to be improved by either decreasing the size of the minimum chunk or increasing the streaming rate from the publishers; since the size of the minimum chunk would be a requirement of the gait processing algorithm in use, increasing the streaming rate would be the only parameter under control.

To decrease (2), each session would get a set of gait processing algorithm hosts (python child process running MatLab engine), placed in a queue structure and dequeued when minimum

chunk was reached. This way, many chunks could be processed in parallel.

(3) was addressed by restricting the number of parallel python children running MatLab engine in a session. The python child processes could also be *requeued* after having finished from a round of processing; not letting the python children terminate after every round allowed for reuse of the same instance of the MatLab engine.

The finalized system architecture is pictured in Figure 4.2c.

4.5 Long Term Storage

There were two different databases created for the platform, both hosted on the AWS instance first used to host the entire project. The first database was used for the long term storage of results, users and sensor data or "measurements"; the other database was used for storing the time logs used in system diagnostics.

Three collections existed within the first database: users, which is the set of users that have permission to use the system; measurements, the set of data points streamed from the clients (example in Table 4.1) and results which is the set of result objects generated using a subset of the measurements.

Table 4.1: Example of a measurement object from database

_id	ObjectId("58a5b4d4acef3c2c58dd7149")
accVec.accX	-17.255
accVec.accY	-7.216
accVec.accZ	-11.922
userID	ObjectId("589d7952f57abd03decc43c8")
userTs	1487254740697
serverTs	1487254740699
index	4000

Chapter 5

Evaluation

5.1 Experiment Setup

Performance was tested using a varied number of clients, streaming data at different rates. During performance testing clients were simulated in Node.js. The simulated client written in node adopted the same communication protocol as of that between the server and a real client[1].

During testing the system was hosted on a virtual machine on Google Cloud with 4 vCPUs, 10GB memory and an SSD disk with 30GB of storage. The choice of cloud provider is elaborated in appendix A.

Experiments were conducted for a range of 1 - 5 simultaneous clients with 2 sensors on each client. The streaming rate was varied between the frequencies: 128, 80, 62.5, 40, and 20 Hz. Signal data was simulated using the MAREA Gait Database[55]

5.1.1 Average Measurement to Result Projection Time

All measurements were time stamped upon arrival, and again when the result in which that measurement was used, had been stored in the database; this time difference is denoted t_j in Eq 5.4. All measurements went via a buffer to the gait analysis algorithm; the buffer threshold, as explained above had the name minimum chunk or mc. Because a result was always stored together with the measurements used in generating that result, every result would be associated with a vector of measurements of exactly mc length; furthermore, based on order of arrival, a measurement would have an index j where j would be a member of J.

By averaging the sum of the $t_j s$ from dt_1 to dt_i in DT, the **average measurement to result projection time**($ARPT$) with respect to order could be calculated.

$$DT = \{dt_1, dt_2, .., dt_i, .., dt_M\} \tag{5.1}$$

$$dt_i \triangleq [(t_1, t_2, .., t_j)^T] \tag{5.2}$$

$$J = \{1, 2, 3, .., mc\} \tag{5.3}$$

$$ARPT_j = \frac{1}{M} \sum_{i=1}^{M} DT_{ij}; j \epsilon J \tag{5.4}$$

[1]Here "real client" refers to an actual person wearing the sensors and streams the data via the mobile app.

5.1.2 Mean Chunk Processing Time

The mean chunk processing (MCPT)(def 2) was calculated using a Δt value from time stamps t_0 and t_1 where the t_0 was taken right before data was piped to the algorithm and t_1 right after results arrived from the algorithm.

$$MCPT = \frac{1}{N}\sum_{i=0}^{N-1}\Delta t_i \quad N = number\ of\ results \tag{5.5}$$

5.2 The Dashboard

The Dashboard was a fully separated webserver written in Node.js, it is the **implementation of a subscriber** designed to illustrate the ability to reproduce the output of the native algorithm runtime. From the Dashboard a user could subscribe the live signal from one of the connected patients (publishers).

The live signal would be plotted in real-time on the webpage and could be overlayed with signals from the other sensors worn by the client. When the gait events were computed for a set of data points they would be projected onto the live signal (Appendix **??** Figure 6.7a). There were also other metrics on display such as stride time and number of steps.

The Dashboard served as a test for the real-time aspects of the platform and illustrated one of the ways the output data from the platform could be used.

5.2.1 Implementation details

The surrounding structure of the visual components of the Dashboard was built upon a pre-existing template called "Light Bootstrap Dashboard" [56]. All the graphs for data representation were developed from scratch using a JavaScript library for manipulating SVG images called d3.js [57].

Although d3 proved very useful, the development of the visual data representation was time consuming, a lot due to having the plot of the walking signal be updated in real-time.

The Dashboard connects to the gait platform via WebSockets; it adapted a similar protocol to that of a publisher but it needed to supply information regarding what publisher and channel it wanted the data from. Once the Dashboard had subscribed to a publisher, the stream of data would continue as long as that publisher remained connected to the platform.

Chapter 6

Result

The results in section 6.1 have been generated using the logging system on the server. Concepts such as mean chunk processing time (MCPT)(2) and WebSocket server overhead (WSO)(3) are used throughout to understand the meaning of the data.

Section 6.2 shows how the native algorithm output could be reproduced on the developed Dashboard.

6.1 Table and Diagram Contents

Figure 6.1 is showing the average result projection time (ARPT)(Eq 5.4) and how it changes dependent on order and server load. Messages arrive from a publisher and they have some order; for a specific message, the order of its arrival determines to some degree how long time there will be until that specific measurement is used in gait analysis. Figure 6.2 illustrates how order influences ARPT.

The green area in Figure 6.1 is the mean chunk processing time for that experiment superimposed on the ARPT plot; the blue slope indicates how the delay decreases for data points closer to the minimum chunk (1300); the remaining space comprised of $min(ARPT) - MCPT$ is the WebSocket server overhead.

Figure 6.3 is showing how the area, or integral changes at different streaming rates and publishers. Steep slope relates to high scaling penalty.

Figure 6.4 and 6.5 shows how the MCPT (green area in Figure 6.1) and WSO (orange area in Figure 6.1) is changing at different streaming rates and publishers; furthermore, Figure 6.6 plots the difference between the MCPT and WSO.

Finally, Table 6.1 displays the measured CPU usage (and RAM) with the corresponding MCPT during the experiments.

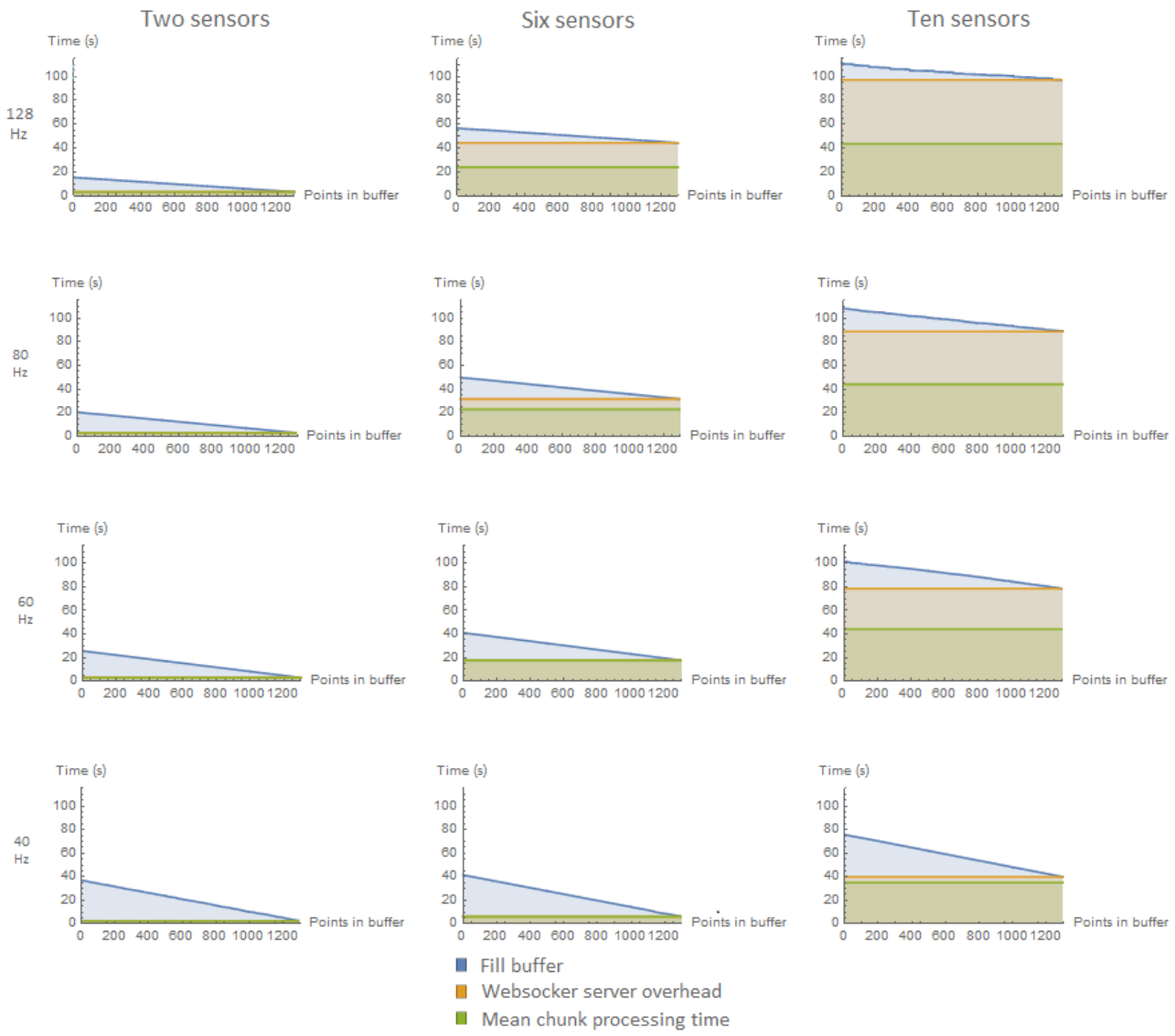


Figure 6.1: Table of plots showing how the ARPT is distributed for varying amounts of sensors and streaming rates.

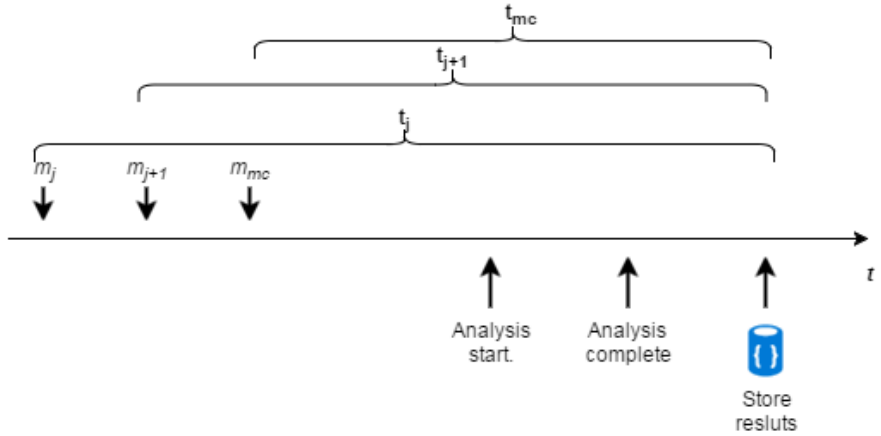


Figure 6.2: Time line for a set of measurements, where m is a measurement in the system. t_j is a time marker for that specific measurement; ARPT is computed using the the time markers t_j

.

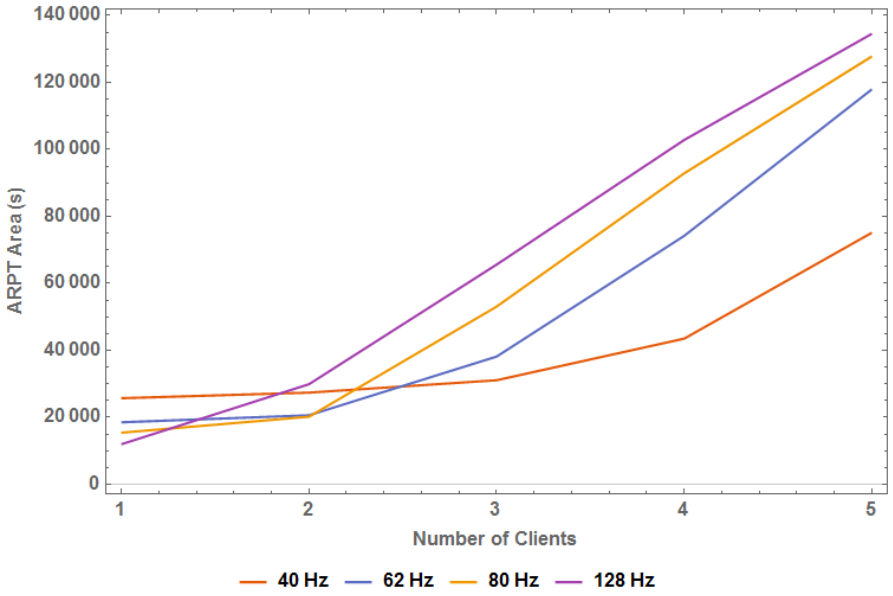


Figure 6.3: The area or integral of the graphs in Figure 6.1, and how it changes at different streaming rates for a varied number of simultaneous publishers.

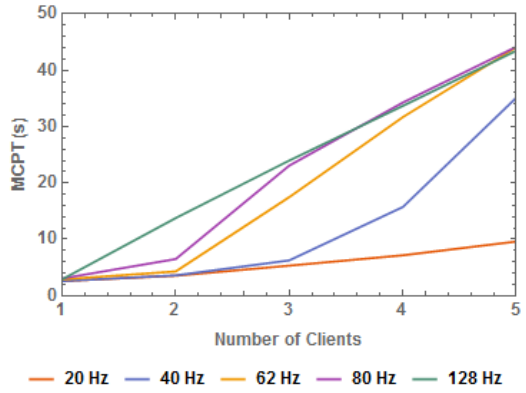


Figure 6.4: The change in mean chunk processing time (MCPT) as the number of simultaneous publishers increases.

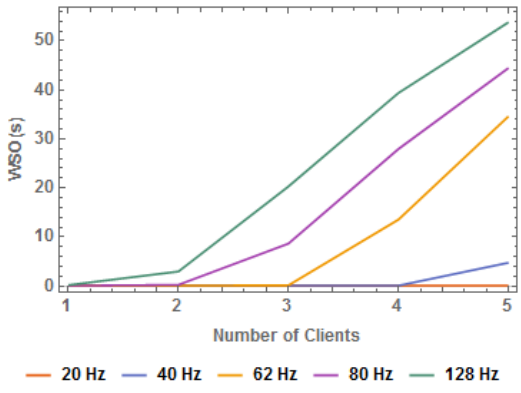
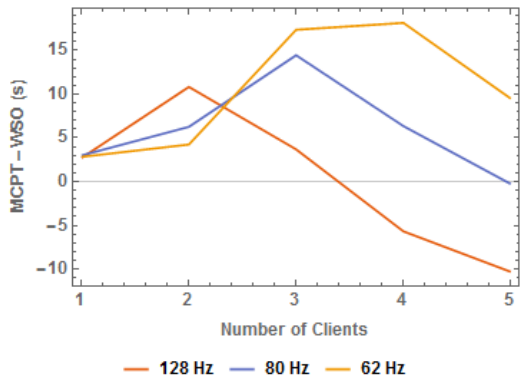
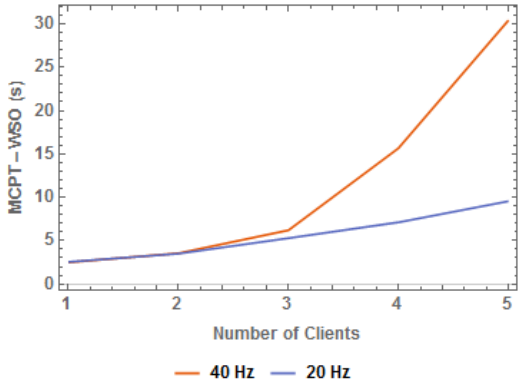


Figure 6.5: The change in WebSocket server overhead(WSO) as the number of simultaneous publishers increases.



(a)



(b)

Figure 6.6: The Δt between the MCPT and WSO as the number of simultaneous publishers increases.

1 client & 2 sensors

Streaming Rate	128Hz	80Hz	60Hz	40Hz	20Hz
Cpu usage	57-62%	40-45%	30-37%	15-25%	8-13%
Mean Chunk Processing Time	2.9s	2.85s	2.48s	2.46s	2.32s

2 clients with 2 sensors each

Streaming Rate	128Hz	80Hz	60Hz	40Hz	20Hz
Cpu usage	95-98%	94-98%	76-80%	47-56%	29-36%
Mean Chunk Processing Time	17.40s	11.70s	4.11s	3.18s	3.05s

3 clients with 2 sensors each

Streaming Rate	128Hz	80Hz	60Hz	40Hz	20Hz
Cpu usage	98-99%	97-99%	97-99%	75.84%	34-46%
Mean Chunk Processing Time	28.12s	27.12s	22.15	6.36s	5.40s

4 clients with 2 sensors each

Streaming Rate	128Hz	80Hz	60Hz	40Hz	20Hz
Cpu usage	99%	98-99%	98-99%	96-99%	50-62%
Mean Chunk Processing Time	38.83s	36.19s	36.98s	29.93s	7.67s

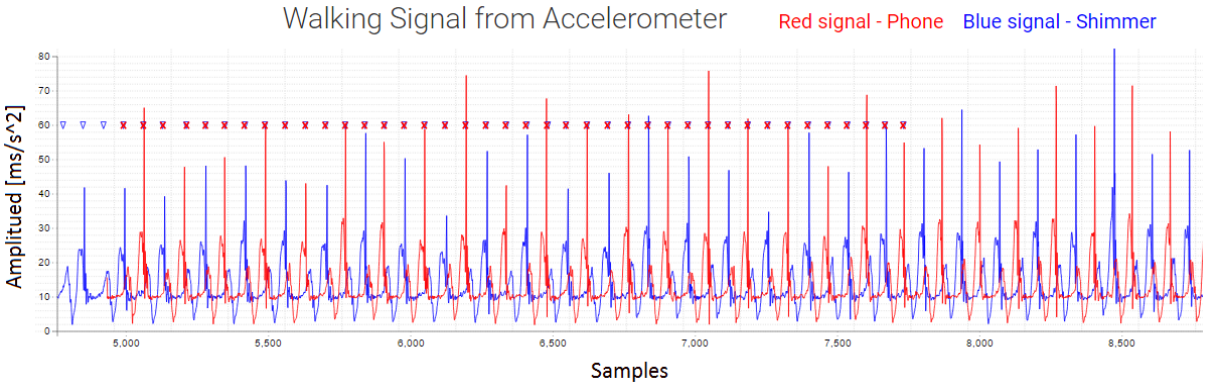
Memory usage, streaming rate 128Hz

Clients & Sensors	1Client & 2sensors	2Client & 2sensors	4Client & 2sensors
Maximum memory usage	1.5Gb	2.5Gb	5Gb

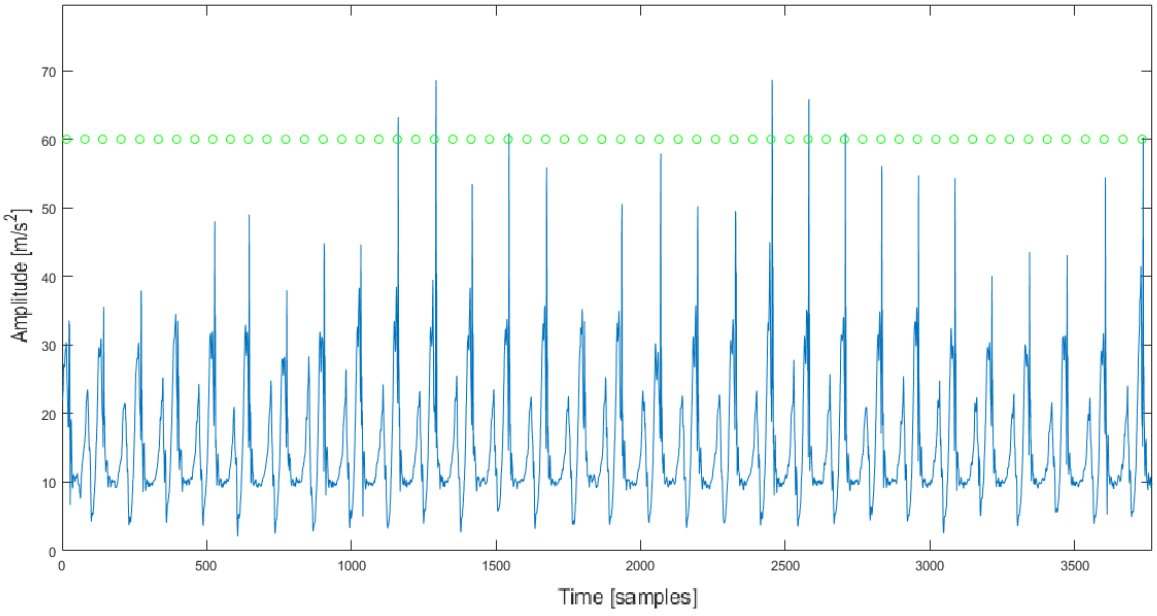
Table 6.1: CPU/RAM usage and the associated MCPT during the different experiments.

6.2 Graphs From the Dashboard

The following subsection shows the graphs displayed on the dashboard. Figure 6.7a which is a screenshot from the dashboard, is the walking signal with projected gait events, Figure 6.7b is the corresponding graph as displayed by MatLab. Figure 6.8 is showing plots for stride time, Figure 6.8a is from the dashboard and Figure 6.8b is from MatLab.

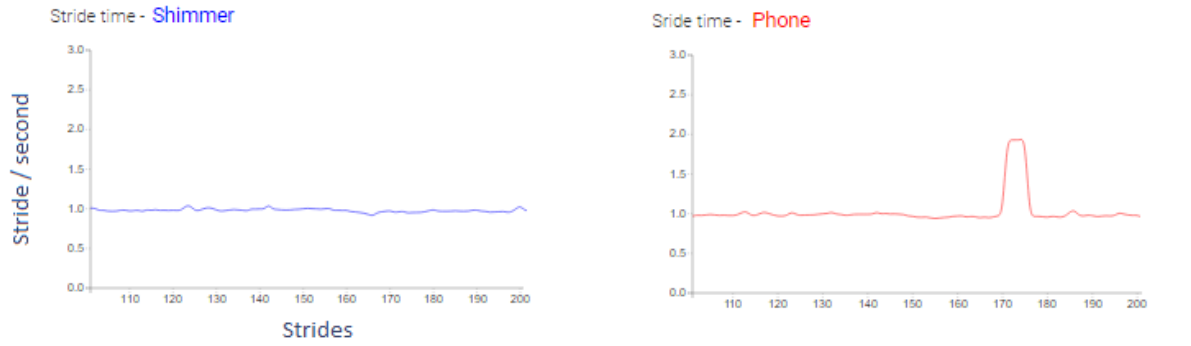


(a)

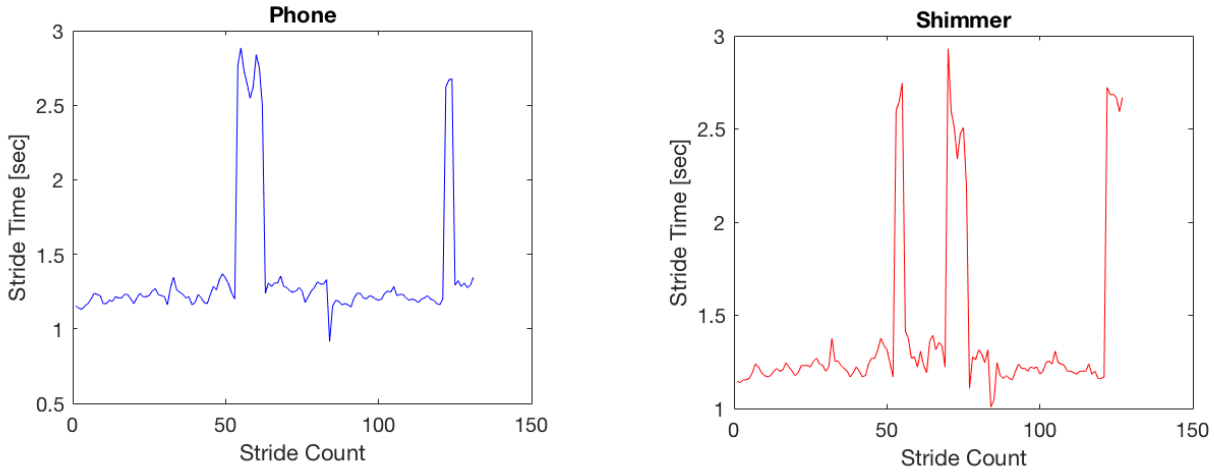


(b)

Figure 6.7: (a) A screenshot of the walking signal from the dashboard; the red part is the walking signal as collected by the phone on one leg; the blue part is the walking signal as collected by the medical sensor on the other. The blue triangles and the red crosses mark the detected gait events in the walking signal. In (b) a walking signal (one leg) with the detected gait events (green circles) is displayed; taken from a screenshot of the MatLab output. (a) and (b) are **not** produced using the same input data.



(a)



(b)

Figure 6.8: (a) is the display of stride time taken as a screenshot from the Dashboard. (b) is the corresponding plot, generated by MatLab. (a) and (b) are **not** produced using the same input data.

Chapter 7

Discussion

7.1 Performance and Performance at Scale

The limit of the system was 5 simultaneous publishers at a streaming rate of 128 Hz. The three areas in Figure 6.1: blue (Fill buffer time), green (MCPT) and orange (WSO) change as load increases; this is to be expected as more publishers compete over the same resources. It is believed that performance optimization can be thought of as optimizing for the smallest integral of the ARPT; the parameters under direct control are streaming rate and chunk size (during experiments the minimum chunk size *mc* was used).

Figure 6.3 shows that the integral of ARPT for the different streaming rates converges on similar fixed rates of increase, as the number of simultaneous publishers increases. Similar behaviour is seen when only the change in MCPT is observed (Figure 6.4); the rate of change is increasing to some point after which remains at a constant rate.

Working from the hypothesis that there in fact is some maximum rate of change which is resulting from increasing load on the system; the question of *when* the non-linear scaling properties (seen for instance at 20 Hz in Figure 6.3) resolves into the hypothesised maximum rate of change, is raised. Some clues are found in Figure 6.5 which shows that the WSO has a very sudden onset in every experiment. The point of onset shows to be related to an increase in the rate of change in both Figure 6.3 and 6.4.

The sudden onset of the WSO is believed to be related to the CPU usage reaching the 99th percentile; some indications of this is seen in Table 6.1 as there is a clear jump in MCPT at the same time as we see the onset of WSO in Figure 6.5.

Given that the hypothesis above holds, a try at finding the maximum number of simultaneous publishers for a given streaming rate, is made. If maximum CPU usage (99%) is to be avoided to maintain better scaling properties; and the WSO is strongly correlated with CPU usage; keeping the WSO under the corresponding limit is essential. That limit is believed to be found when the MCPT is equal to the WSO, forming the peaks seen in Figure 6.6 (a); The number of publishers at that peak would then be the maximum amount of simultaneous publishers for that streaming rate.

7.2 Scaling Up

As seen in the results, the current version of the platform responds poorly even at a small increase in the number of simultaneous clients; the task at hand is such that even small changes

in the number of clients have impact in terms of number of packets sent to the server (section 2.3). The following are suggestions in architectural improvements that could enable more simultaneous clients on the platform:

1. *Measurement bundle.* The reason for sending one measurement at a time was to improve the real-time qualities of the live signal. However, sending measurements in groups of 2 or even 3 may not have any visible effects on the live signal at all, while still improving the scaling factor by 2 - or 3x.

2. *Remote processing.* The p in Figure 4.2c is representing a gait analysis process on the same VM. Creating a dedicated instance for gait analysis may enable a more steady MCPT, as it would not be affected of the number of messages coming in from other publishers; the overall structure described in Figure 4.2c would remain, but p would now represent a remote API call to the dedicated gait instance.

3. *Load balancing.* Adding a load balancer to the system tuned to start a new instance at MCPT - WSO = 0 could keep the performance level stable over an increasing number of publishers.

4. *No MatLab support.* It may be that the MATLAB runtime as used in this project is not appropriate for use at scale. Whether or not (for instance) a python conversion of a gait analysis algorithm is actually more effective remains to be tested.

While RAM was a limiting factor when the platform was hosted on AWS it is seen from experiments that it was not the bottleneck in the tests performed on the Google Cloud instance, Table 6.1.

7.3 Real-time at the Endpoint

Real-time was a feature around which the platform was designed; it is the reason for opting for a communication protocol like WebSocket (section 2.3.2) on the client- as well as the server side; and also why there was an aim at keeping time of data transmission close to time of data caption. Quantitative measurements which includes not only the internal time delays of the system, but also the delay from the endpoint perspective (from the subscriber) remains for future work.

For addressing this task one might suggest that measuring the latency, for instance via round trip time delay, would offer a quantitative result reflecting the real-time capabilities of the platform from end to end; while this is true, the method was discarded for it would also measure a variable not under control i.e. network delay.

A better candidate might be to measure the jitter. Jitter is a measurement of disturbances in periodic phenomenon; this method could be applicable as the data sources of the system (the publishers) are sending packets with a period of $T = \frac{1}{f_s}$. An expected result from such a measurement would be for the jitter to be correlated with the WSO.

7.4 Critical Features for Commercial Viability

1. *Commercial licence for MatLab.* In this project a student licence for MatLab was used; for deployment in non-educational sectors a commercial licence must be purchased.

2. *Cloud server funding.* This project managed to stay within the free trial options of Google Cloud and AWS (appendix A). However, the instance running on Google cloud which was used under testing costs around $80 per month.

3. *Protection of integrity and sensitive data.* This topic is discussed in section 7.5.

7.5 Protection of Integrity and Sensitive Data

The Swedish Data Protection Authority[58] writes that according to *The Personal Data Act*[59], the following is to be considered sensitive information: "According to The Personal Data Act, sensitive data is any data that reveals race, ethnicity, political opinion, religious or philosophical values, memberships in organisations and personal information regrading *health*".

Since gait analysis can be used to draw conclusions regarding a person's health and may also allow for identification based on their way of walking, gait data is to be considered *sensitive*; the coming paragraphs will address existing vulnerabilities in the developed platform and offer some suggestions in terms of implementation of security features.

Developing a platform that deal with people's private information and especially intimate data regarding health, requires a rich set of security features. Potential vulnerabilities start at data being transmitted from the personal device to the platform; unauthorized parties must be prohibited from intercepting this communication channel; a method for achieving a secure channel is presented in the next subsection.

Once data has reached the cloud server it must be stored in a secure way. MongoDB, which was the database chosen in this project offer some security features. However, the full extent of their native functionality for secure data storage was never evaluated in this project. The sensitive data stored in the database would need to undergo encryption as to ensure its integrity in a scenario of compromised security on the platform. Whether or not MongoDB should remain the database of choice under future development will depend on its ability support encryption and secure connections.

The final point in this security discussion is about data retrieval. The developed platform allowed any party to subscribe to an output channel via the dashboard; future development should see to restrict this access to authorized parties only. Examples of authorized parties would be the client who is sending the data and maybe an assigned doctor or researcher to review the data. A method of authentication is presented in the next subsection.

7.5.1 Secure Communication Between Client and Server and Authentication

There were two main security concerns for the connection between the client and the server:

(i) *Authentication* - Who should be allowed to connect to the server?

(ii) *Secure data stream* - How could a channel be protected in a way as to ensure the integrity of the sensitive data?

(i) Socket.io does not have native support for client authentication. A method for authenticating a WebSocket connection is to use secure tokens. One implantation described by [60] involves the following steps: A client logs on to a server via a secure HTTP (HTTPS) connection; if the log in details are correct, the server responds with a token; the client then submits a request to connect to the WebSocket server, with the token included; the WebSocket server *validates*

the token and the client is dropped if the validation fails.

(ii) Section 11.1.2 of the WebSocket specification[29] states that WebSocket can by used together with TLS[61] thus protecting against attacks such as man in middle, where a third party intersects the communication and access the sensitive information.

7.6 Potential Economical Utility in Cloud Based Gait Analysis

Bringing gait analysis to the cloud comes with many challenges. As discussed in previous sections, fundamentals like ensuring integrity and security of sensitive information, as well as building economically sustainable systems which can meet high demand is a challenge. However, the incentives for confronting these challenges are large.

The reader is asked to consider the following scenario: A patient suffering from mobility impairment is located in a country which lacks the facilities and economical resources to offer the appropriate care at an accessible cost. The mobility impairment might stem from a physical injury or neurophysiological decease like Parkinson's; regardless of the cause, regular or even continuous monitoring of the patient might be crucial to ensure proper rehabilitation or increased quality of life.

To address this consider a cloud based medical platform that incorporates relevant knowledge in the form of specialized gait analysis algorithms, and means to draw relevant conclusions from the analysis. This platform would be accessible via the global span of the Internet. Using a large body of data and automated computer systems, serving large amounts of patients with little or no human supervision becomes a possibility.

On the patient side of the system the smartphone could be utilized. The density of smartphones in the population is large; the ability of these devices to collect and transmit data allows them to serve as an instrument sufficient for analysis, thus removing the need for medical grade sensors.

Cheap and accessible instruments combined with decreased levels of human supervision translates to an increased level of access to gait analysis at places where this would otherwise not be a possibility. The increased level of access stems from lowered economical strain on both society and individuals; society benefits in not having to invest in specialized facilities, like the gait labs discussed earlier in this report; in turn these potential savings could result in cheaper treatment for individuals.

Chapter 8

Conclusion

From section 1.3 **Goal**: "*Purpose a system design for a cloud based gait analysis platform focused on near real-time results. Implement and test the proposed system design. Evaluate the system in terms of performance and scalability.*".

The developed platform integrates a gait analysis algorithm in a cloud environment; it handles sensor data streamed to it at high rates and uses the incoming data as input for the gait analysis algorithm. The system enables multiple simultaneous clients and separate them in terms of input and output; clients can stream data from 2 sensors at a time. How output can be presented is demonstrated by the dashboard implementation.

The performance of the platform was evaluated using simulated clients and an integrated system for time logging.

Problem statement **1**:*"Can a cloud based gait analysis platform enable existing indoor systems for gait analysis to be applicable in an outdoor environment?"*

Via the integration of MatLab into the platform and the remote processing abilities of cloud computing, this project demonstrates how subjects of gait analysis can be spatially decoupled from the receiving components of their gait analysis systems; this given that they have means to transmit data from their sensor nodes to the cloud. Moreover, the platform shows promise in being able to handle a wider range of existing gait analysis systems due to its support for a generic MatLab script; this together with the modular system design which enables a tailored container for the script to be defined makes up the enabling features of this system.

The dashboard developed shows how the output from the native run time of the gait analysis algorithm could be mimicked in a web client (section **??** Figures: 6.7a, 6.7b, 6.8a and 6.8b). The dashboard has additional features of value as it not only removes the need for external software beyond a modern web browser; it also shows the walking signal in near real-time with results projected onto it.

Problem statement **2**:*"Can cloud based gait analysis platform enable real-time or near real-time results"*

This question has yet to be answered in full due to the lack of quantitative measurements of the real time aspect of the developed platform (see discussion in previous chapter). However, what can be stated thus far is that the result projection time, at times when the system is under the critical load will be largely composed of the time penalty imposed by the gait analysis

algorithm itself; this indicates that the system could host a real time gait analysis algorithm without adding much time penalty in terms of its own overhead.

Final Words

While work still remains in bringing down the scaling costs of the platform as to make it deployable on large scale; this project shows promise that cloud based, mobile, free-living gait analysis has potential to be functional.

Bibliography

[1] R. Williamson and B. Andrews. Gait event detection for fes using accelerometers and supervised machine learning. *Rehab. Eng., IEEE Trans*, 8(3):312–319, 2000.

[2] A. Salarian, H. Russmann, F. J. G. Vingerhoets, C. Dehollain, Y. Blanc, P. R. Burkhard, and K. Aminian. Gait assessment in parkinson's disease: toward an ambulatory system for long-term monitoring. *IEEE Transactions on Biomedical Engineering*, 51(8):1434–1443, Aug 2004. ISSN 0018-9294. doi: 10.1109/TBME.2004.827933.

[3] Janice Loudon, Marcie Swift, and Alexander Stephania Bell. *Clinical Orthopedic Assessment Guide - 2nd Edition*. Human Kinetics Publishers, 2008.

[4] Dustin A. Bruening and Sarah Trager Ridge. Automated event detection algorithms in pathological gait. *Gait Posture*, 39(1):472 – 477, 2014. ISSN 0966-6362. doi: http://dx.doi.org/10.1016/j.gaitpost.2013.08.023. URL `//www.sciencedirect.com/science/article/pii/S096663621300581X`.

[5] Oliver J. Woodman, C Oliver J. Woodman, and Oliver J. Woodman. An introduction to inertial navigation, 2007.

[6] Mitchell Yuwono, Steven W. Su, Ying Guo, Bruce D. Moulton, and Hung T. Nguyen. Unsupervised nonparametric method for gait analysis using a waist-worn inertial sensor. *Applied Soft Computing*, 14, Part A:72 – 80, 2014. ISSN 1568-4946. doi: http://dx.doi.org/10.1016/j.asoc.2013.07.027. URL `//www.sciencedirect.com/science/article/pii/S1568494613002822`. Special issue on hybrid intelligent methods for health technologies.

[7] Jan Rueterbories, Erika G. Spaich, Birgit Larsen, and Ole K. Andersen. Methods for gait event detection and analysis in ambulatory systems. *Medical Engineering Physics*, 32(6): 545 – 552, 2010. ISSN 1350-4533. doi: http://dx.doi.org/10.1016/j.medengphy.2010.03.007. URL `//www.sciencedirect.com/science/article/pii/S1350453310000718`.

[8] Di Pan, Rohit Dhall, Abraham Lieberman, and B. Diana Petitti. A mobile cloud-based parkinson's disease assessment system for home-based monitoring. *JMIR mHealth uHealth*, 3(1):e29, Mar 2015. doi: 10.2196/mhealth.3956. URL `http://mhealth.jmir.org/2015/1/e29/`.

[9] Hugo Hiden, Simon Woodman, Paul Watson, and Jacek Cala. Developing cloud applications using the e-science central platform. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 371(1983), 2012. ISSN 1364-503X. doi: 10.1098/rsta.2012.0085. URL `http://rsta.royalsocietypublishing.org/content/371/1983/20120085`.

[10] J. Martin. *Programming real-time computer systems*, page 4. Prentice-Hall series in automatic computation. Prentice-Hall, 1965. URL `https://books.google.se/books?id=LgpKodW460oC`.

[11] Aner Weiss, Sarvi Sharifi, Meir Plotnik, Jeroen P. P. van Vugt, Nir Giladi, and Jeffrey M. Hausdorff. Toward automated, at-home assessment of mobility among patients with parkinson disease, using a body-worn accelerometer. *Neurorehabilitation and Neural Repair*, 25(9):810–818, 2011. doi: 10.1177/1545968311424869. URL `http://dx.doi.org/10.1177/1545968311424869`.

[12] Silvia Del Din, Alan Godfrey, Brook Galna, Sue Lord, and Lynn Rochester. Free-living gait characteristics in ageing and parkinson's disease: impact of environment and ambulatory bout length. *Journal of NeuroEngineering and Rehabilitation*, 13(1):46, 2016. ISSN 1743-0003. doi: 10.1186/s12984-016-0154-5. URL `http://dx.doi.org/10.1186/s12984-016-0154-5`.

[13] S. Del Din, A. Hickey, S. Woodman, H. Hiden, R. Morris, P. Watson, K. Nazarpour, M. Catt, L. Rochester, and A. Godfrey. Accelerometer-based gait assessment: Pragmatic deployment on an international scale. In *2016 IEEE Statistical Signal Processing Workshop (SSP)*, pages 1–5, June 2016. doi: 10.1109/SSP.2016.7551794.

[14] Giancarlo Fortino, Daniele Parisi, Vincenzo Pirrone, and Giuseppe Di Fatta. Bodycloud: A saas approach for community body sensor networks. *Future Generation Computer Systems*, 35:62 – 79, 2014. ISSN 0167-739X. doi: http://doi.org/10.1016/j.future.2013.12.015. URL `http://www.sciencedirect.com/science/article/pii/S0167739X13002793`. Special Section: Integration of Cloud Computing and Body Sensor Networks; Guest Editors: Giancarlo Fortino and Mukaddim Pathan.

[15] Min Chen, Sergio Gonzalez, Athanasios Vasilakos, Huasong Cao, and Victor C. M. Leung. Body area networks: A survey. *Mobile Networks and Applications*, 16(2):171–193, 2011. ISSN 1572-8153. doi: 10.1007/s11036-010-0260-8. URL `http://dx.doi.org/10.1007/s11036-010-0260-8`.

[16] BTS GAITLAB — Integrated Motion Analysis Systems — BTS Bioengineering. `http://www.btsbioengineering.com/products/bts-gaitlab/`. Accessed: 2017-04-23.

[17] Anita Sant' Anna, Nicholas Wickström, Helene Eklund, Roland Zügner, and Roy Tranberg. *Assessment of Gait Symmetry and Gait Normality Using Inertial Sensors: In-Lab and In-Situ Evaluation*, pages 239–254. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38256-7. doi: 10.1007/978-3-642-38256-7_16. URL `http://dx.doi.org/10.1007/978-3-642-38256-7_16`.

[18] Begonya; Mendez-Zorrilla Amaia Muro-de-la Herran, Alvaro; Garcia-Zapirain. Gait analysis methods: An overview of wearable and non-wearable systems, highlighting clinical applications. *Sensors*, 14(2):3362–3394, 2014.

[19] Stacy J Morris Bamberg, Ari Y Benbasat, Donna Moxley Scarborough, David E Krebs, and Joseph A Paradiso. Gait analysis using a shoe-integrated wireless sensor system. *IEEE transactions on information technology in biomedicine*, 12(4):413–423, 2008.

[20] Philipp Leusmann, Christian Mollering, Lars Klack, Kai Kasugai, Martina Ziefle, and Bernhard Rumpe. Your floor knows where you are: sensing and acquisition of movement data. In *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*, volume 2, pages 61–66. IEEE, 2011.

[21] A. M. Howell, T. Kobayashi, H. A. Hayes, K. B. Foreman, and S. J. M. Bamberg. Kinetic gait analysis using a low-cost insole. *IEEE Transactions on Biomedical Engineering*, 60 (12):3284–3290, Dec 2013. ISSN 0018-9294. doi: 10.1109/TBME.2013.2250972.

[22] Siddhartha Khandelwal and Nicholas Wickström. Gait event detection in real-world environment for long-term applications : Incorporating domain knowledge into time-frequency analysis. *IEEE transactions on neural systems and rehabilitation engineering*, 24(12): 1363–1372, 2016.

[23] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[24] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[25] RFC 4251 - The Secure Shell (SSH) Protocol Architecture network working group of the ietf. `https://tools.ietf.org/html/rfc4251`. Accessed: 2017-03-07.

[26] Amazon Leads; Microsoft, IBM Google Chase; Others Trail synergy research group. `https://www.srgresearch.com/articles/amazon-leads-microsoft-ibm-google-chase-others-trail`. Accessed: 2017-03-07.

[27] AWS Free Tier Details - Amazon. `https://aws.amazon.com/free/`. Accessed: 2017-03-22.

[28] Amazon EC2 Instance Types amazon. `https://aws.amazon.com/ec2/instance-types/`. Accessed: 2017-03-07.

[29] RFC 6455 - The WebSocket Protocol rfc 6455 - the websocket protocol. `https://tools.ietf.org/html/rfc6455`. Accessed: 2017-03-04.

[30] RFC 793 - Transmission Control Protocol rfc 793 - transmission control protocol. `https://tools.ietf.org/html/rfc793`. Accessed: 2017-03-07.

[31] Handshaking (networking) - Wikipedia. `https://en.wikipedia.org/wiki/Handshaking`. Accessed: 2017-03-09.

[32] Frame (networking) - Wikipedia. `https://en.wikipedia.org/wiki/Frame_(networking)`. Accessed: 2017-03-09.

[33] V. Pimentel and B. G. Nickerson. Communicating and displaying real-time data with websocket. *IEEE Internet Computing*, 16(4):45–53, July 2012. ISSN 1089-7801. doi: 10.1109/MIC.2012.64.

[34] Node.js — node. `https://nodejs.org/en/`, . Accessed: 2017-02-20.

[35] S. Tilkov and S. Vinoski. Node.js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83, Nov 2010. ISSN 1089-7801. doi: 10.1109/MIC.2010.145.

[36] Node.js — SPAWN. `https://nodejs.org/api/child_process.html#child_process_child_process_spawn_command_args_options`, . Accessed: 2017-02-20.

[37] Node.js — FORK. `https://nodejs.org/api/child_process.html#child_process_child_process_fork_modulepath_args_options`, . Accessed: 2017-02-20.

[38] Introducing MATLAB Engine API for C/C++ and Fortran - MATLAB Simulink - MathWorks Nordic. https://se.mathworks.com/help/matlab/matlab_external/introducing-matlab-engine.html, . Accessed: 2017-03-07.

[39] MATLAB Engine API for Python - MATLAB Simulink - MathWorks Nordic. https://se.mathworks.com/help/matlab/matlab-engine-for-python.html, . Accessed: 2017-03-07.

[40] An introduction to microservices — Opensource.com. https://opensource.com/resources/what-are-microservices, . Accessed: 2017-02-20.

[41] Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST). http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Accessed: 2017-02-20.

[42] Introduction to Microservices — NGINX. https://www.nginx.com/blog/introduction-to-microservices/, . Accessed: 2017-02-20.

[43] Microservices? please, don't - dzone integration. https://dzone.com/articles/microservices-please-dont, . Accessed: 2017-02-20.

[44] Web framework rankings — HotFrameworks. https://hotframeworks.com/. Accessed: 2017-05-06.

[45] ASP.NET — The ASP.NET Site. https://www.asp.net/. Accessed: 2017-05-06.

[46] The Web framework for perfectionists with deadlines — Django. https://www.djangoproject.com/. Accessed: 2017-05-06.

[47] MQTT mqtt version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. Accessed: 2017-03-20.

[48] U. Hunkeler, H. L. Truong, and A. Stanford-Clark. Mqtt-s x2014; a publish/subscribe protocol for wireless sensor networks. In *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, pages 791–798, Jan 2008. doi: 10.1109/COMSWA.2008.4554519.

[49] Socket.io. https://socket.io/, . Accessed: 2017-03-20.

[50] What is mongoDB - mongoDB. https://www.mongodb.com/what-is-mongodb. Accessed: 2017-03-22.

[51] MySQL System Properties — db-engines. https://db-engines.com/en/system/MySQL. Accessed: 2017-05-06.

[52] PostgreSQL - About — PostgreSQL. https://www.postgresql.org/about/. Accessed: 2017-05-06.

[53] Shard Keys — MongoDB Manual 3.4. https://docs.mongodb.com/manual/core/sharding-shard-key/. Accessed: 2017-05-06.

[54] V. Ambriola and G. Tortora. *Advances in Software Engineering and Knowledge Engineering*. Series on Software Engineering and Knowledge Engineering. 1993. ISBN 9789814502573. URL https://books.google.se/books?id=k8bsCgAAQBAJ.

[55] Siddhartha Khandelwal and Nicholas Wickström. Evaluation of the performance of accelerometer-based gait event detection algorithms in different real-world scenarios using the {MAREA} gait database. *Gait Posture*, 51:84 – 90, 2017. ISSN 0966-6362. doi: http://dx.doi.org/10.1016/j.gaitpost.2016.09.023. URL //www.sciencedirect.com/science/article/pii/S0966636216305859.

[56] Light-bootstrap-dashboard - Creative-Tim. https://www.creative-tim.com/product/light-bootstrap-dashboard. Accessed: 2017-03-28.

[57] Data-Driven Documents - d3js.org. https://d3js.org/. Accessed: 2017-03-28.

[58] What is sensetive data? vad menas med känsliga personuppgifter? - datainspektionen. http://www.datainspektionen.se/fragor-och-svar/personuppgiftslagen/vad-menas-med-kansliga-personuppgifter/). Accessed: 2017-03-20.

[59] Personuppgiftslagen personuppgiftslag (1998:204) svensk författningssamling 1998:1998:204 t.o.m. sfs 2010:1969 - riksdagen. https://www.riksdagen.se/sv/dokument-lagar/dokument/svensk-forfattningssamling/personuppgiftslag-1998204_sfs-1998-204. Accessed: 2017-03-20.

[60] Token-based Authentication with Socket.IOLearn to implement token-based authentication using Socket.IO. https://auth0.com/blog/auth-with-socket-io/, . Accessed: 2017-03-22.

[61] RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2. https://tools.ietf.org/html/rfc5246. Accessed: 2017-03-22.

[62] Free tier Google Cloud Platform - Google. https://cloud.google.com/free/, . Accessed: 2017-03-22.

[63] CUSTOM MACHINE TYPES - Google. https://cloud.google.com/custom-machine-types/, . Accessed: 2017-03-22.

[64] Eric W. Weisstein. Little's Law. http://mathworld.wolfram.com/LittlesLaw.html. Accessed: 2017-01-27.

Appendices

Appendix A

Tested Cloud Hosting Options

When searching for the appropriate cloud provider, there were some requirements that had to be met for the proposed platform to run well. The VM had to be at least powerful enough to run the MatLab engine, which was the bridge from MatLab to python. MatLab specified requirements of at least 2GB of RAM, any Intel or AMD processor and typically 4-6GB of storage (depending on what tools would be utilized from MatLab). However, these requirements were all for the desktop version, no requirements specification was found for the MatLab engine.

A.1 Starting with AWS

The first VM instance that was tried was one of Amazon Web Services(AWS) free tier options, the T2.micro. It had 1GB of memory and an Intel Xeon processor. Experiments on this VM revealed that running the MatLab engine (for Python) required significantly less RAM than the desktop version did. Approximately, the MatLab engine would consume around 300-400Mb of RAM.

It was first when running more than one client simultaneously on the platform that the T2.micro proved too underpowered; performance stagnated and the system could not cope with the traffic.

A.2 Transition to Google Cloud

As mentioned in section 2.3.1, Google Cloud offered $300 for new users to spend on computational resources on their platform [62] . One of the benefits of using Google cloud was a feature called "custom machine types" [63] which lets users customize the amount of CPUs and gigabytes of RAM the virtual machine is allocated.

On Google Cloud an instance with higher specification could be used, as long as the $300 would last through the testing phase. It resulted in a customized virtual machine with 4 vCPUs, 10GB memory and an SSD disk with 30GB of storage. The VM was located in Belgium, chosen for its geographical proximity. The new more powerful instance allowed for testing at larger scale than was possible on AWS.

Appendix B

Client Server Communication Protocol

To connect to the server, clients had to adapt to either the publisher or subscriber protocol.

B.1 Publisher Protocol

1. **Connect to the server IP via WebSockets**
2. **Register as publisher (2 sensors)**
   websocket.emit('register', {user: 'John', type: 'publisher', channel: '1')})
   websocket.emit('register', {user: 'John', type: 'publisher', channel: '2')})
3. **Await event *'registered'* for each channel**
4. **Start streaming**

B.2 Subscriber Protocol

1. **Connect to the server IP via WebSockets**
2. **Register as subscriber (2 sensors)**
   websocket.emit('register', {user: 'Tim', type: 'publisher', socketid: ABC123, channel:'1')})
   websocket.emit('register', {user: 'Tim', type: 'publisher',socketid: ABC123, channel:'2')})
3. **Await data..**

Appendix C

Project plan

C.1 Introduction

Gait is a person's manner of walking.

Gait is centered around three main components: locomotion, balance and ability to adapt to the environment. In the human body this is achieved through a balance between various interacting neuronal and musculoskeletal systems[1]. Effects of dysfunction in any of these interacting systems would appear in gait, thus stating the importance of gait analysis. For instance, a neuro-physiological disease like Parkinson's disease (PD) will have measurable effects on gait even at an early stage in the disease's progression[2].

A gait cycle is described as the time between successive foot contacts of the same limb. In one gait cycle there are 2 steps and 1 stride. A step is defined as the time between the heel strike of one foot and the heel strike of the other. A stride is completed after two successive heel strikes on the same foot[3]. Two important events in a normal gait cycle are the heel strike and toe off. They are essential in estimating stride and step parameters and that is why being able to detect these events is of great importance in any gait analysis application(s).

Gait analysis is normally performed in clinical gait labs equipped with various sensing modalities. Labs equipped with motion capture systems and force plate equipment offer very rich data but suffers from not being able to monitor subjects during longer periods of time[4]. Recent advancements in MEMS tech[5], has significantly improved gait analysis by providing wearable sensing technologies and ambulatory systems[6]. These wearable technologies are generally based on inertial sensors such as accelerometers, gyroscopes and magnetometers.

Tests confined to labs and dependent on supervision are suffering from not being able to observe patients during longer periods of time in daily life; they are also expensive to perform. The efforts made into tackling both the problems of cost and observation time has yielded: (i) better algorithms for analyzing gait in different environments[7]; (ii) mobile applications which uses the embedded sensors of smartphones[8]; (iii) remote processing which makes gait analysis available for more people.

There is demand for a framework for performing gait analysis in real-world environments. Some general purpose platforms for performing scientific experiments such as e-Science central[9] have been proposed along the years. However, there remains a challenge regarding integrating applications reliant on a more rich runtime environment, which will typically be the case for gait analysis algorithms. One of the fundamental challenges in such a project entails getting these

algorithms to run in a cloud environment. Making a specialized platform for gait analysis will allow for a more tailored architecture and targeted optimization.

C.2 Related work

According to Weiss et al.[11] and Din et al.[12] performing gait analysis in highly controlled settings is suboptimal. This is said to be because of two main reasons. The first being the white-coat effect during clinical experiments causing a bias in the collected data itself. The second is that the subject is not observed during longer periods of time and hence there is lack of data for long-term gait analysis.

Din et al[12] writes: "Free-living gait is naturalistically dual task because of the distractions, environmental obstacles, and task complexities that limit attentional compensation; while conversely attentional control is optimised during scripted gait tests in the laboratory."

For anyone wanting to conduct gait analysis of subjects in daily life there are three main components that must be in place. First, the subject(s) must have access to a mobile sensor, for instance a wearable accelerometer. Second, there needs to be a way to collect the data from the subject and send it to the concerned party, the researcher. Third, the data must be analyzed by the concerned party and "the results should readily available to the subjects". While the first part, i.e. distributing the sensors to subjects for use at home, is a manageable task, the second and third is considered more problematic. As elaborated in [13], after the sensor has been worn at home by a subject, the data must be sent back to the researcher via some file sharing service like Dropbox™. The researcher then has to run the processing algorithm on each and every dataset received. In their example, a monitor period of a single patient for 7 days will result in 250 mb of data. This would then have to be processed, and would take around 20 minutes for each data set to finish. In this instance processing is done in MatLab™.

The e-Science central[9] aims at making it easier for researchers to "store, share and analyse their data, and for developers to create new scientic services and applications". The e-Science central is cloud based and allows for researchers to create workflows by combining applications that either already exists on the platform, or uploading their own applications to the platform. The service is interfaced via a web browser or API, although the API only lets you run a pre-existent workflow.

Din et al.[13] acknowledges that this type of platform has at least the potential to alleviate the task of large scale, unsupervised gait analysis experiments of subjects in their home environment, but goes on to highlight a critical problem: in order for the service to be useful it must be able to run what is typically algorithms too sophisticated in terms of libraries and dependencies to be easily deployed.

Moving away from the idea of a general purpose research platform, the work of Pan et. al. [8] introduces PD Dr. This work narrows the focus and concentrates on patients suffering from Parkinson's disease. Realizing the value in monitoring subjects or patients in their daily life, the work of Pan presents a platform for gait assessment that allows the patient to conduct tests at home using an app which they call PD Dr. The app presents the patient with a set of tests and instructions on how to go about in performing them. The app uses the embedded accelerometer of the phone to collect data. After a test is completed, the data is sent to a cloud server for processing. The resulting output is sent back to the client and can also be accessed by a doctor via a database. The cloud component of PD Dr is modular in design and is argued to scalable. But according to the authors, the system is not designed to perform long term continuous monitoring of subjects.

The approaches on the literature indicate that to get the better results from gait analysis, subjects need to be monitored in their natural environment and ideally also during longer periods of time. Understanding that studies on large scale becomes resource intensive due to the amount of manual labor needed there is demand for a platform to overcome these problems.

Making use of personal devices such as smartphones helps in further lowering of thresholds for large scale gait analysis, this is demonstrated with the mobile application and cloud service PD Dr. While PD Dr allows for 'at home' gait assessment it lacks the general purpose of the e-Science central and long term monitoring capabilities.

We see that in all of the above examples there is no real mention of optimization of delivering output with short delay. The idea of collecting data, analyzing and getting results back in real-time appears yet unexplored even though there might be applications of gait analysis which would benefit greatly from real-time feedback. Additionally, no solutions are presented for running existing algorithms in a cloud environment.

- Could the smartphone serve as a more general input device for gait analysis?
- Can a platform be designed as to support a typical gait analysis algorithm and also aimed at fast delivery of results?

C.3 Research Goal

Develop a cloud based framework for providing solutions for real-world gait analysis. The work will be centered around the server-side of the platform, which shall be evaluated in terms of performance and scalability.

C.3.1 Key Features of the Proposed Platform

The platform will consist of a cloud server, or at least a cloud component (there might be a need to run multiple servers to get the desired performance). There will also be a mobile application which can communicate with the cloud component. The mobile app which is using data primarily from its embedded accelerometer, will be the collector of data in the system.

The platform can perform Gait analysis on the data submitted to it from its clients. During development gait analysis will be done using an algorithm which performs detection of heel strike and toe off. The algorithm runs in MatLab.

The mobile app collects data and has the means to transmit that data to the server. Both the mobile and the server component is responsible for maintaining data integrity and ensure the quality of the output.

Interested parties given that they have some required level of access can subscribe to the output of a given process. A process here is a gait algorithm running for an input of data potentially streamed from a mobile app.

As a whole the platform handles the collecting, processing and presentation of the data and the corresponding processed data

C.4 Method

C.4.1 Approach

The development will take a bottom up approach. Beginning by developing a proof of concept prototype and improving from there. The project will work in sprints according to the agile model. After each iteration the quality of the platform will be measured. The assessment of its quality will be based on the key features mentioned in purpose.

A standard test for evaluating each version must be developed. The standard test must be developed in a way so it can be an objective measure of the progress made along each critical axis (scalability, reliability, performance, accessibility and presentation). The test after each version will be helping in answering the question on what to do next.

C.4.2 Verification

For verifying the server there are two main features of interest namely: *performance*, and *scalability*.

Performance relates to how fast results or components of desired results can be produced after a session has been initiated.

Scalability is the property which binds *performance*, to the number of clients or connections currently active on the platform i.e how the performance changes based on the number of users.

C.4.3 Experimental setup

Clients: Clients will be simulated to mimic the data output and the interface of the mobile app. The data will be based on MAERA Gait Database[55].

Server: The platform will be running on Ubuntu 16.04 LTS in the cloud. The hardware specifications is: Assumptions: Network is always available. The data stream will represent data sampled at 128 Hz.

C.4.4 Scalability

The scalability of the platform will be measured based on:
• Throughput: transactions per second (Little's law[64]).
• Response time: millisecond (ms) (end to end)

C.4.5 Performance

The performance in the mobile application will be measured by

• End to end response time: ms
• Response time per component: ms

C.5 Timeplan

The development will take form using Scrum, which is an agile development method. It is organized in sprints, which can last between 2 and 4 weeks. For this project, each sprint will take 1 or 2 week(s). First goal is to create a proof of concept due on 2nd February. During development and sprints, there will be time dedicated to documentation and writing on the report

Table C.1: Timeplan

Phases	**Duration**	**Date**
Sprint 1 - MVP	2 weeks	16 Jan - 30 Jan
Project plan seminarium	-	3 Feb
Sprint 2 - Improvements of MVP	2 weeks	6 Feb - 20 Feb
Sprint 3 - Quality iteration	2 weeks	20 Feb - 6 Mar
Sprint 4	2 weeks	6 Mar - 20 Mar
Midterm seminar	-	16 - 17 Mar
Sprint 5 - Last coding sprint	2 weeks	20 Mar - 3 Apr
Writing report	ongoing	3 Feb - 12 May
Final seminar	-	18 - 19 May
Individual examination	-	24 May
Utexpo, presentation of project	-	24 May
Last day for correlated report	-	4 Jun

UtExpo 2017

In UtExpo 2017 we plan on showcasing the working model of the proposed platform

Marcus Kärrman - Bachelor of science
in computer engineering

Hampus Carlsson - Bachelor of
science in computer engineering

PO Box 823, SE-301 18 Halmstad
Phone: +35 46 16 71 00
E-mail: registrator@hh.se
www.hh.se