Faculty of Engineering of the University of Porto

A Novel Machine Learning Algorithm for Classification of Gait Phases from EMG Data

Pedro Miguel Koch e Silva do Carmo

Dissertation
Master in Biomedical Engineering

Supervisor: João Manuel R. S. Tavares (PhD)

February 2019

Faculty of Engineering of the University of Porto

A Novel Machine Learning Algorithm for Classification of Gait Phases from EMG Data

Pedro Miguel Koch e Silva do Carmo

Dissertation
Master in Biomedical Engineering

Supervisor: João Manuel R. S. Tavares (PhD)

February 2019

Abstract

Gait event detection has been a fundamental part of gait analysis for several years, especially in Functional Electric Stimulation and Myoelectric prosthesis control. Recently, these areas have seen a rise in research regarding wearable systems using footswitches, Inertial Measurement Units (IMUs) and Electromyography (EMG). However, EMG has been consistently outperformed by other methods. The aim of this dissertation is to develop a robust EMG-based machine learning algorithm that can detect Initial Contact and Toe-Off events with a tolerance of 125ms, and rival the performance seen in IMU-based methods.

For this purpose, a novel Deep Learning Algorithm combining 2 Long Short Term Memory (LSTM) and 2 Dense Neural Network layers was developed using Tensorflow 1.13.1 and Python 3.7.3. EMG data from both left and right Gastrocnemius medialis was acquired from 15 healthy subjects at 2000Hz, resampled at 500Hz and low-pass filtered at 4Hz to create the EMG envelope (e-EMG). The envelopes were labelled as Stance or Swing according to their corresponding gait phase, calculated from ground reaction data obtained with force plates.

The labelled data was fed to the algorithm in sequences of 180 samples for training. The algorithm then classified the last sample in the sequences as “Stance” or “Swing”. IC was considered the transition from Swing to Stance and TO the transition from Stance to Swing.

The mean time delays between gait events and the algorithm’s detection were -4 (± 24)ms for IC and 6 (± 22)ms for TO. 100% of gait events were detected by the algorithm, although only 87% of detected events corresponded to real events. In addition, the time delays fell well within the 125ms tolerance window with over 99.9% confidence level. When comparing our algorithm with other state-of-the-art methods using EMG and IMUs, we found that our algorithm performed just as well, if not better, cementing the value of using LSTM for EMG classification.

Resumo

A deteção de eventos de marcha é uma parte fundamental da análise de marcha, especialmente para aplicações como Eletro-estimulação e controlo de próteses. Recentemente, na área de análise de marcha os sensores "wearable" como *floorswitches*; acelerómetros e giroscópios; e eletromiografia têm vindo a despertar maior interesse nos investigadores. No entanto, a eletromiografia costuma apresenta resultados piores para este tipo de análise. O objetivo desta dissertação é criar um algoritmo de *Machine Learning* com base em eletromiografia que consiga detetar eventos de *Initial Contact* (IC) e *Toe Off (TO)* com 125ms de tolerância, e rivalizar com métodos baseados em acelerómetros e giroscópios.

Para tal, foi criado um inovador algoritmo de Deep Learning, que combina duas camadas de Long Short Term Memory (LSTM) e duas camadas de Rede Neuronal Densa, com recurso a Tensorflow 1.13.1 e Python 3.7.3. Dados eletromiográficos do gastrocnémio medial esquerdo e direito foi adquirida a 2000Hz de 15 sujeitos saudáveis, reamostrados a 500Hz e filtrados a 4Hz para obter o envelope do sinal. Os envelopes foram anotados como "Swing" ou "Stance" de acordo com forças de reação de solo recolhidas com plataformas de força.

Os dados anotados foram introduzidos ao algoritmo em sequências de 180 amostras para treinar o algoritmo, que classificou a última amostra de cada sequência como "Swing" ou "Stance". O evento de IC foi considerado a transição de Swing para Stance, e o TO foi considerado a transição de Stance para Swing.

A diferença média em milissegundos entre o acontecimento de um evento e a sua deteção pelo algoritmo foi -4 (± 24)ms para IC e 6 (± 22)ms para TO. 100% dos eventos foram detetados pelo algoritmo, no entanto, apelas 87% dos eventos detetados pelo algoritmo corresponderam a eventos reais. Para além disso, o intervalo de previsão para as diferenças temporais com nível de confiança de 99% encontra-se dentro do intervalo de tolerância de 125ms. Comparando o nosso algoritmo com outros métodos de deteção de eventos de marcha usando eletromiografia e acelerómetros e giroscópios, concluímos que o nosso algoritmo teve uma performance tão boa como os restantes métodos, se não melhor. Como tal, esta dissertação serviu para solidificar o valor de LSTMs na deteção de eventos de marcha com recurso a dados eletromiográficos.

Contents

List of Figures

List of Tables

List of Acronyms

ADAM	Adaptive Moment Estimation
ANFIS	Adaptive neuro-fuzzy inference system
AR	Autoregressive coefficients
BIC	Bayesian information criteria
COG	Centre of Gravity
d-EMG	EMG Derivative
e-EMG	EMG Envelope
EMG	Electromyography
FA	Feet Adjacent
FD	Frequency domain
FES	Functional Electrical Stimulation
FF	Foot Flat
FSR	Force Sensing Resistors
GA	Genetic Algorithms
GD	Gradient Descent
GP	Gait Partitioning
HIST	EMG Histogram
HO	Heel Off
HS	Heel strike
IC	Initial Contact
i-EMG	Integrated EMG
Im-EMG	Intramuscular EMG
IMUs	Inertial measuring units
KEMG	Kinesiological Electromyography
LDA	Linear discriminant analysis
LSTM	Long Short Term Memory
MAV	Mean Absolute Value
mGAS	Gastrocnemius medialis
MNP	Mean Power
MPC	Motorised prosthesis control
MUAP	Motor unit action potential

NN	Neural Network
NWS	Non-Wearable Sensors
OIC	Opposite initial contact
PCA	Principal component analysis
PKF	Peak Frequency
PS	Pre-Swing
RF	Reaction force
RMS	Root Mean Square
RNN	Recurrent Neural Network
r-RMG	Raw EMG
SD	Standard Deviation
sEMG	Surface EMG
SSC	Slope Sign Changes
TA	Tibialis anterior
TD	Time domain
TO	Toe off
TS	Terminal Stance
TV	Tibia Vertical
WL	Waveform Length
WSS	Wearable sensors
ZC	Zero Crossing

Chapter 1

Introduction

1.1. Motivation

Gait analysis (GA) is the study of human gait. One of the fundamentals of GA is the human gait cycle, which splits human gait into a series of gait periods and gait events, which act as the boundary between the various periods. As a result, having systems that can automatically detect these periods and events is very beneficial for experts in this field. This process is called gait partitioning (GP), and has various applications in sports, rehabilitation, diagnostic settings [1].

In recent years, advances in gait analysis have provided many people with life changing technologies, like for example, functional electrical stimulation (FES) [2] [3], and motorised prosthesis control (MPC) [2], which can help people with certain disabilities to perform tasks which would otherwise be difficult or impossible to perform and improve their life quality. However, there is much work yet to be done in this area before we can fully eliminate disabilities.

For these technologies to work effectively, they must accurately keep track of the user's gait cycle in real-time, so they rely on GP methods. GP has for a long time relied mostly on expensive, heavy and static equipment, but these tools are not suitable for the applications stated above, since they need to accompany the user throughout their daily lives and activities. For these reasons, researchers have taken an interest to wearable systems, such as footswitches and foot pressure insoles (FPIs); inertial measurement units (IMUs); and electromyography (EMG) sensors to perform GP [4].

Footswitches and FPIs are very cheap and accurate sensors, but they lack versatility due to their simplicity. They have low durability, and suffer from a lot of problems, like being unable to distinguish walking from non-walking activities, or not always being suitable for pathological gait applications. Also, due to their simplicity, these kinds of sensors are the easiest to analyse, not having the need for complex algorithms to provide useful information [1], [2], [5].

IMUs are less accurate than footswitches but have a longer lifespan and are more versatile. They are capable of distinguishing walking and non-walking activities and are reliable in pathological scenarios. They are also more complex than footswitches and FPIs, often requiring the need for machine learning algorithms, but simpler rule-based algorithms can also be used to great effect with these sensors [1], [4], [6]–[8].

Finally, EMG sensors have the worst accuracy for detecting gait events and require complex machine learning algorithms to be able to perform but outperform other sensors at detecting gait onset and offset, intent recognition, proportional prosthesis control, error correction and robustness to pathological gait. In other words, EMG sensors trade accuracy and simplicity for versatility, which give them much potential for GA applications [3], [9]–[11].

As a result, EMG sensors have been widely adopted for many GP applications, but their dependence on complex ML algorithms and their relatively low accuracy is still a problem that hinders progress in this area.

1.2. Goals

The aim of this dissertation is to develop a new machine learning algorithm for GP with EMG sensors. The algorithm must meet the following requirements.

- Function in real-time;
- Detect gait events within 125ms;
- Be able to distinguish walking and non-walking activities, so no false events are detected;
- Be robust to different environments such as slopes, stairs, load changes, etc.;
- Be comfortable and not impair the user's gait.

In many applications, such as diagnosis, sports and surgery planning, real-time performance is not a necessity, as information retrieved can be processed and analysed at a different time, without impacting the quality of the diagnostic. Real-time performance is crucial, however, for applications that require an action to be performed at a specific instant in the gait cycle, such as FES and MPC [12].

The interval between the detection of a gait event and its real occurrence is called "detection delay", or simply delay. It has been shown that detection delays below 125ms are acceptable for MPC and FES, so the algorithm should detect events well-within this period [13], [14]. Delayed event detection leads to undesirable results and discomfort. In MPC, user of prosthesis with delays higher than 125ms reported sluggish behaviour of the prosthesis and showed overshooting problems and worse performance at performing simple tasks [13].

Being able to distinguish walking from non-walking activities is crucial in systems like FES and MPC. In these systems, an action is performed when a particular instance is detected by the algorithm. As a result, activities such as standing, sitting, shifting weight from one leg to the other, etc., should not be detected by the algorithm, since that would result in an undesired action, which could cause the patient to lose balance or even fall [6].

The algorithm must be robust to different environments, as the real world is constantly filled with slope changes and difficult terrain. A system that would only work on the user is walking on perfectly flat and level ground would not be suitable for real-world applications [15].

Lastly, the EMG system must be comfortable and not impair the user's gait [16], as that would have the opposite effect to the pretended one, which is to improve the user's comfort.

1.3. Dissertation structure

This dissertation is divided into 8 chapters: Introduction, Gait Analysis, State-of-the-art, Machine Learning, EMG, Methodology, Results and Discussion and Conclusion.

2 – Gait Analysis

Chapter two will begin with an overview of gait analysis' importance and various applications, and then will introduce the core fundamentals behind gait analysis, such as the gait cycle and the tools involved in gait analysis. After a brief introduction to the history of GA and the current rise of wearable systems, the concept of gait partitioning will be addressed, along with its applications in rehabilitation, sports and diagnosis. To finish off this chapter, the advantages of using wearable systems to perform GP are shown, especially for FES and MPC.

3 – State-of-the-art

Chapter three describes the state-of-the-art GP methods that rely on wearable systems. In particular, the advantages and disadvantages of footswitches and FPIs, IMUs and EMG. At the end of chapter three, it is concluded that EMG is a versatile technique with great potential for GP but has a lower accuracy than other sensors and needs complex signal processing and machine learning algorithms to function as desired.

4 – Machine Learning

Chapter four will focus on the fundamentals of machine learning, the different steps involved in using machine learning for classification problems, such feature selection, feature extraction and algorithm evaluation, and the advantages and disadvantages of popular ML algorithms.

5 – Electromyography

Chapter five will instead provide an overview of electromyography, namely a basic description of how the EMG signal is produced by the human body, the difficulties of signal acquisition due to EMG signal's random and noisy nature and the signal processing techniques used to try to mitigate said problems. After EMG signal is acquired and processed, features can be retrieved from the signal, making the original feature set.

6 – Methodology

This chapter discusses all the pre-processing techniques used throughout this dissertation and gives a brief overview of the developed deep learning algorithm and the statistical analysis performed on the algorithm's results.

7 – Results and Discussion

In this chapter, we present the results of the algorithm and discuss them in relation to our proposed goals. Afterwards, the algorithm is compared to other state-of-the-art gait event detection methods.

8 – Conclusion

The final chapter gives a final overview of the dissertation and suggests some possible pathways that could be applied in the future for better performance.

Chapter 2

Gait Analysis

In this chapter we will review some fundamentals about Gait Analysis, including a summary of the history of gait analysis, a few applications and the recent outburst of wearable systems in this area. Then we will introduce and explain the concept of the gait cycle, events and periods, and give a brief overview of the function of relevant muscles for human gait. Other concepts such as EMG signal properties, acquisition and processing; and basic Machine Learning concepts and algorithms will also be presented in this chapter. Finally, the state-of-the-art methods in gait event detection will be presented at the end of this chapter.

2.1. Introduction to gait analysis

From an evolutional standpoint, *Homo Sapiens* is the only obligate bipedal primate, and this locomotion method is reflected in many human anatomical features, especially in the lower limbs and foot, which have evolved to maximise the efficiency of bipedalism [17].

Bipedalism freed the upper limbs from a paradigm of power grasping to a paradigm of human-like precision grasping, which is required for tool handling, and therefore of paramount importance in human evolution [18].

Even so, walking is seen as a trivial and mundane task, but the mechanisms of walking are extremely complex and require coordination between several different body mechanisms. The process begins in the motor cortex in the brain, the cerebellum is responsible for coordination of the muscles, the spinal nerves regulate the tension generated by the muscle fibres through feedback received from Golgi tendons and other proprioceptive receptors. In order to achieve the desired motion, the muscles must generate the appropriate amount of tension, the joints must have the correct range of motion and the bones must be robust enough to withstand the forces involved. Any abnormality in any of these systems, from the motor cortex to the bones, will probably result in an abnormal gait [11]. Furthermore, issues in different systems will translate into different gait pathologies, so gait analysis can be used to identify the root of the problem and lead to a more efficient treatment. More so, gait analysis can be used to assess the success of a hip replacement surgery, or, in some cases, even evaluate if surgery is necessary. In other cases, it can be used to prematurely diagnose certain neurological condi-

tions that affect gait, such as Parkinson disease [1]. As a result, it comes as no surprise that human gait analysis has sparked an interest in many researchers working in rehabilitation, diagnostic and athletics fields [19].

2.2. Fundamentals of human gait

So far, this report has described why researchers have taken an interest in human gait, and the fruits of their labour. But what is human gait? How can researchers describe it and measure it? This section will attempt to answer those questions.

2.2.1. Gait cycle, gait events and gait periods

Human gait is a cyclic process where muscles, nerves and senses all act together in unison to achieve a common goal: Bipedal locomotion. To better understand the motions involved and facilitate analysis, human gait is divided into gait cycles, which get subdivided into different repetitive movements, called "gait periods". A gait cycle begins with the contact of one foot on the floor and ends when the same foot repeats that action [1], [4] and is usually divided into two phases - Stance and swing. The stance phase begins when the heel of the patient strikes the ground (heel strike -HS) and ends when the toe leave the ground (toe off – TO), and the swing phase begins when the foot leaves the ground and ends with the HS, repeating the cycle [1], [4], [15]. The problem with this approach is that it is dependent on two "gait events" to separate the gait phases, which becomes a problem when analysing pathological gait. Certain pathologies can cause a subject's heel to strike the ground later than usual or after a different part of the foot, which invalidates that event as a trigger to the stance phase [1]. The solution for this problem is to divide these two phases into various periods, which allow the technicians a higher degree of precision. Since each period is the interval between two events, using a more granular model (more periods) will result in a higher probability of finding gait events which are unaffected by the pathology being studied, but these models will also be harder to analyse [4]. One of the most common gait models was introduced by the Rancho Los Amigos committee, which divides the gait cycle into 8 different subphases [19]. However, there are models which divide the gait cycle into any number of phases ranging from 2 to 8, and some even express the gait cycle in a percentage value, so it is up to the technician to decide which model is best suited to study a given pathology [4]. Some examples of gait models are show in **Figure 2.1**.

Granularity	Gait Phases								
Two Phases	Stance					Swing			
Three Phases	First Rocker	Second Rocker				Swing			
Four Phases	Heel Strike	Flat Foot	Heel Off			Swing			
Five Phases	Heel Strike	Flat Foot	Heel Off	Toe Off		Swing			
Six Phases (a)	Initial Contact	Loading Response	Mid Stance	Terminal Stance	Pre Swing	Swing			
Six Phases (b)	Loading Response	Mid Stance	Terminal Stance	Pre Swing		Swing 1	Swing 2		
Seven Phases	Loading Response	Mid Stance	Terminal Stance	Pre Swing		Initial Swing	Mid Swing	Terminal Swing	
Eight Phases	Initial Contact	Loading Response	Mid Stance	Terminal Stance	Pre Swing	Initial Swing	Mid Swing	Terminal Swing	
Gait [%]	0					60			100

Figure 2.1 - Different Gait Cycle Models and their terminology from [4].

By analysing this figure, we can clearly see that there is no universal terminology in place, and often the same name applies to different concepts in different models. In order to avoid this confusion, we will use the terminology introduced in Vu et al. [15], shown in **Figure 2.2**.

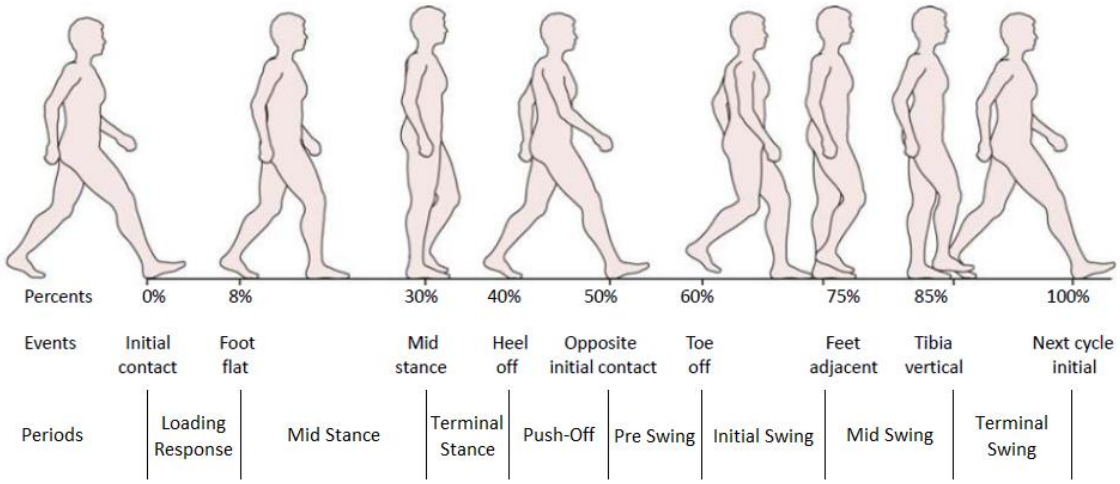

Figure 2.2 – Gait Cycle divided in % values, and gait periods, showing the relevant gait events. Adapted from [15].

Vu et al. considers a gait model with 8 separate periods of gait, divided by 8 gait events. The gait events and periods identified by Vu are described below.

Gait events are single instance, usually used to mark the transitions between two gait periods. The gait events considered in [15] are:

1. Initial Contact (IC): This is the first moment when the reference foot touches the floor (usually with the heel) and marks the beginning of the Stance phase. This moment also marks the beginning first "double-support" moment since the opposite leg is also in its stance phase;
2. Foot Flat (FF): This is the moment when the reference toe touches the ground, so the foot is now fully in contact with the floor;
3. Mid stance (event): This is the moment when the centre of gravity (COG) of the patient is overtakes the heel of the reference foot;
4. Heel Off (HO): The moment when the reference foot's heel leaves the ground;
5. Opposite initial contact (OIC): The moment when the Opposite foot contacts the ground, marks beginning of the second "double-support" moment;
6. Toe Off (TO): The moment when the reference foot completely leaves the floor. This moment marks the end of the Stance phase and beginning of the Swing phase;
7. Feet Adjacent (FA): The moment when the reference foot gets ahead of the opposite foot;
8. Tibia Vertical (TV): The moment when the reference tibia is in a vertical position.

Gait Periods are the periods in between two gait events. The gait events considered in [15] are:

1. Loading Response: The period between IC and FF, in this period the reference leg is loaded with the patient's weight, hence the name. In typical gait, the knee is slightly flexed to absorb shock, and the heel is used as a rocker;

2. Mid Stance (period): Not to be confused with Mid Stance (event). While the foot is stationary on the floor (from FF to Mid Stance (event)), the body's weight is shifted forwards, and the reference limb advances in relation to the foot;
3. Terminal Stance (TS): Similar to Mid Stance, but happens from Mid Stance (event) to HO. The body's COG is pushed forward, getting ahead of the reference foot;
4. Push-Off: As the body's COG overtakes the reference foot, the heel rises and the body rocks forward. The beginning of the second double-support moment marks the end of this period;
5. Pre-Swing (PS): Similar to TS but occurs after OIC. This is the second double-support period and lasts until the toe of the reference foot leaves the floor;
6. Initial Swing: Occurs between TO and FA, this is the first period of the Swing phase. To lift the reference foot of the ground, the hip and knee flex;
7. Mid Swing: Begins when both feet are adjacent. The hip reaches its maximum flexion during this phase;
8. Terminal Swing: After TV, the limb prepares for Loading Response. The hip and knee extend but the knee remains slightly flexed.

With this terminology, it is now possible to discuss different gait models and their uses in clinical settings. Please keep in mind that when discussing different models, the gait periods will not always match with this model. In these cases, we will define the gait periods as the moments between two gait events. It is also important to keep in mind that this model cannot be applied to certain pathologies which require their own gait models, but it can be applied to some abnormal gait cycles, such as Trans-tibial and Trans-femoral amputees' gait.

2.2.2. Gait muscle activity

Walking is an extremely complex activity, which involves the coordinated activation and coordination of dozens of muscles. To put this matter into perspective, Vaughan, Davis and O'Connor [19] provide us with a graph of 28 of the main muscles involved in human gait plotting their activation levels against the percentage of the gait cycle (**Figure 2.3**). Meanwhile, the same process, shifted by half a cycle, is repeating on the other leg. This type of analysis is important because it gives insight into the roles of muscles during the gait cycle and shows that muscle activity occurs in a cyclic, semi-predictable nature.

2.3. Gait analysis history and methodology

Section 2.2 described the fundamentals of human gait, namely the gait cycle, gait parameters and gait muscle activity. In short, it describes "what" gait analysis studies. This chapter will describe "how" researchers perform gait analysis, how the methodology evolved throughout the years and the real-world applications of this science.

2.3.1. Gait analysis: how does it work?

There are three main schools of thought in GA, but we must consider that these areas are not independent of each other and should be used together by clinicians to achieve the desired results. These schools of thought are kinematics, kinetics and muscle activity quantification.

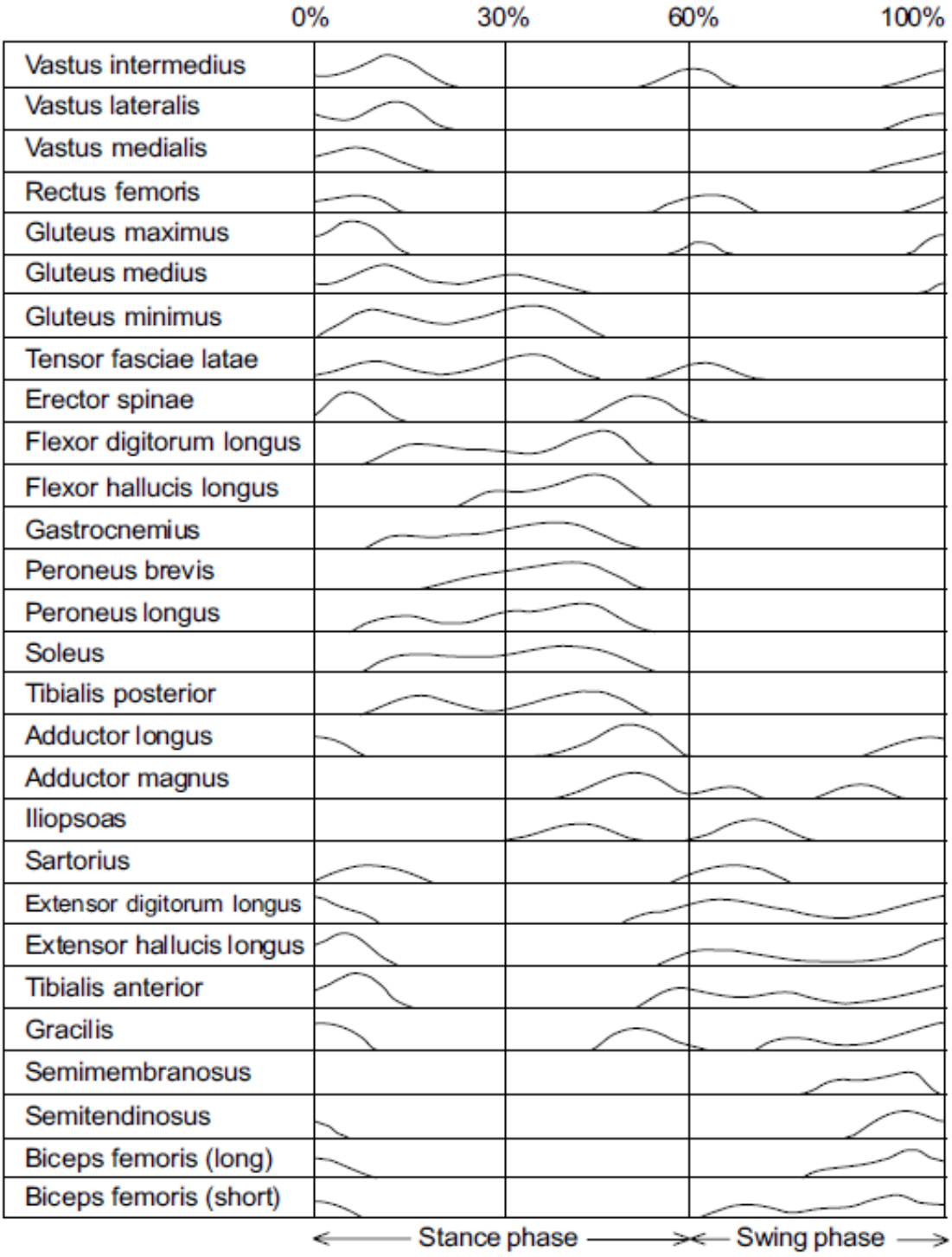

Figure 2.3 - Muscles involved in human gait and their respective activation levels. From [19].

Kinematics

Kinematics is concerned with studying the different movements involved in human gait. Usually, the subject's body is divided into different segments, and each segment's movement is tracked by a system of cameras. Markers are placed on specific body locations to facilitate the tracking of the segments. Then, a 3D rendering of the subject's movement is obtained and placed in a coordinate system in order to be studied. This allows the user to gain knowledge about joint angles, joint angular velocities and joint accelerations, as well as the

positions, orientations and velocities of different body segments [16]. Nowadays, systems like VICON automate a great part of the process, making it possible to collect measurements just minutes after the arrival of the subject, but this was not always the case [16].

Gait kinematics has many uses in the athletics and clinical fields. For example, Robert M. Kay et. al. used a VICON system to find that when kinematic GA was performed in preoperative orthopaedic patients, experts would change their recommended treatment plan in 90% of the cases, and in 40% of cases the recommended procedures were deemed unnecessary after GA [20]. In addition, DeLuca Pa combined video data with EMG and force plate data to find that preoperative GA reduced the cost of surgery in patients with cerebral palsy [21].

Kinetics

In order to walk, muscle fibres must pull on tendons, which in turn exert tor-ques on bone structures, which produce movement in the necessary body parts. With kinematics, only this final movement layer can be studied, but it is much more useful for a clinician to understand the underlying biomechanical forces responsible for the studied movements [22]. In fact, this was always one of the greatest goals of the early pioneers of GA [22]. This is where kinetics comes in, as it studies the forces and torques responsible for human gait. However, we cannot directly measure the forces from muscle fibres or tendons acting on the internal joints, so we must rely on indirect measurements [19]. According to Newton's third law: "If you press a stone with your finger, the finger is also pressed by the stone." [23]. In the same fashion, when we walk, we produce forces on the ground and the ground responds with an equal, but opposite force, which is commonly called a "reaction force" (RF). Researchers have tried to measure these RFs since 1872, when M. Carlet developed a mechanism that could measure the forces applied on the foot. However, this primordial device only provided one-dimensional information, so much work was still needed to obtain the full three-dimensional RF [22]. Researchers in Germany were able to use kinematic data to estimate the three-dimensional RFs but did not yet have the necessary tools to measure them directly [22].

Finally, in 1916, Jules Amar developed the world's first device capable of measuring RF in all three dimensions, vastly improving on previous designs [22]. This was the early foundation of the modern force plate, the most powerful tool in a gait analyst's arsenal and an indispensable tool in any GA laboratory. Amar's design was gradually improved on, achieving better accuracy, reliability and simplicity. So, there is finally a system to reliably measure the ground forces, but what exactly can be achieved with this data? In the early days, obtaining the 3D ground forces was the main goal of investigators, although studying these external forces still does not shed light on the internal forces acting on the individual joints. For that, complex moment calculations were necessary so the limitations in computational power hindered progress in this area [22]. With advances in computational power, inverse dynamics approaches were used to successfully calculate the net moments and forces applied in the joints of the subject. Vaughan, Davis and O'Connor [19], describe this process in detail in [19]. Once the net internal forces and momentums of the joints are known, they can be applied in a variety of scenarios. From the designing of special braces to reduce the joint moments acting on certain joints to treat disorders like coxarthrosis [24], or for assessing normal or pathological gait [25].

It is important to understand that knowing the joint forces is not the same as knowing the individual muscles forces acting on said joints. There are often reciprocal forces acting on

joints, where different muscles are simultaneously exerting abduction and adduction forces on the same joint in order to lock a joint in place [16], [19].

Muscle activity quantification

We have seen how we can apply kinetics and kinematics to get an insight into the mechanisms of movement. However, these methods cannot show us the muscle activity of individual muscles. Kinesiological Electromyography (KEMG), on the other hand, uses surface or intramuscular electrodes to study the function of individual muscles and their effect on joint movement and gait cycle. This technique works by measuring motor action potentials produced by muscle tissue, but it is important to remember that this measurement is not a direct measurement of muscle tension

The first recording of human electromyographic activity during voluntary contraction was performed by Emil du Bois-Reymond in 1849, sparking the age of EMG. However, it wasn't until the 1940s that this technology started being used to study dynamic movement [26]. In the 1960s, EMG was already being used to treat a variety of specific disorders in clinical settings. Hardyck [27] recorded laryngeal muscles activity to detect subvocalization while reading. If subvocalization was detected, the subjects were given an auditory stimulus. As a result, subjects quickly stopped subvocalizing, which had a positive impact on their reading development. In the following decades, EMG was used as a way to plan surgical interventions in children with cerebral palsy, such as tendon transfers [28], [29]. Muscles which showed continuous activity during walking had their tendons lengthened, which decreases muscle stiffness and spasticity, resulting in an improved control by the patient [29].

Nowadays, EMG provides a fast, cheap and non-invasive way to simultaneously monitor multiple muscle activities, and this allows practitioners to visualize hidden synergies between muscles, especially during complex activities such as walking or running [19], [27], which in turns provides the practitioner with a much more intuitive sense of the role of each muscle during a specific activity. Although EMG does not directly provide a measurement of muscle contraction force, there have been studies that attempt to estimate muscle force from EMG signal [30].

2.3.2. Wearable Systems in Gait Analysis

We have talked about the early methods utilized in GA, and their shortcomings. It is a common theme that many GA approaches rely on bulky, expensive and immobile equipment. Force platforms and Image Processing Systems use what is commonly called "non-Wearable Sensors" (NWS). NWS test are performed in controlled environments such as laboratories where the equipment is permanently set. The advantage of these methods is that they are less vulnerable to random variability in their data acquisition, since they are less vulnerable to external variables, and as such achieve higher accuracies. On the other hand, these methods are more expensive than their wearable counterparts, since they need to be carefully installed and calibrated and usually require expensive equipment and trained technicians to operate. Another disadvantage of these methods is that subjects cannot be regularly monitored performing their day-to-day activities, but only during short sporadic tests, which may not be representative of the subject's true condition, and some movement restrictions are applied to the subject. Even so, the gold standards in GA are still NWSs for their superior fidelity [1], [4].

Currently, the focus is starting to shift towards light, wearable sensors (WSs). This will mean that laboratories do not need to invest such large sums of money into expensive equipment, and setup and calibration routines will be simplified. But the biggest advantage of WSs is that they allow measurements to be taken in a larger array of scenarios and environments, including every-day activities where a clinician cannot be present. Some wearable sensors can even store data to later be reviewed by clinicians [1], [4].

For example, accelerometers and gyroscopes can be attached to the patient's body to perform kinematic measurements, by tracking both linear and angular accelerations of different body parts, respectively. In addition, magnetoresistive sensors can be used to calculate the 3D orientation of a body segment. These three sensors are typically combined into inertial measuring units (IMUs), which are more accurate than the individual sensors. Goniometers can be used to determine angles between body segments and subsequently the angular velocity of joints such as the knee [1], [4].

As for kinetics, researchers regularly use footswitches and pressure insoles. Footswitches are very basic pressure sensitive devices, which are placed on the sole of the foot, often in pairs or groups. They can be used to accurately detect foot contacts with the ground, but they suffer from a short lifespan and are unable to provide information about ground RF. Foot pressure insoles behave like footswitches, except they record information about the entire foot surface in contact with the floor. This allows them to measure more useful information such as vertical ground RF [4], [31]. Some foot pressure insoles have tried to measure 3D GRFs, although with limited success. For this reason, to perform the full measurement of 3D ground RF, force platforms are still the only technology available, so they are still required for certain applications such as calculating the internal joint forces and momentums.

Muscle activity quantification systems already rely on EMG electrodes, which are a wearable technology. Chapter 4 will provide a brief overview of EMG.

As with any technology, WSs also have disadvantages in a GA setting. WSs are less accurate than their non-wearable counterparts; require the use of batteries, which can restrict their performance; and are more prone to interference and noise. Also, WSs biggest strength of being able to be used outside of a laboratory setting can also be seen as a weakness, since it is not possible to check if the data obtained in an uncontrolled environment was affected by external factors [1], [4]. As such, it is up to the researchers to analyse each situation carefully and use decide which kinds of sensors should be used. This will depend greatly on the parameters being analysed and the error which the researchers are willing to tolerate.

2.4. Introduction to Gait Partitioning

So far, this chapter has introduced the basics of the gait cycle, and the basics of gait analysis. By using gait analysis techniques to study the gait cycle, we introduce the concept of GP. This section will provide a brief explanation of GP applications and methodologies.

As we know, the gait cycle is divided into periods, which are determined as the intervals between specific gait events. GP is the process of accurately determining gait events and periods by analysing kinetic, kinematic and muscle activity quantification data. With accurate GP, researchers can [1], [3], [4], [15], [30], [32]–[34]:

- Evaluate gait recovery of patients after surgical interventions or physiotherapeutic treatments;
- Classify various daily activities;

- Perform early diagnosis and track the progress of neurodegenerative or systemic diseases which impact gait;
- Assist in professional coaching;
- Electrically stimulate muscles (FES) in a controlled manner to assist patients with pathological gait;
- Assist and monitor the recovery of stroke patients;
- Develop control strategies for powered prosthetics to assist in the recovery of lower limb mobility.

GP can be performed with a variety of sensors, both wearable and non-wearable. Each sensor has its own strengths and weaknesses. For example, force platforms are the gold standard for detecting IC and HO events [4], [35]–[37], because of their accuracy, but cannot be used to detect other events [4]. Video systems such as VICON can be used to detect all 8 gait events [3], [38]with great accuracy, and are seen as the gold standard for high granularity models, but their use is limited to laboratories and cannot be implemented for daily applications [4]. For example, FES and prosthetics control are two applications that are must be active throughout the patient's daily life, so cannot rely on NWS. As a result, researchers have tried to use wearable sensors, already in use for other gait analysis purposes, to perform GP. The next chapter will present a brief overview of the state-of-the-art methods in GP with wearable sensors.

Chapter 3

State-of-the-Art

It is difficult to compare GP methods since there is not a standardised metric to evaluate these methods. Some papers use accuracy of gait classification, others use time delay between the tested method and a control in milliseconds, and some even express this difference in percentage of the gait cycle [15]. Depending on the problem at hand, different applications can be more suitable than others. For example, for control of powered prosthetics, online classification of data is necessary, and time delays for detection of IC and TO must not exceed 125ms [13]. In other applications such as diagnosis, offline classification can be enough to provide useful information [39] although more granularity may be required [40]. In this chapter, we will provide a brief overview of some of the main techniques used for GP, namely footswitches, IMUs and EMG, and discuss their advantages and disadvantages. This chapter will focus mainly on IC and TO detection delays, in order to have a fair comparison between the different methods.

3.1. Gait partitioning methods

Section 2.3.2 gives a brief description of wearable sensors commonly used in GP, in particular footswitches, pressure insoles, IMUs, accelerometers and gyroscopes, and EMG. This section will describe how these sensors are applied to GP problems, and present their advantages and disadvantages, which will then be compared in **Section 3.2.**

3.1.1. Footswitches

Many researchers showed that footswitches can detect IC and TO almost as accurately as force platforms [2], [4], [35]. Although they are unable to identify periods in the swing phase, some studies using footswitches achieved granularities as high as six gait periods, although this resulted in lower accuracies than studies using lower granularity models [41].

Balbinot [5], describes the use of two low-cost piezoelectric footswitch sensors located at the heel and toe to detect IC and TO with delays of (16±1) ms and (20±9) ms respectively, in relation to force plates, which is consistent with literature values [12].

Cherelle [2], used a 4-period model to control a powered lower limb prosthesis. Their system used two Force Sensing Resistors (FSR), one placed at the “heel” and another at the

"toe" of the prosthesis, to detect 4 gait events: IC, FF, HO and TO. Their method divided the gait cycle into 3 stance periods: IC to FF, FF to HO and HO to TO; and 1 swing period: TO to IC. The footswitches had an active and inactive state, indicated by a pressure threshold. By checking the active state of the heel and toe resistors, it is possible to know in which period the patient is currently and control the prosthesis accordingly. For example, in the IC to FF period, an actuator contracts a spring, which is then released on the push-off period, from HO to TO.

In conclusion, footswitches are very cheap and accurate for low granularities. Besides, they require very simple algorithms to perform GP, in comparison to other techniques [5]. In some cases [2], [42], a basic threshold level is sufficient to achieve adequate results. However, footswitches are unable to reach granularities higher than 6 gait periods, and, most importantly for applications like FES and prosthetics control, they are sensitive to disturbance by non-walking activities. Activities like sitting, standing, shifting weight from one leg to the other, sliding feet, etc., could trigger the sensors, leading to unwanted consequences [6]. Furthermore, their low durability is another known problem of these sensors and their success with pathological gait is limited [8], [43]. Foot pressure insoles work with the same principles as footswitches, so they suffer from the same problems [4], although they avoid the additional problem of sensor positioning, which is difficult on patients with abnormal gait and can lead to poor results [12], [43]. Overall, Footswitches and FPIs have good accuracy for GP, but are limited in their applications due to their disadvantages.

3.1.2. Inertial measurement units

IMUs consist of accelerometers, gyroscopes and sometimes magnetometers, which when combined provide information about the IMUs', translation and rotation on three axes, which in turn, provides information about the body part the IMU is attached to. Unlike footswitches and FPIs, IMUs can detect periods of the swing phase of the gait cycle, and thus can be used to identify all 8 phases of gait [4]. IMUs are also more forgiving in their placement, when compared to footswitches and EMG sensors [6], [15].

Maqbool et al. [44] used a single shank-mounted inertial measurement unit (IMU) consisting of a 3D accelerometer and gyroscope to detect IC, FF, HO and TO. The data from the IMU was fed to a rule-based algorithm, and gait event detection delay was measured in relation to footswitch sensors. Results for transfemoral amputees are shown in **Table 3.1.**

Table 3.1 – Gait event comparison between Maqbool et al, Mannini et al. and Müller et al. [6], [45], [46]. Timing delays are presented in average delay ± SD, in milliseconds. Negative values mean the event was detected by the IMU system before the reference system.

Gait Event	Time Delay ± SD (ms)		
	Maqbool et al.	Mannini et al.	Müller et al.
IC	21.8 ± 20	62 ± 47	100 ± 50
Floot Flat	-153 ± 103	-3 ± 53	-
HO	114 ± 60	86 ± 61	-
TO	-7.5 ± 15.5	36 ± 18	50 ± 70

Mannini et al. [6] utilized a single IMU attached to the foot to detect IC, FF, HO and TO. For this study, only acceleration data from the IMU's gyroscope was fed to a Hidden Markov Model algorithm. Event detection delays in relation to a reference VICON system are presented in **Table 3.1**. This study also reported that exact sensor placement is not necessary to achieve better results and that the algorithm was robust to weight shifting and load changes.

Finally, Müller et al. [45] used a single IMU, containing 3D accelerometers and gyroscopes to the instep of a shoe. Müller was able to detect 4 gait phases using a rule-based algorithm, but only reported timing delays for IC and TO, which are shown in **Table 3.1**. Like Mannini et al., Müller et al. found that the IMU provided satisfying results, even without exact placement.

In conclusion, IMUs have slightly worse accuracy than footswitches, but overcome many of their disadvantages, by being durable, easy to mount on different body parts and able to detect all phases of gait [15].

3.1.3. Electromyography

EMG sensors, like IMUs, can be used to detect all 8 gait periods, but have lower accuracy than other GP sensors. Nevertheless, these sensors have plenty of advantages which make them suitable for GP and so have been widely used for this purpose, especially in FES and MPC applications [3], [30], [38], [46]. There is a lot of variability in classification studies using EMG. Electrodes can be placed in different muscles, the features selected for training the algorithms can be different, as can the actual ML algorithms used. For example, Bayesian Information Criteria [38], Linear Discriminant Analysis (LDA) [47], Artificial Neural Networks (NN) [48], Fuzzy Inference Systems [3], [33], Stacked Sparse Autoencoders [47], and Hidden Markov Models [49], have all been used to perform classification problems with EMG signal.

Nazmi et al. [48] used EMG signal from tibialis anterior (TA) and gastrocnemius medialis (mGas) to detect IC and TO events in healthy gait. The acquired EMG features – RMS, SD, MAV, IEMG, and WL – were fed to an ANN comprised of three layers with one input neuron for each feature, two output neurons for gait phase classification and 10 neurons in the hidden layer. Footswitch data was used as the control to measure the delay of the ANN classification. IC and HO events were correctly classified 87.5% of the time. Of these 87.5% occurrences, the average time differences between ANN and footswitch data was 35 ± 25 ms for IC and 49 ± 15 ms for TO. Although the study showed promising results, the testing dataset consisted of only one patient, only two muscles were analysed, and the authors admit the use of footswitch sensors is not the most robust method for detecting IC and TO. Having data from more muscles would increase the dimensionality of the classification problem, but it could also lead to more reliable classification, and according to the authors, the use of force plates to detect IC and TO events could also lead to more robust results.

Joshi et al. [38] used EMG signal from the mGAS, TA, bicep femoris and quadriceps femoris to classify the normal gait cycle into 7 phases. This study used MAV, WL, variance and 4 AR coefficients to train an algorithm which used a combination of Bayesian information criteria (BIC) for feature selection and LDA for classification. Unfortunately, the study does not show the average time differences of individual gait event detection between the EMG classification technique and the VICON based technique used as the control, but it achieved an overall classification accuracy of 76 ± 13 ms across the whole gait cycle. Although these results do not look too promising compared to others in literature, it's important to keep in mind that

classifying 7 separate periods of gait is a more demanding problem than classifying only 2 periods.

Finally, Lauer et al. [3] used an adaptive neuro-fuzzy inference system (ANFIS) to detect seven gait periods in seven children with cerebral palsy. EMG signal was acquired from the left and right vastus lateralis muscles. Acquired signal was sampled at 1.2 kHz and bandwidth filtered from 20 to 350 Hz. From this signal, only raw values from both muscles and their respective derivatives were fed to the ANFIS algorithm. Timing delays were measured in relation to a VICON system and are shown in **Table 3.2**. An interesting characteristic of this study was that the evaluation of the algorithm was performed a second time, with the same children, two months after the original evaluation, to check if the repeatability of the algorithm over time. These results are also shown in **Table 3.2**.

Table 3.2 – Timing delays between ANFIS and reference VICON system. 2nd Validation data was collected two months after 1st Validation to test the repeatability of the algorithm [3]. Timing delays are presented in average delay ± SD, in millisecond. Negative values mean the event was detected by ANFIS before VICON.

Gait Event	Time Delay ± SD (ms)	
	1st Validation	2nd Validation
IC	4 ± 40	18 ± 38
Midstance	16 ± 64	5 ± 61
HO	-12 ± 50	-3 ± 49
OIC	-7 ± 37	-2 ± 35
TO	-5 ± 31	-12 ± 36
Feed Adjacent	-12 ± 45	-20 ± 55
Tibia Vertical	29 ± 44	19 ± 42

So, from this study, it can be concluded that EMG shows good repeatability and adequate accuracy for performing GP. It also shows that EMG can be used with patients with pathological gait. However, the ANFIS algorithm did not perform classification in real-time, which makes this system unsuitable yet for real-world applications. In the next section, we will further discuss the advantages and disadvantages of EMG for GP problems.

3.2. Advantages and disadvantages of EMG

EMG allows the classification of eight gait periods, the highest granularity used in gait analysis, only matched by IMUs [4] and video systems. However, EMG signals are very sensitive to noise, and so require higher complexity in signal acquisition and processing. EMG based methods also usually achieve lower accuracies than methods such as footswitches or IMUs and utilize more complex algorithms to perform gait analysis. However, there are some areas where EMG goes above and beyond other GP systems, especially in areas which are useful for powered prosthesis control and FES applications. Below are listed some of the main reasons why EMG was chosen in this dissertation to perform GP.

Firstly, EMG has the potential to predict human movement before it occurs, making it useful for detecting gait onset and offset [30]. This was confirmed by [50], where EMG detected gait initiation in transfemoral amputees between 63 and 138 ms prior to inertial sensors.

Secondly, EMG can differentiate between gait and non-gait activities [6]. This means that EMG will not detect false positive events, such as sitting or shifting weight from one leg to another. A false positive could lead to a loss of balance or even a fall, which would be an unacceptable risk in real-life applications.

Furthermore, EMG has better error correction potential than footswitches. In instances where gait events where not correctly identified by EMG, the error was at most resolved by the following gait event, as opposed to the use of footswitches, where errors were only detected and resolved by the next gait cycle [33].

Lastly, EMG has the ability to actively control the torque output of prosthetics, making it far superior to other systems that do not possess that ability. In fact, [9], [10] showed that patients utilising footswitch-controlled prosthetics showed worse walking patterns and higher muscle activation levels compared to subjects using proportional myoelectric controlled prosthesis, going as far as labelling footswitch controlled prosthetics as a hindrance to patients. Their results were later confirmed by Pauw et al. [42], who showed increased metabolic costs in patients using the footswitch-controlled Ankle Mimicking Prosthetic Foot prototype 4.0.

3.3. Final remarks

The objective of this chapter was to explain the main differences between the different wearable sensors used in GP, their strengths, weaknesses and performances. Examples of studies were presented for each type of sensor, to provide some insight into the different methods used by researchers to obtain gait event timings from sensor data, and compare results obtained with different kinds of sensors. One problem of this approach was the lack of a standardised metric to evaluate the success of the sensors. So, to finish this chapter, we present a comparison of time delays between the detection of IC and TO, using the different methods. All timing delays presented in **Table 3.3** were referenced either to VICON systems or force plates, to ensure a fait comparison between the values.

Table 3.3 - Footswitches, IMUs and EMG timing delays referenced to VICON systems or force plates. Footswitch, IMUs and EMG data taken from Balbinot [5], Mannini *et al.* [6], and Lauer *et al.* [3]. Timing delays are presented in average delay + SD, in milliseconds. Negative values mean the event was detected by the wearable system before the reference system.

Gait Event	Time Delay ± SD (ms)		
	Footswitches	IMUs	EMG
IC	16±1	62 ± 47	4 ± 40
TO	20 ± 9	36 ± 18	-5 ± 31

From this table, we can see the accuracy of the footswitches in relation to the other two methods, and that IMUs and EMG present similar accuracies. Even so, this dissertation will focus on the potential of EMG due to the advantages described in **Section 3.2**.

Chapter 4

Machine learning

This chapter will provide a short explanation of what is Machine Learning and why it can be used to detect gait events from EMG data. Then, the fundamental concepts of machine learning, such as feature selection and extraction and overfitting, are described, as well as how the algorithm's performance is evaluated. Finally, some examples of ML algorithms will be presented, along with their strengths and weaknesses.

4.1. Introduction to machine learning

Machine Learning is a term which denotes an algorithm that has the capability to automatically process large amounts of data, retrieve useful information from said data, and apply the information to perform a particular task. One of the tasks ML algorithms excel at is classification problems, such as the detection of gait events from EMG data [33].

A classification algorithm is tasked with labelling an instance with a particular "class" and can be deductive or inductive. In deductive problems, the rules that define the different classes are known, and the ML algorithm is tasked with applying those rules to a particular instance to determine the class of that instance. In inductive classification, the classification rules are not known, and must be estimated by the algorithm, by analysing different instances. In other words, deductive problems process from the general to the specific and inductive problems from the particular to the general [51].

Another way to describe ML algorithms is by their learning method. Supervised learning uses previously labelled instances, so the algorithm knows what class it should be outputting for a particular instance. Unsupervised learning uses unlabelled instances and is used when researchers are looking for new classes with which to label said instances. In Reinforced learning, a "trainer" provides the algorithm with a scalar feedback about its performance, and the algorithm follows the actions that maximise this scalar feedback to maximise performance [51].

In the case of gait event detection with EMG, the classification rules are not known, but data can be labelled by a reference system, like force plates or a VICON system. This means that the classification algorithms best suited for this problem are supervised induction algorithms. In this report, a classifier will refer to a supervised induction algorithm [51].

As stated, induction algorithms are fed a large amount of data (training set) and must "learn" how to automatically label new, unseen data (testing set). Since induction algorithms learn from data, it is very important that the data used to train the algorithm is reliable and fit for the purpose at hand. Data is considered "unreliable" if one of the following circumstances applies [51]:

- There is noise in the data;
- The data has missing values;
- The curse of dimensionality.

Fortunately, there are methods that can be used to counteract these problems. Pre-processing techniques, such as the use of filters, can minimize the "noisiness" of data or other techniques can be used to measure the "noisiness" of data to check if it is suitable to be used by the classifier, and missing values in many cases can simply be ignored [51].

In addition, feature selection and feature extraction are techniques used to reduce the dimensionality of a problem. Every additional feature introduces a new dimension which exponentially increases the complexity of a problem [52], so in certain scenarios it is not possible to employ the entire feature set obtained from data acquisition in the machine learning algorithm.

Feature selection and extraction solve this problem by creating new and smaller feature subsets that will be used to train the algorithm, instead of the initial feature set. Having less redundant and irrelevant data allows the algorithm to function faster and more accurately. In fact, using a proper feature subset is the most important factor in training a classification algorithm [53].

4.2. Feature selection

We have discussed that using too many features will negatively impact the performance of the algorithm, so from the available features, we must select which ones should be kept. The features that are kept are part of what is called the "feature subset". If we have n features, we can obtain 2n feature subsets. It is simply not practical to test and compare all 2n subsets, so other methods must be used to obtain a feature subset that gives the classifier a good accuracy [54]. We want the feature subset to have as few features as possible, but how do we go about selecting which features should be fed to the algorithm and which should be scrapped? The first, more subjective method, is to rely on experts in the field of the problem to indicate which features are more informative [52]. Alternatively, one can use the wrapper or filter methods to perform feature selection [53].

4.2.1. Filter method

The filter method is essentially a pre-processing step. It creates a set of criteria that the features must meet. The features that do not meet these criteria are eliminated and the ones that do are passed on to the algorithm [55]. For example, the user can specify that only features with a variance above a certain threshold, or only the k features with the higher variance can be used to train the algorithm, based on the intuitive assumption that features that do not show a lot of variance also do not introduce a lot of new information. Likewise, we can measure the correlation between features and specify a correlation threshold. Features that

correlate too much with other features introduce redundant information and so can be removed [53].

The advantage of filter methods is that they are very easy to implement, require very low computational power and are based on an intuitive understanding of the algorithm. However, they do not take into account the classifier's performance, they simply evaluate the intrinsic properties of the features to accept or delete them based on subjective criteria [54]. Another disadvantage is that the criteria used in filter methods must be manually set, i.e. thresholds, and the wrong criteria can result in poor results for the classifier [55].

4.2.2. Wrapper methods

By contrast, the wrapper methods do take into account the classifier's performance, resulting in much higher accuracies than filter methods. However, these methods are much more computationally heavy than filters and are prone to "Overfitting" [56], a concept we will discuss ahead. The main steps in wrapper methods for feature selection are show in **Figure 4.1.**

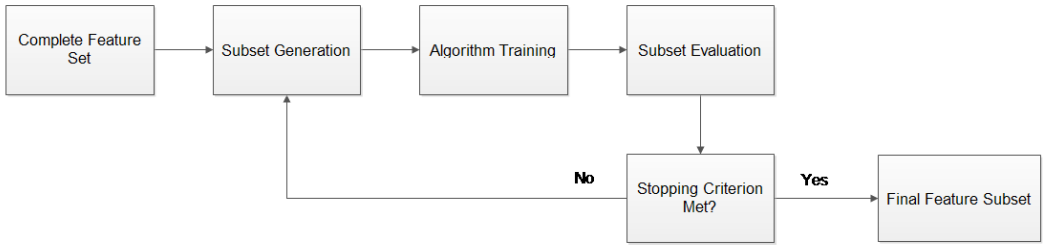

Figure 4.1 - Wrapper methods work pipeline. Adapted from [54].

There are three main philosophies in wrapper methods: complete search strategy, heuristic search strategy and random search strategy.

Complete search

Complete search algorithms compare different feature sets until the best solution (global optimum) is found. For example, the "Branch and Bound" algorithm introduced in [56] tries to find the optimal subset of size m from a set of n features. If we have 24 features, and want a subset of size 12, there are 2 704 156 possible combinations, which would be impossible to test, but the branch and bound algorithm only needs to evaluate 6000 combination to reach the global maximum, so how does the algorithm achieve this? The algorithm is able to do this by first assuming that the selection criterion function, which we want to maximise, satisfies monotonicity, meaning that if we have a particular combination of features, removing more features from this combination will give the selection function a lower value. Without going into detail, this is how the algorithm works for $n = 6$; $m = 2$: in order to select 2 features, the algorithm must delete 4 ($6 - 2 = 4$).

Let each element in $\{1, 2, \ldots, n\}$ represent one of the n original features, and $(Z_1, \ldots, Z_p)$ be the removed $p = m - n$ features. Then $(Z_1, \ldots, Z_p)$ is a solution to the problem, by telling us which features to delete. Now we introduce the selection criterion function $J_p(Z_1, \ldots, Z_p)$. The best possible solution is the one that satisfies the condition:

$$Z_p(Z_1,\dots,Z_p) = \max J_p(Z_1,\dots,Z_p) \tag{4.1}$$

Monotonicity guarantees us that:

$$J_1(Z_1) > J_2(Z_1,Z_2) > \cdots > J_p(Z_1,\dots,Z_p) \tag{4.2}$$

The algorithm then generates a solution tree, and greedily follows the node for which J_i is highest, until it reaches the final level (level p). $(Z_1, \dots, Z_p)$ is the first solution. J_p is computed and stored as J_o (J optimal), and $(Z_1, \dots, Z_p)$ is stored as Z_o. Now the algorithm explores other branches of the solution tree, using the same greedy method. If a node has $J_i<J_o$, we can conclude from **Equation 4.2** that:

$$J_o(Z_1,\dots,Z_p) > J_i(Z_1,\dots,Z_i) > J_{i+1}(Z_1,\dots,Z_{i+1}) \tag{4.3}$$

This means that $(Z_1, \dots, Z_i)$ and every node subsequent to it will have a criterion value below J_o, and so can be ignored. This is the reason why the algorithm can find the global optimal by only analysing a fraction of the possible feature sets. If the algorithm reaches the final level again, the current J_p is compared to J_o. If:

$$J_p(Z_1,\dots,Z_p) > J_o \tag{4.4}$$

J_p is stored as J_o, and $(Z_1, \dots, Z_p)$ as Z_o, and the algorithm continues exploring other branches. Otherwise, J_o and Z_o remains the same and the algorithm continues as previously. When all the nodes have been explored, the algorithm stops, and Z_o is the best possible solution for n feature selection.

Complete search algorithms, although being very simple and providing global optimum solutions, suffer from exponential complexity, meaning a small increase in the number of features will massively increase the amount of time needed to compute the answer [54]. Furthermore, they require the selection criterion to meet certain requirements, such as monotonicity [53], [56].

Heuristic search

Heuristic search algorithms rely on guesses, estimates and approximations to solve problems. As a result, a heuristic solution will rarely be the best possible one but can still be adequate to tackle many problems [54].

Forward Selection and Backward Elimination are examples of heuristic search algorithms. In forward selection, we begin with an empty feature subset and start by “feeding” the algorithm the features, individually, and checking the outputs of the algorithm. The feature that provides the most desirable output is added to our feature set. Then, the process repeats, adding one more feature to the feature subset, until the algorithm’s performance is satisfactory. Backward elimination applies the same principle backwards, meaning our initial feature set contains all available features, one feature is removed before running the algorithm and checking the output. This process is iterated until we find the feature that, once removed, has the least negative impact on the algorithm’s performance and so this feature is removed

from the feature set, and the process repeats until the algorithm's performance is satisfactory [54].

Heuristic search methods should not be used when one is concerned with achieving the global optimum solution, due to their often trial-and-error and greedy nature. However, these methods will reach a global minimum solution in a very short amount of time and with low computational power, and so can be used if a local optimum solution is adequate, or if finding a global optimum solution would be deemed too time-consuming [54].

Random Search

Random Search methods escape being trapped in local solutions by introducing a randomizing effect. However, many parameters must be set to ensure that the randomizing effect is not too strong or weak, meaning their success is dependant on many factors. Examples of random search methods are Genetic Algorithms (GA), Simulated Annealing and Particle Swarming Optimization [54].

A Genetic Algorithm mimics biological evolution to optimize a solution. The way this algorithm works for feature selection is as follows [57]: all n available features are put in a binary vector $[f_1, f_2, \ldots, f_n]$, where f_i can have value 1 – indicating that f_i is active – or 0 – indicating f_i is inactive. For example, for $n = 4$, the vector $[0, 1, 0, 1]$ means feature f_2 and f_4 are active. In the first step, the GA creates the first population of m randomized vectors. The system evaluates the classifier with each feature vector in the population and selects the best ones based on a survival criterion. The next step is called the "crossover", where the best vectors are paired and produce "offspring", a new population with the best characteristic of the previous population. In this step, "mutations" can also occur, which randomly modify the offspring. Over many generations, the population will evolve into one that maximizes the survival criterion, reaching a global optimum solution for the feature selection problem.

Random Search methods are usually slow and computationally heavy. Furthermore, they require a big set of parameters (Survival criterion, mutation rate, and initial population for GAs) to be finely tuned in order to achieve a desirable result, otherwise the algorithm can get trapped in local optimums, or not converge to a solution at all. So, although these algorithms can find global optimum solutions, this process is slow and not always guaranteed. In addition, random search algorithms are the most prone to suffer from Overfitting [54], [57].

4.3. Feature extraction

In feature extraction, instead of choosing features from the original feature set, we produce new features from the original feature set [53]. Creating new features may seem counterintuitive when tackling the problem of dimensionality but when a problem contains many features, it is likely that some features will contain redundant or irrelevant information. Feature extraction is used to combine the relevant information of several features into a smaller pool of features, by eliminating redundancy and irrelevant information. This results in an overall reduction of the dimensionality of the problem, without much loss of critical information about the original signal. Two examples of feature extraction are principal component analysis (PCA) and linear discriminant analysis (LDA). PCA is considered one of the most powerful dimensionality reduction techniques but does not take into account the classes of the data points, which can lead to clustering and poor classification. LDA on the other hand,

takes into account classes, making it extremely useful for classification problems, but it can only be used with labelled data [52].

4.3.1. Linear discriminant analysis

LDA is an extremely useful extractor in supervised classification problems because it guarantees maximum separability between classes [58]. Suppose we have a labelled classification problem with 2 classes and *n* features or dimensions. It is possible to project the data points in the *n*-dimension hyperspace onto 1 axis, therefore reducing the dimensionality of the problem. LDA finds the axis that maximises separation of the two classes.

The new axis is created according to two criteria: After the data is projected onto the new axis, the distance between the means of the two classes $(\mu_1 - \mu_2)$ must be maximised and simultaneously, the variance within each class (s_1^2 and s_2^2) must be minimised. These two criteria are joined to create the Fisher's Criteria [59], or separation equation:

$$W = \frac{(\mu_1 - \mu_2)^2}{s_1^2 + s_2^2} \tag{4.5}$$

Where $(\mu_1 - \mu_2)^2$ represents the variance between classes, and $s_1^2 + s_2^2$ represents the variance within classes. The difference between the means of the two classes is squared to avoid negative values. Since we want to maximise variance between classes and minimise variance within classes, we need to find the axis which maximises *W*.

Now that the new axis has been found, our *n*-feature dataset can be represented in only one dimension, which means that, by creating a new feature, the dataset's dimensionality has been reduced. The process remains semi-identical for a higher number of classes, but instead of creating one new axis, more axes can be created. For a *k*-class problem, the maximum number of axis that can be created is *k-1*.

Although LDA implies some loss of information, it preserves class-discriminatory information, which is the kind of information that is most relevant for classification problems [60].

4.4. Algorithm evaluation

When describing the different feature selection methods, all FS algorithms shared a step where they must analyse the classifier's performance. A classifier's performance can be evaluated in multiple ways. The simplest way to do so is to split the complete dataset into separate training sets and testing sets. After the ML algorithms learns from the training set, it can be run on the testing set. Afterwards, classification parameters such as accuracy, sensitivity, specificity, error rate, ROC AUC, etc, can be measured [61]. Other parameters such as speed and robustness (how sensitive the algorithm is to noise) can also be used if they are deemed important [51], [55].

However, a more reliable way of estimating the classifier's performance is to use *K*-fold cross-validation [62], [63]. In this method, the original training set is further split into *K* folds $(F_1, \ldots, F_K)$. Then, copies of the original training set are created, until there are *K* identical training sets $(S_1, \ldots, S_K)$ and every training set S_i is split into training and validation sets, according to this rule:

$$for\ i\ \in (1,k), F_i will\ be\ the\ validation\ fold\ in\ set\ S_i \tag{4.6}$$

This process is visually explained in **Figure 4.2**.

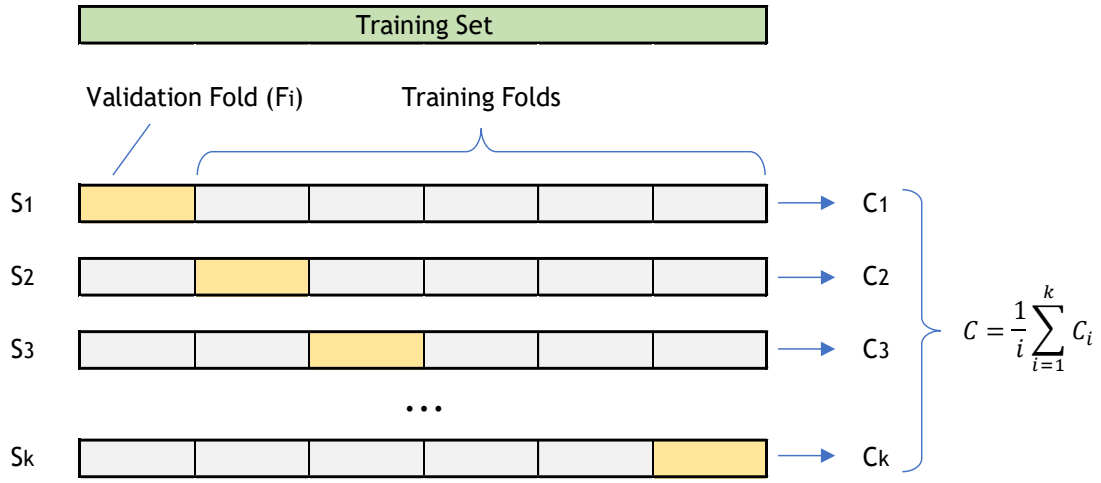

Figure 4.2 – K-fold cross-validation method for algorithm evaluation.

For every set S_i, the chosen classification parameter (C_i) is evaluated, and the average of these parameters is calculated to obtain a better estimate of the classifier's performance. When the classifier's performance is deemed satisfactory, for example when the classification parameter C surpasses a predefined threshold, the classifier is tested on the independent test set to obtain a more representative estimate of the classifier's performance [51], [55].

4.5. Overfitting

Overfitting is a problem that occurs when the inductive classification algorithms works "too well". As was explained, an inductive classification algorithm learns from the training set to inductively create a general set of rules that classify new data. If the set of rules matches too closely with the training set, the algorithm is said to have low predictive power [55]. Overfitting can happen for a number of reasons, namely if the algorithm is more complex than what is necessary, or if it has become too specialized on the training set by learning for too long. **Figure 4.3** shows how overfitting affects the algorithm's accuracy. As the algorithm becomes over-specialised on the training (validation) data, it can classify the validation set better, but loses the ability to correctly classify unseen data [51], [55].

There are, however, ways to reduce the risk of overfitting, such as using cross-validation techniques to evaluate the classifier's performance, using a model that is complex enough to model the problem **Figure 4.3** - but not too complex to induce overfitting – or using a "stopping criterion". A stopping criterion tells the algorithm when to stop training, so it does not have time to become over-specialised on the training set [55].

There is a large number of machine learning techniques used for EMG signal classification. NNs [47], [48], [64], Hidden Markov Models [49], Fuzzy Inference Systems [3], [33], K-nearest neighbour [65], Random forests and decision trees [65], and others. As expected, different algorithms will have different strengths and weaknesses, and as such will be more suited for different tasks. This chapter will give a very brief overview of some of the ML algorithms.

NN usually perform better with continuous features, and multi-dimensional feature sets. Large datasets are also typically necessary to achieve high precisions. These algorithms also perform well even when class boundaries are diagonal or non-linear. A drawback of these algorithms is their lack of transparency, or interpretability, and their relatively slow training and classification [51].

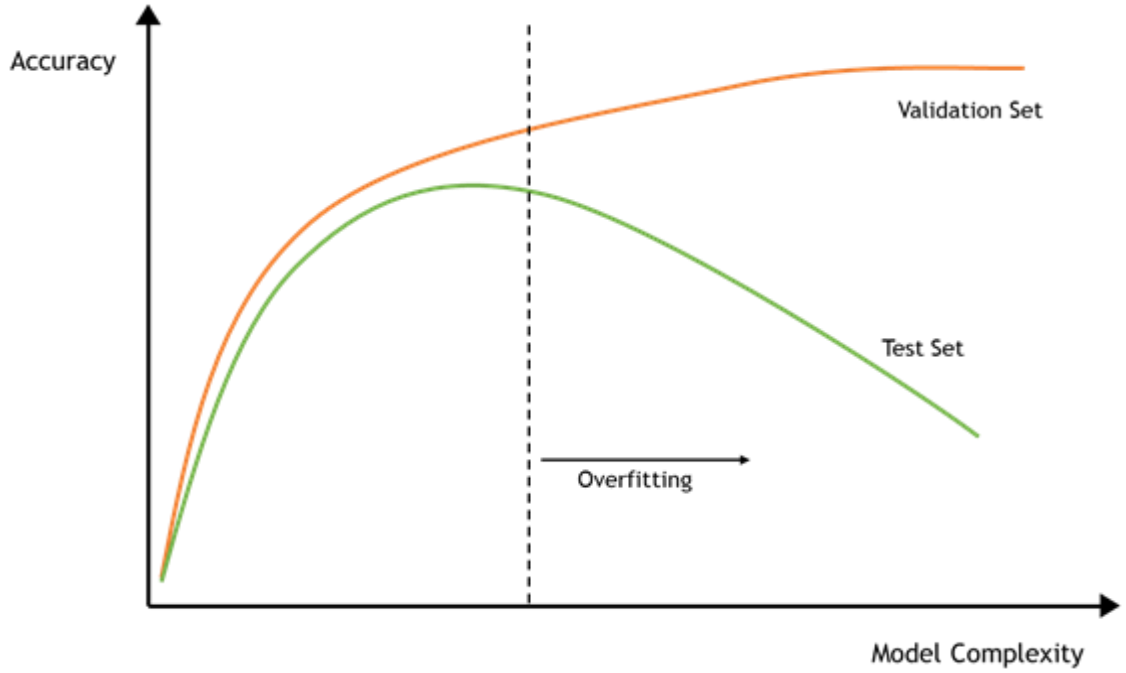

Figure 4.3 – Overfitting illustration. Utilizing very complex models will decrease the test set accuracy, while improving the validation set accuracy. The dotted line represents when overfitting begins, which coincides with a drop in test set accuracy.

Decision trees, on the other hand, perform better with discrete features, are several orders of magnitude faster than NNs and are very interpretable. However, decision trees are unable to classify diagonally separated classes [51].

K-nearest neighbour algorithms have faster training than other algorithms but very slow classification. They are also very sensitive to irrelevant features and have low interpretability [51].

Hidden Markov Models are interpretable but are not well-suited for many problems with high dimensionality [51].

Fuzzy Inference Systems have slow training and a lot of parameters that must be set. This acts as a double-edged sword, since it allows an experienced user to tailor the algorithm to their liking but is a very daunting task to an unexperienced user or can lead to inaccurate results. However, fuzzy are very interpretable and for this reason are very used in medical settings [51].

As a result, it is impossible to declare that one algorithm is better than all others, as they all have their qualities and defects. Therefore, it is up researchers to analyse a particular problem and decide which algorithm is best suited for solving it.

4.6. LSTM

Long-Short-Term-Memory is a type of Recurrent Neural Network (RNN) introduced in 1997 [66] that can process sequential information, such as time-series data, language or any kind of information where the sequence of information is fundamental to its meaning. For example, the phrases “Man bites dog” and “Dog bites man” are composed of the same words (“man”, “bites” and “dog”), but have completely different meanings. Although a regular feed forward NN is unable to process these kinds of sequential dependencies, RNNs have feedback loops which, akin to “memory”, allow the persistence of information.

An RNN layer receives an input sequence of size [j x m] and is composed of j RNN cells. A regular RNN cell receives two inputs: xt and ht-1, where xt is the tth element of the input sequence and ht-1 is the output of the RNN for the previous inputs: xt-1 and ht-2, and outputs ht. In other words, at every timestep an RNN receives the current element from the input sequence and the previous output of the RNN to produce an output. This phenomenon can be illustrated in **Figure 4.4**.

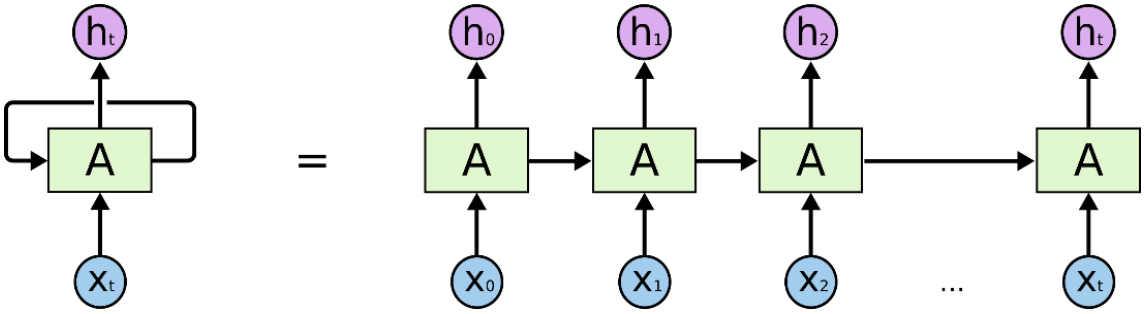

Figure 4.4 - Left side of equation: Rolled RNN cell. A represent the RNN cell, x_t the input and h_t the output. Right side of equation: Unrolled RNN cell. This view allows the flow of information along the different timesteps. From [67].

On the right side of the equation, we can clearly see the recurrent aspect of RNNs. This view of an RNN cell is denominated “Unrolled” and allows a better understanding of the overall functioning of an RNN. An LSTM is a special type of RNN. An LSTM layer also presents a chain-like structure and receives a sequence of size [j x m] and has j LSTM cells. To fully understand the workings of an LSTM, we must analyse the operations that occur within a single LSTM cell [68], as shown in **Figure 4.5**.

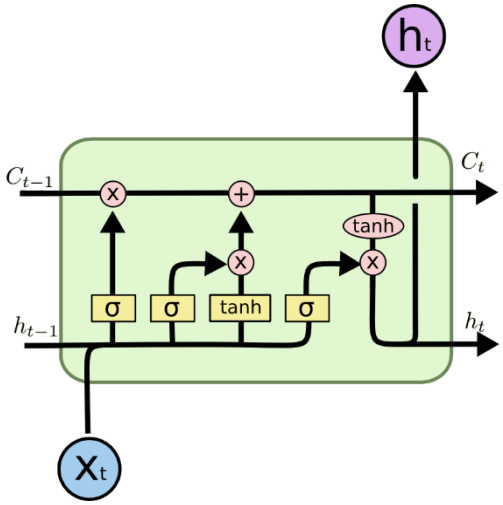

Figure 4.5 - Representation of internal operations of an LSTM cell. From [67].

Within the LSTM cell there are three gates represented by the lower case sigmas (σ): forget gate, input gate and output gate. Each gate acts as a single layered feed-forward NN, and its these gate's interactions that allow the LSTM cell to keep track of previous information to make predictions. In addition, there are 3 inputs and 2 outputs to each LSTM cell. These inputs are:

xt - The input sequence at timestep t, with size [1 x m], m being the number of features in the input sequence.

Ht-1 - The hidden state vector at timestep t-1, with size [1 x n].

Ct-1 - The cell state vector at timestep t-1, with size [1 x n].

And the outputs are:

Ht - The hidden state vector at timestep t, with size [1 x n].

Ct - The cell state vector at timestep t, with size [1 x n].

Of these inputs and outputs, the cell state is the centerpiece of the LSTM network. It acts as the network's "memory", transporting information along the LSTM chain. Information can be added or removed from the cell state, but before doing so it must pass through the cell's gates, which control which information to add and remove. The presented diagram can be quite confusing, so the gating mechanisms will be explained one by one to make comprehension easier.

The first gate is the forget gate presented in **Figure 4.6**.

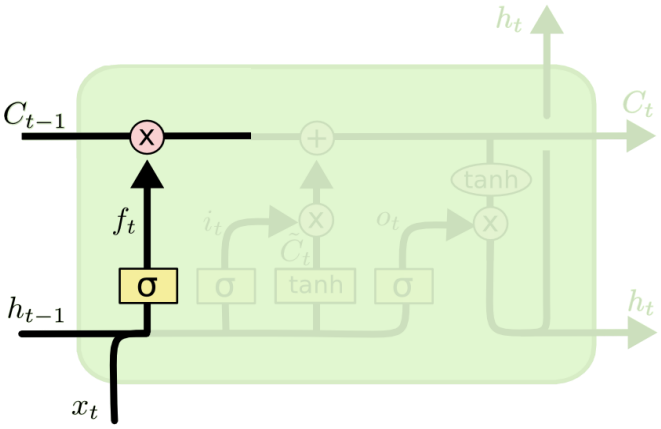

Figure 4.6 - The forget gate. Adapted from [67].

The forget gate takes inputs x_t and h_{t-1} and passes them through the forget function:

$$f\_t = \sigma(x_t \cdot U_f + h_{t-1} \cdot W_f + b_f) \tag{4.7}$$

Where Uf and Wf are weight matrices of size [m x n] and [n x n] respectively, and bf is a bias vector of length n. n is not a predetermined parameter, but can be any natural number, although this number must be consistent across the entire LSTM layer. The values in Uf and Wf are parameters of the LSTM cell. These parameters are responsible for the training process of the network/cell and will be adjusted during the training phase to improve the network's performance. The multiplication xt•Uf results in a matrix of size [1 x n] and the multiplication ht-1•Wf also results in a matrix of size [1 x n]. The sum of these two matrices and vector bf is performed element-wise, which results in a vector of length n. This vector is then passed through the following sigmoid function:

$$\sigma(y) = \frac{1}{1 + e^{-y}} \tag{4.8}$$

The output of forget function is the "forget" vector, f_t, of length n and values with range [0,1]. This vector will be multiplied element-wise with Ct-1. As mentioned, the cell state acts as the memory of the LSTM cell. By multiplying an element of Ct-1 by a value close to 1, that element remains mostly unchanged, whereas multiplying an element by a value closer to 0 will result in the network mostly "forgetting" that element. In short, the forget vector dictates how much of each element of the previous memory state, Ct-1, should be forgotten and how much should be kept, hence its name.

The next gate is the input gate presented in **Figure 4.7**.

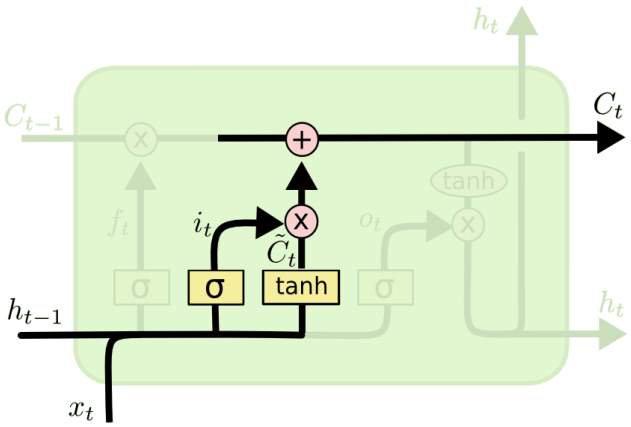

Figure 4.7 - The Input gate. Adapted from [67].

The input gate works fundamentally like the forget gate, except with different weighted matrices Ui and Wi and a different bias bi, although these are all the same size as their counterparts in the forget gate. The input function is as follows:

$$i_t = \sigma(x_t \cdot U_i + h_{t-1} \cdot W_i + b_i) \tag{4.9}$$

The function takes the same inputs as the forget function and uses the same sigmoid function σ(y), and outputs the input vector i_t with length *n* and range [0,1]. However, this vector is not multiplied by the previous cell state C_{t-1}, but by a candidate vector, $\bar{C}_t$. The candidate vector contains values that could be added to the previous cell state to create the new cell state vector. Similarly to the forget vector, which defines which values of C_{t-1} to keep, the input vector defines which values of $\bar{C}_t$ should be kept and ultimately added to C_{t-1}.

$\bar{C}_t$ is created according to a candidate function, similar to the input and forget functions:

$$\overline{C_t} = \tanh(x_t \cdot U_{\bar{C}} + h_{t-1} \cdot W_{\bar{C}} + b_{\bar{C}}) \tag{4.10}$$

However, instead of using the sigmoid function σ(y), the candidate function uses a hyperbolic tangent function, tanh(y):

$$\tanh(y) = \frac{\sinh(y)}{\cosh(y)} \tag{4.11}$$

As a result, the candidate vector $\bar{C}_t$ has elements in range [-1,1]. As stated, the element-wise multiplication of $\bar{C}_t$ and i_t results in a vector that will be added element-wise to C_{t-1}. At this point, some information has been removed from C_{t-1} and some new information has been added, updating it to the current cell state C_t.

The final gate is the output gate presented in **Figure 4.8**.

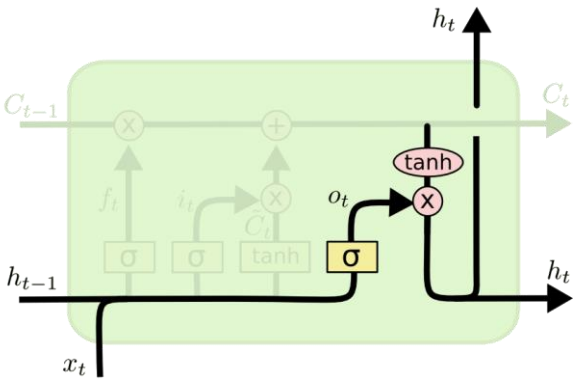

Figure 4.8 - The output gate. Adapted from [67].

The output function is:

$$\sigma\_t = \sigma(x_t \cdot U_o + h_{t-1} \cdot W_o + b_o) \tag{4.12}$$

The output of this function is the output vector with length n and range [0,1]. This vector will be multiplied element-wise by tanh(Ct). This multiplication dictates which values of tanh(Ct) will be stored in ht. This vector ht is called the hidden state vector, and the LSTM cell outputs two copies of this vector. One copy will be inputted into the next LSTM cell of the current layer, if one is present. The other copy will become the input of the succeeding LSTM layer at timestep t as shows in **Figure 4.9**. If there is no succeeding LSTM layer, the output of the current LSTM layer is the vector hj which is the hidden state vector at timestep j (j being the length of the LSTM input sequence).

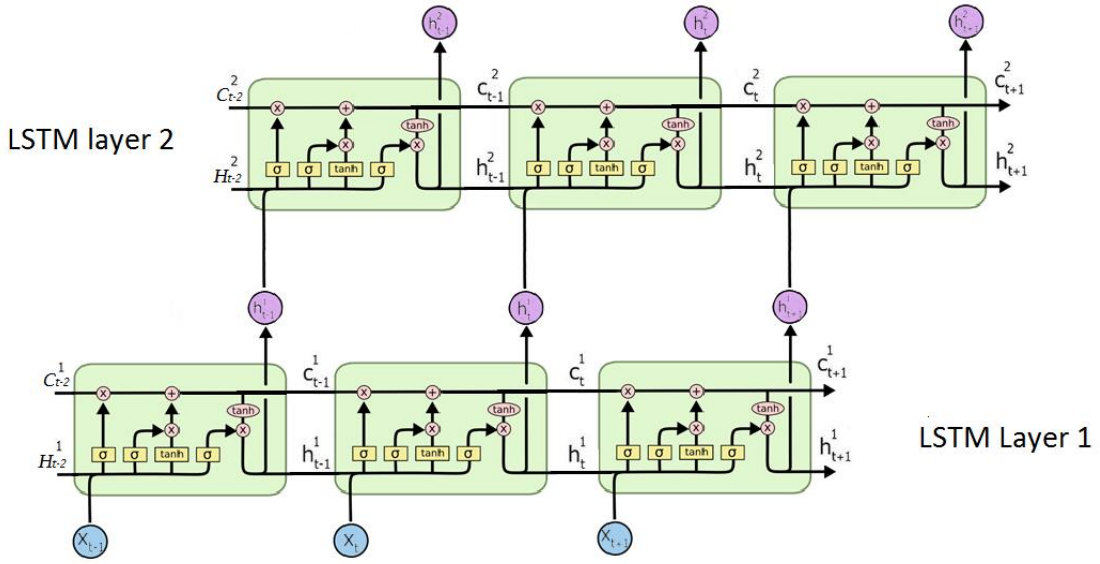

Figure 4.9 - Diagram of Stacked LSTM layers. Adapted from [67].

Overall, an LSTM layer acts as a function. This function receives j inputs of size [1 x m] and outputs either j outputs of size [1 x n] or a single output of size [1 x n]. This function also contains numerous parameters which will be adjusted throughout the training phase. Namely, these parameters are the values of the matrices U and W, and the biases b. As mentioned, each of the four LSTM functions (forget, input, candidate, output) has a set of matrices U and W and a vector b. The parameters in U and W are called weights and the parameters in b are called biases, meaning that each function has

$$m \times n + n \times n + 1 \times n \tag{4.13}$$

parameters. Thus, the total number of parameters in an LSTM layer is:

$$P(n,m) = 4(m \times n + n^2 + n) \tag{4.14}$$

In conclusion, LSTMs are a type of RNN that allows the flow of information throughout the use of a cell state. Information can be added or removed from this cell state with the use of gating mechanisms. This allows the LSTM to retrieve relevant information from a sequence and make accurate predictions on sequential data.

4.7. Loss function

The Loss function chosen for this classifier was “Binary Cross-Entropy" [69], which is defined as:

$$L_i(y_i, p_i) = -y_i(\log(p_i) + (1 - y_i)(\log(1 - p_i)) \tag{4.15}$$

Where is the label of sequence i and p_i is the algorithm’s prediction for sequence i. Since yi must be either 0 or 1, one of the terms of the function will be multiplied by 0, simplifying the expression. The loss function measures how “correct” a prediction is. If an output vector is similar to the desired label, the loss function output, denominated “loss”, will be close to 0,

otherwise it will be a higher value. For this reason, it is the algorithm's goal to minimize the lost function.

However, the loss function is dependent on and so is dependent on the classifier's function. It takes all the network's parameters as inputs, and outputs a single value, the loss. Since a network can have hundreds of thousands of parameters, finding the global minimum of the loss function can be all but impossible. However, there are different strategies made to overcome this problem and help the algorithm find a local minimum for the loss function, called "optimizers".

4.8. ADAM Optimizer

In this thesis we used the Adam optimizer, which was first published in 2014 [70] and has since been widely used in Deep Learning scenarios. Adam stands for Adaptive Moment Estimation, meaning it computes adaptive learning rates for each parameter individually. To fully understand how the Adam optimizer works, it is important to understand the concept of gradient descent (GD) and momentum. Mathematically speaking, the gradient in GD is a vector containing the partial derivatives of the cost function with respect to each of its inputs, which are the algorithm's parameters. Therefore, by knowing the partial derivative of the cost function for one parameter, the optimizer can calculate how to shift that parameter to lower the cost function. The general GD function is:

$$W_{i,h+1} = W_{i,h} - \eta \times \frac{\partial}{\partial W_i} L(W_i) \tag{4.16}$$

Where Wi, h+1 is the parameter i at training step h+1 and Wi, h is the value of the same parameter at the previous step. ∂ L(Wh)/ ∂ Wi is the partial derivative of the lost function with respect to Wi at step h. Since we want to minimize the loss function, the change to Wi will have opposite sign to the partial derivative. If ∂ L(Wh)/ ∂ Wi is large, a change to Wi will result in a relatively large change for L(Wh), meaning this parameter will have a big impact on the value of L. Therefore, to rapidly decrease L(Wh), the change to a parameter should be proportional to its partial derivative. Eta (η) is the learning rate. A large eta will result in bigger shifts and faster optimization but can cause the optimizer to overshoot the minimum of the Loss function. An algorithm with a low learning rate will be better at finding a local minimum but will have slower training. We can rewrite the GD function in the following way:

$$W_{h+1} = W_h - \eta \times \nabla \mathrm{L}(\mathrm{W_h}) \tag{4.17}$$

In this case, Wh+1 is a vector containing all the updated parameters, Wh is the vector containing the current weights and L(Wh) is the gradient of the loss function, which is a vector containing the partial derivatives of the cost function with respect to each of its parameters. Similarly, we can write **Equation 4.17** as:

$$\Delta \mathrm{W_h} = -\eta \times \nabla \mathrm{L}(\mathrm{W_h}) \tag{4.18}$$

Which gives us the vector ΔWh containing the changes that must be done to each parameter Wi at step h in order to provide the largest decrease to the loss function. GD, however,

has two major disadvantages. The first is its liability of letting the loss function become trapped in local minima, which results in worse algorithm performance. The second is a slow computational power. Momentum is a different optimization strategy that aims to solve both these problems.

Momentum builds on the idea of GD. Instead of always following the gradient of the loss function at step h, momentum keeps part of the previous gradient at step h-1. Momentum gets its inspiration from the physical concept of momentum, and as such it is easily understood with a physics analogy. Let us imagine a ball rolling down a hill with constant slope. The ball is the value of the Loss function, the hill is the topology of the loss function and the slope of the hill is the gradient of the Loss function, which is constant. Using GD, ΔWh is constant since ∇L(Wh) is also constant. This means the rate at which the value of the loss function decreases is constant, since it is proportional to ΔWh. In our analogy, it means the ball rolls down the hill with constant speed. In reality, due to momentum, the ball's speed would increase as the ball continues to roll, even though the slope of the hill does not change. In other words: the speed of the ball in the present depends simultaneously of the slope of the hill and the speed of the ball in the past. The same happens with Momentum in machine learning. The rate at which the parameters change at step h (ΔWh) depends on the gradient of the loss function at step h (∇L(Wh)), and on the rate at which the parameters change at step h-1 (ΔWh-1). Mathematically:

$$\Delta W_h = \beta \times \Delta W_{h-1} - \eta \times \nabla \mathrm{L}(\mathrm{W_h}) \tag{4.19}$$

Substituting ΔWh-1 by - η x ∇L(Wh-1):

$$\Delta W_h = -\eta\left(\beta \times \Delta \mathrm{W_{h-1}} + \nabla \mathrm{L}(\mathrm{W_h})\right) \tag{4.20}$$

Alternatively, we can write **Equation 4.20** as:

$$\Delta W_h = -\eta \times M(h) \tag{4.21}$$

Where:

$$M(h) = \beta \times M_{h-1} + \nabla \mathrm{L}(\mathrm{W_h}), \mathrm{M}(0) = 0 \tag{4.22}$$

In **Equation 4.22**, ß is a hyper-parameter in [0,1] that decides how much importance should be attributed to ΔWh-1. This value is commonly set to 0.9. Adam implements the moment function that has been described in Equation 4.22, with some minor alterations. The equation for momentum in Adam is:

$$M(h) = M_{h-1} + (1-\beta) \times \nabla \mathrm{L}(\mathrm{W_h}), \mathrm{M}(0) = 0 \tag{4.23}$$

This effectively makes the moment M at step h an exponential moving average of the previous moment, Mh-1, and the current gradient of the Loss function ∇L(Wh).

In the beginning of this section, we mentioned that Adam computes adaptive learning rates. However, so far in our Adam equation, the learning rate η is still a fixed constant. To

achieve an adaptive learning rate, a parameter G(h) is added to Equation 4.23, resulting in the final Adam equation for adapting parameters:

$$\Delta W_h = -\frac{\eta}{\sqrt{G(h)+\varepsilon}} \times M(h) \tag{4.24}$$

$$M(h) = \beta \times M(h-1) + (1-\beta) \times \nabla L(W_h), M(0) = 0 \tag{4.25}$$

$$G(h) = \gamma \times M(h-1) + (1-\gamma) \times \nabla L(W_h)^2, G(0) = 0 \tag{4.26}$$

In this equation, G can be interpreted as the second moment of the gradient of the loss function. γ is a hyperparameter like ß in [0,1] that regulates the weight of G(h-1) and usually has value 0.999. ε is a small value, 10-8, added so that the denominator cannot be 0.

4.9. Dropout

Dropout is a technique used to improve a model's accuracy and reduce overfitting [71]. The idea behind Dropout is that some parameters will have a bigger impact on the training of the model than others. As a result, these parameters will have larger impacts on the Loss function, making their updates larger than updates to other parameters. Effectively, this can result in certain parameters dominating training and others not having an impact, which decreases the model's performance. Dropout solves this problem by randomly blocking certain connections in a layer for every training sequence. This means that all parameters will have the opportunity to train effectively, without relying on dominant ones. There are many ways to add Dropout to an LSTM, but Zaremba et al. [72] suggests the method presented in **Figure 4.10**.

In **Figure 4.10**, we can clearly see that Dropout was only applied to non-recurrent connections allowing the network to gain the benefits of Dropout regularization without impacting the network's memorization ability.

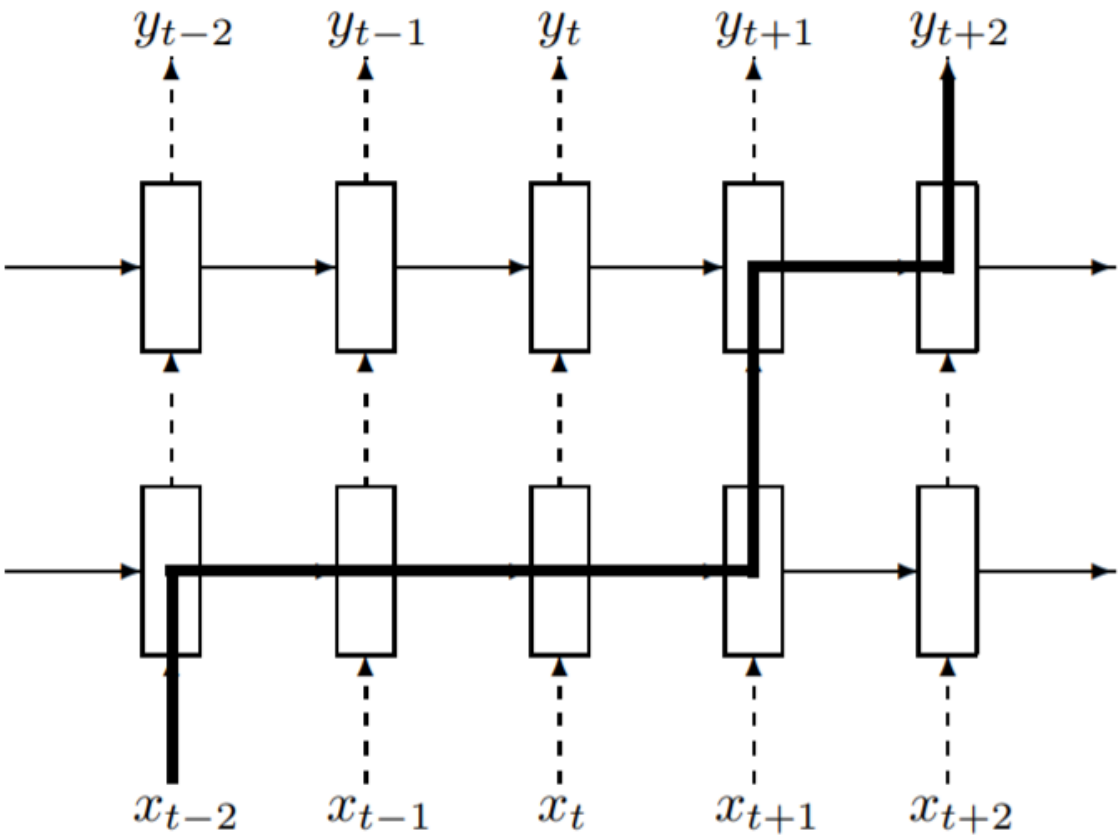

Figure 4.10 - Example of unrolled double-layered LSTM network. The dashed lines indicate connections were dropout is applied, the solid arrows indicate the recurrent connections, where dropout is not applied. Finally, the thick line shows the flow of information in the network. after Zaremba et al. (2014) [72].

Chapter 5

Electromyography

This chapter will review some fundamentals of EMG signal. While chapter 3 focuses on the applications and history of this technique, this chapter will discuss the more technical aspects of EMG signal characteristics, acquisition and processing.

5.1. Introduction to EMG

EMG is a technique commonly used to read the bioelectrical signals produced by skeletal muscles when contracting for a wide variety of applications. Most notably, EMG signal can be used in therapeutic settings, as shown in chapter 3. There are two types of EMG: intramuscular (Im-EMG) and surface EMG (sEMG) [73]. Surface EMG has the major benefits of being easy to use and non-invasive, but can only provide information about superficial muscles, and is very sensitive to crosstalk phenomena, meaning it can record signal from multiple muscles, rendering this technique not specific [74]. Intramuscular EMG, on the other hand, can capture signal from only one muscle and is less susceptible to noise, but is invasive and harder to use, making it unpractical for many clinical applications [69]. For this reason, sEMG is used in most clinical and therapeutic settings, such as myoelectric prosthesis control [75], and this dissertation will deal with sEMG. Unless stated otherwise, EMG will refer to sEMG in this dissertation.

5.2. Signal description and characteristics

A nerve and the muscle fibres it innervates are called a motor unit. Once an action potential reaches a muscle fibre, it propagates through the motor fibre and is termed the motor action potential, or MAP. A motor unit action potential (MUAP) is the sum of all MAPs in a motor unit. The electromyographic signal is the sum of all the MUAPs firing near the electrode. Since the MUAPs are not synchronized, this sum can have positive and negative voltage values, ranging from -5mV to 5mV [73], and frequencies mainly ranging from 50 to 150 Hz [19]. However, some applications may require the recording of higher bandwidths, in the 150-500 Hz range, to detect subtle changes such as fatigue in the muscle fibres [75].

5.3. Signal acquisition and processing

Since the amplitude of the signal is so low, it is very sensitive to noise and the adequate signal processing practices must be put into place to improve signal-noise ratio and assure proper quality of the data [73]. The first step to acquiring good quality EMG signal is the correct placement and fixation of the electrodes. Surface EMG for the Non-Invasive Assessment of Muscles, or SENIAM, has a detailed guide containing correct electrode placement for 28 different muscles, and explaining techniques such as correct skin preparation to ensure good contact between the electrodes and the skin and minimize noise sources [76].

The next step is choosing an appropriate EMG sensor. Depending on the application, different bandwidths of the electromyographic signal will need to be recorded, meaning that an adequate sampling frequency must be chosen. If only the main 50-150 Hz bandwidth is of interest, a 500 Hz sampling frequency would be sufficient for signal acquisition, however, for studying higher bandwidths higher frequencies are necessary [19]. According to the Nyquist-Shannon sampling theorem, the sampling frequency must be, at least, twice as high as the highest frequency component of the studied signal [77]. For this reason, many studies use a sampling frequency of 1000 Hz [46], [48], [78] or higher [3], [9], [19] in many gait analysis applications.

After the signal is acquired, the signal processing stage begins. The first step of processing EMG signal is rectification, since only positive values are of use, so the negative values must either be discarded (Half-wave Rectification) or the absolute of the signal must be taken (Full-wave Rectification). Usually, full-wave rectification is used [73]. Afterwards, filtering techniques should be used to reduce unwanted noise and improve signal-noise ratio [73]. In literature, both High-Pass and Low-Pass filters are used, while notch filters are avoided to minimize distortion of the original signal [73].

High Pass filters are used to remove low-frequency noise (baseline drift), which can be caused by several reasons, but mainly is due to motion artefacts (sliding of the muscle relative to the electrode), skin-electrode interface artefacts (thermal and electro-chemical noise) and the inherent instability of the signal due to the randomness in the firing rate of the motor units [73], [79]. There is no consensus as to what cut-off frequency or slope should be used for the High-Pass Filter, but many researchers follow the SENIAM guidelines, which use a 20 Hz, 12dB/oct Butterworth filter [48]. The Butterworth style filter is used because it has no ripples in the bandpass region, so the original signal is not distorted [79].

The Low Pass filter, which is designed to remove high frequency noise is typically used at the 400-500 Hz range because in this zone the amplitude of the noise usually surpasses the amplitude of the EMG signal [79].

Even when taking all the correct measures during signal acquisition and processing, it is near impossible to obtain a completely noise-free signal. Some sources of noise simply cannot be completely eliminated by standard methods [79], and so researchers must take extra care to check for their presence on the acquired signal. Two of these sources of noise are electrical background noise and baseline contamination with muscle activity. One way of correctly identifying these artefacts is to analyse the frequency spectrum of the signal, since electrical noise shows a big power spike at 50 Hz [80], and muscle activity contamination will show a frequency spectrum similar to muscle contraction but appears during periods when the patient is at rest. After the electromyographic signal has been acquired, filtered and checked

for noise artefacts, researchers can begin studying the signal and retrieving information from it.

5.4. Feature retrieval

Raw EMG is not a suitable candidate for determining gait periods or detecting gait events, due to its pseudo-random nature. As a result, researchers usually decompose the signal into several features [3], [33], [38], [48]:

- Raw EMG (r-RMG);
- Standard Deviation (SD);
- Root Mean Square (RMS);
- Mean Absolute Value (MAV);
- Integrated EMG (i-EMG);
- Waveform Length (WL);
- Zero Crossing (ZC);
- Slope Sign Changes (SSC);
- EMG Histogram (HIST);
- EMG Derivative (d-EMG);
- EMG Envelope (e-EMG);
- Autoregressive coefficients (AR);
- Mean Power (MNP);
- Peak Frequency (PKF).

This process is commonly mislabelled in literature as “Feature Extraction”. However, feature extraction is a dimensionality reduction process, where a signal or feature set is reduced to a more workable format, with minimal loss of relevant information (see chapter 4.3). Decomposing the EMG signal into SD, RMS, MAV and other features results in a lot of lost valuable information and increases dimensionality. Therefore, in this work we will label this process as “Feature Retrieval” to avoid any confusion with feature extraction.

Features can be retrieved from the time domain (TD); like RMS, MAV or SD; or from the frequency domain (FD); such as MNP or PKF. However, retrieving features from the FD is slower, since it requires signal transformation, and FD features often have reduced classification power, which makes them unsuited for real-time classification problems [48], [65]. The combination of retrieved features is called the original feature set, which will, by processes of feature selection and/or extraction, originate the feature set which will ultimately be fed to the ML algorithm for training and classification.

Chapter 6

Methodology

In this chapter we will present the methodology behind the acquisition and pre-processing of our data, the labelling and sequencing process. Finally, we give a brief overview of the machine learning algorithm, and how we statistical test were performed.

6.1. Data acquisition and processing

EMG data was collected from 14 healthy subjects, aged 21-38, from both left and right leg mGAS and TA muscles - using a Trigno Avanti Sensor [81] - with a 2000Hz sampling frequency, following SENIAM guidelines [82]. EMG signal pre-processing was performed in accordance to literature [83] [3]. Overall, 73 trials were performed, with EMG being collected from 4 muscles, ultimately resulting in 292 series of EMG data. Ground RF data was acquired with the use of four Bertec [81] force plates (models 6090-15 and 4060-15), and sampled at 2000 Hz and synchronized with the EMG data. All data acquisition was performed at Laboratório de Biomecânica do Porto (LABIOMEP) [3].

6.1.1. EMG pre-processing

EMG pre-processing was performed in Matlab r2016 Mathworks. To create the envelope of the EMG signal, the signal was first rectified and resampled at 500Hz. Rectification was performed by subtracting the average of signal from the signal (remove DC component). Before resampling, the signal passed through a second order Butterworth Low Pass Filter with a 600Hz cutoff frequency, in accordance to [3]. This filter acts as an anti-aliasing filter to avoid aliasing during the resampling process. Resampling was performed by keeping every fourth sample, since 2000/500=4. The absolute of the resampled signal will then be passed through another 2^{nd} order Butterworth LP filter with a cutoff frequency of 4Hz, to obtain the envelope of the signal. The final step consisted in normalizing the envelope. Normalization has a lot of benefits: it allows us to make comparisons between the signal of different muscles, subjects and days. Although there are many normalization methods, we chose the Min-Max scaling method since it is best suited for comparing the patterns of the EMG signal over time and between different subjects [85], [86]. The normalization function chosen was the following:

$$x' = \frac{x - mean(x)}{\max(x) - \min(x)} \tag{6.1}$$

This function linearly "squashes" the values of the e-EMG between 0 and 1, according to the minimum and maximum values recorded. In addition, having our data in a [0,1] interval improves the ML model's speed and accuracy. Finally, the emg data is in a normalized state, but before the data can be fed to an LSTM, labels must be assigned to data.

6.1.2. Labelling

The labels for the gait events were obtained from ground RF. IC and TO events were detected by calculating the average and variance of the baseline signal of the vertical component of the force. IC was considered the sample where the vertical force surpassed a threshold defined as the baseline average + 10 SDs, and TO was the sample where this value went below the same threshold. This process was implemented using Matlab r2016a Mathworks. With the use of force plates, we are restricted to only detecting IC and TO events and cannot obtain more granular information about the gait cycle.

The gait events were divided into 2 categories (Left Leg events and Right Leg events) and two different label vectors were formed, one for each leg. Each element of the label vector represented the gait phase for that sample, a 1 representing "Stance" phase and a 0 representing "Swing" phase, meaning IC and TO mark the transitions between phases, or the transitions between 0's and 1's, and 1's and 0's, respectively. Two additional vectors were created for each leg, one depicting IC and TO events 15 samples early and one depicting IC and TO events 15 samples early. These vectors were used to test how well the algorithm can predict events happening in the future.

6.1.3. Sequencing

The next step in pre-processing involves splitting the data into sequences and a respective label. The LSTM algorithm works by looking at the last *j* samples in the signal and predicting if the final sample in that sequence corresponds to a Swing state or a Stance state. To do so, different data series were combined together to form new multi-feature data series. Specifically, 3 different feature sets were tested: l-mGAS + r-mGAS, l-mGAS+ l-TA, r-mGAS+r-TA. Each new data series was divided into multiple sequences of length *j=180*. Each sequence was assigned the label ("Stance" or "Swing") of the last sample in said sequence. In this manner, a total of 91652 labelled sequences were created, which formed our dataset. However, before beginning the training of an LSTM algorithm, we must make sure the dataset is balanced. This means there must be roughly the same amount of "Stance" and "Swing" labels in the dataset. To achieve this, the algorithm counts how many Stance and Swing labels there are. From the label with the higher count, sequences were removed until, finally, there are the same number of each label. Then, all the remaining sequences are added to a final database. This step eliminates 22374 sequences labelled as "Stance", leaving 69278 sequences in the final database.

The final step in the pre-processing phase involves dividing the final database into training and testing subsets. An 80/20 split is common in many ML and Deep Learning algorithms, so 80% of sequences were used to train the algorithm, leaving 20% for testing. In the end, both subsets were shuffled.

6.2. Algorithm structure

6.2.1. Algorithm description

In this thesis, we utilized Google's open-source machine-learning library - Tensorflow 1.13.1 [87] - running on Python 3.7.3 to develop and train the machine learning algorithm. Tensorflow implements a version of Keras 1.0.8, a user-friendly high-level API which allows modular crafting of complex machine learning algorithms.

For classification, a Deep Learning model with 2 stacked LSTM layers, connected to a 32-neuron NN layer connected to a final 2-neuron Dense NN layer was created. Dropout rates of 0.4 were introduced to reduce the effects of overfitting. Inputs into the first LSTM layer had shapes (*j*,*m*), *j* being the length of the sequences (j=180) and *m* the number of features (m=2). The output of the first LSTM layers is the hidden cell state vectors for each time step in the input, which we specified to be of length *n*=128. Standardization is applied to every layer's output (except the final layer), which rescales the outputs to have mean 0 and unit variance (SD = 1). The second LSTM layer receives the (*1*,*n*) output of the previous layers and outputs the hidden cell state vector of the last timestep, resulting in a single vector of length 64. This vector is fed to a Dense Layer, which outputs a vector of length 32 with "rectified linear" activation, which is inputted into a final Dense Layer with two outputs. Overall, there are 119,842 trainable parameters in the entire model, which were initialized with Xavier-Glorot Uniform Initialization. The loss function chosen for the model was "categorical cross-entropy" and the optimizer was ADAM with η=0.0005, ß=0.9 and γ=0.999.

6.2.2. Algorithm output

The final layer's output is a vector of length *k*=2 which provides the classification of the algorithm. Ideally, a "Stance" input sequence would be classified with a [0,1] vector and a "Swing" sequence would be classified with [1,0]. In reality, the outputs can have values outside of the [0,1] range and don't provide much information about the algorithm's classification. The "Softmax" [88] activation function is used to circumvent this problem, and is defined as:

$$\sigma(a) = \frac{e^{a_i}}{\sum_j e^{a_j}} \tag{6.2}$$

Where a represents the output vector, a_i is the value of element i in vector a, and $\sigma(a)$ is the transformed value of a. The main advantages of the Softmax function is that the transformed output values are within the range [0,1], and the sum of all values in the output vector equals 1. For these two reasons, outputs from a Softmax function are typically seen as the probability of a classification. For example, a [0.96, 0.04] output from our classifier could be interpreted as the algorithm being 96% sure that the input sequence should be classified as a "Swing" [88]. Furthermore, the Softmax function is differentiable, meaning it can be used for optimizers that require gradient descent.

6.2.3. Initialization

Initialization is the process of setting the initial values of weights (biases are initialized as 0). Although this process may seem unimportant (the weights will be updated throughout the learning process), an incorrect initialization can lead to much worse training speed. The overall concept behind initialization is that if the initial weights are too large or too small, an output can become saturated, making the algorithm perform worse and need more epochs to reach satisfactory results. Saturation occurs

In this thesis, we used state-of-the-art Xavier-Glorot Uniform Initialization [89]. In this method, each weight in a layer is chosen from a uniform distribution in [-x,x] where:

$$x = \sqrt{\frac{6}{n_{in} + n_{out}}} \tag{6.3}$$

n_{in} is the number of inputs of the layer and n_{out} is the number of outputs of the same layer. In the original paper, Glorot and Bengio demonstrated the effectiveness of their initialization, which not only solved the problem of saturation, but resulted in a faster convergence and higher accuracy.

6.2.4. Algorithm evaluation

Accuracy was used to measure the algorithm's performance. This metric simply divides the number of correct classifications by the total number of classifications. Our algorithm reached testing accuracies as high as 99,55%. However, for the purpose of detecting IC events, this metric is rather ineffective. To measure the algorithm's performance at detecting IC, individual sequences' classifications were sorted back into their respective trials in order to create classification vectors. These classification vectors have length=s-179, s being the number. of samples in their respective trial. In these vectors, a IC event is defined as the sample where the classification transitions from 0 to 1 and TO as the transition from 1 to 0. To find these samples' indexes, a moving window was passed through the classification vectors, looking for a match to [0,1] for IC or [1,0] for TO. The predicted IC and TO transitions indexes of each trial were stored and compared to the actual IC and TO indexes determined from the label vectors. Predicted events within 50ms of an event were considered True Positives (TP), otherwise they were considered False Positive (FP). If there were no predicted events within 50ms of an event, that event was considered a False Negative (FN).

The result was two vectors with the sample delays of the detection of TO and HO in each trial, which were converted to time delays by multiplying by the sampling time (2ms). Finally, to calculate the overall performance of the classifier, the mean and SD of these vectors were calculated, as well as sensitivity (TP/(TP+FN)) and precision (TP/(TP+FP)). One of the objectives of this thesis is to detect IC and TO within 125ms. To do so, we need to calculate if the errors follow a normal distribution by performing a Shapiro Wilkin test, followed by a prediction interval. Once the normality of the data was tested, we compared our algorithm's performance with other GP methods. The variances between methods were compared with f-tests, and means were compared with one-way t-tests.

Chapter 7

Results and Discussion

The initial objectives of this dissertation were to develop a classifier that was able to:

- Detect IC and TO events within 125ms.
- Be able to distinguish walking and non-walking activities, so no false events are detected.
- Function in real-time.
- Be robust to different environments such as slopes, stairs, load changes, etc.
- Not hinder the user's comfort or mobility.

From the three tested feature sets (l-mGAS+ r-mGAS, l-mGAS+l-TA, r-mGAS+r-TA), the best classification accuracy was seen when the algorithm was fed data from both mGAS muscles, so this combination of muscles was used throughout the rest of the experiments to test different combinations of hyperparameters. The optimized classifier's sensitivity, specificity and mean error obtained from 27 test trials is shown below in **Table 7.1**.

Table 7.1 – Results obtained from both the Real-time detection algorithm and the 30ms Offset prediction algorithm.

	Real-Time Detection		**30ms Offset Prediction**	
	IC	**TO**	**IC**	**TO**
Sensitivity	100%	100%	92%	100%
Specificity	87%	87%	65%	65%
Mean Error ±Std (ms)	-4 ± 24	6 ± 22	23 ± 34	-10 ± 28
Error Normality	Yes	Yes	Yes	Yes
p-value (α=0.05)	0.34	0.37	0.53	0.42

From this graph we can observe that the 30ms Offset algorithm performed worse than the Real-time detection algorithm in all metrics, as expected. To check if the algorithms' errors follows a normal distribution a histogram of the errors and a Shapiro-Wilk test was performed (α=0.10). The Shapiro-Wilk test returned p-value >α, so the null hypothesis cannot be rejected, meaning we can assume the errors to follow a normal distribution.

With the assumption that the errors follow a normal distribution, it is possible to calculate the prediction intervals for the algorithm's error using a t-student distribution with 26 degrees of freedom, as shown in **Table 7.2**.

Table 7.2 - Prediction Interval for both algorithms with 95%, 99% and 99.9% Confidence Intervals.

		95%	99%	99.9%
Real-Time Detection	***IC Prediction interval***	[-49,49]ms	[-67,67]ms	[-89,89]ms
	TO Prediction interval	[-39,51]ms	[-55,67]ms	[-74,88]ms
30ms Offset Prediction	***IC Prediction interval***	[-47,93]ms	[-71,117]ms	[-103,149]ms
	TO Prediction interval	[-58,58]ms	[-78,78]ms	[-103,103]ms

From **Table 7.2**, we can observe that the main objective of the real-time classifier – the detection of IC and TO within 125ms – was achieved, with prediction intervals comfortably inside the 125ms window. However, the secondary objective of being able to predict gait events 30ms before their occurrence showed unsatisfactory results, especially with the low specificity and prediction interval outside of the 125ms tolerance window, so for these reasons, we did not perform any further statistical tests for the prediction algorithm. The real-time classifier only requires the use of two surface EMG electrodes, one in each mGAS, so we can safely assume that the user's comfort and mobility are not affected. However, the algorithm detected multiple false positives, resulting in a low specificity. The robustness of the algorithm to different environments was not tested, due to a lack of data provided in those scenarios, and the online implementation of the algorithm was also not tested. Also, the algorithm's speed is very dependent on the hardware that it runs on. With our hardware (Nvidia Geforce GTX960M with 640 CUDA cores), a single sample is classified in 440µs. Since the sampling time of the inputs of the classifier is 2ms, it is reasonable to assume that an online implementation is possible.

Overall, the classifier shows promising results, with a relatively low mean error. However, the low specificity is far from ideal, and must be addressed. One method of doing so would consist in adopting a new loss function, which would heavily penalise incorrect classifications. Kidziński, Delp and Schwartz [90] utilised this approach in their paper, achieving a mean time delay of 18.3ms in the detection of 95% of IC events. A similar method could be used with our classifier to tackle this issue, such as using a weighted binary cross-entropy function that would heavily penalise incorrect predictions – such as a prediction outside a range of 125ms of an event. However, this process is not straightforward since the Keras API only allows custom loss function with 2 inputs, which would require the use of strategies like function closure to create the custom loss function. The algorithm could also achieve better results if provided with more data to train more effectively.

Next, we evaluated our classifier against other gait classifiers using EMG, as presented in **Table 7.3**.

Table 7.3 – Classification Accuracy and Mean errors for different EMG-based methods. Values in brackets were obtained with training data.

	Nazmi et al. [48]		**Lauer et al.** [3]		**Present Study**	
Classification Accuracy	77% (87.5)		- (97%)		96.5% (99.55%)	
	IC	**TO**	**IC**	**TO**	**IC**	**TO**
Mean error±Std (ms)	35 ± 25	49 ± 15	(18 ± 38)	(-12 ± 36)	-4 ± 24	6 ± 22

The next step is comparing out classifier's performance with other gait event detection methods, namely IMUs, as shown in **Table 7.4**.

Table 7.4 - Sensitivity, Specificity and Mean errors for IMU based method and the present study. Hyphens are placed where values where not specified in the study.

	Maqbool et al [44]		**Mannini et al.** [6]		**Present Study (2019)**	
	IC	**TO**	**IC**	**TO**	**IC**	**TO**
Sensitivity	100%	99.7%	-	-	100%	100%
Specificity	-	-	-	-	87%	87%
Mean error ±Std (ms)	22± 20	-8 ± 16	62 ± 47	36 ± 18	-4 ± 24	6 ± 22

To assess statistical significance, both t-test and f-tests were performed. The results are shown in **Figure 7.1**.

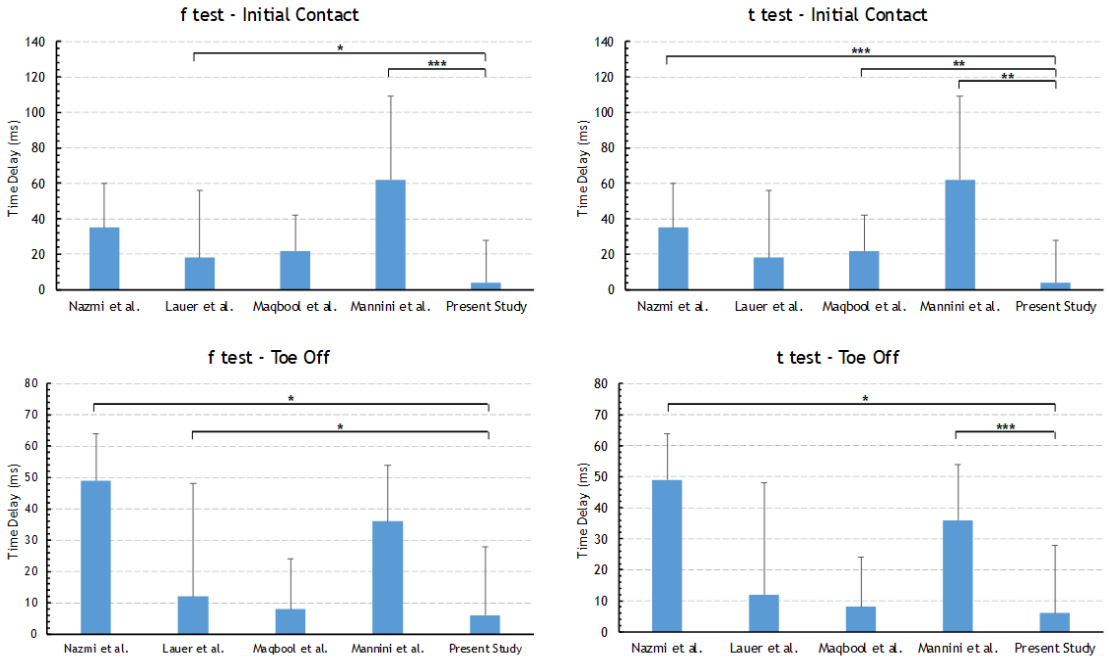

Figure 7.1 - f-test and Welsh t-test, for both IC and TO, between the Present Study and other models. One * indicates significant differences with α=0.05, two * indicates significant differences with α=0.01 and three * indicate significant differences with α=0.001

When comparing to Lauer et al., no significant differences were found between the mean errors (α=0.05). However, the f-tests show the variance in the present algorithm is significantly lower than in Lauer et al. (α=0.05). Therefore, we can safely say that our algorithm outperformed Lauer et al., even though this algorithm used training data to calculate errors.

When comparing to the other EMG-based algorithm, Nazmi et al., our algorithm showed significant lower mean errors for IC (α=0.001) and TO (α=0.05). However, the f-tests show that variance in the present algorithm is significantly higher than in Lauer *et al*. (α=0.05) for TO. Therefore, although our algorithm shows lower mean errors, it shows a higher SD, so we cannot affirm that our algorithm outperformed the Lauer *et al*.

When comparing to Mannini *et al.,* our algorithm's mean error showed significantly lower values for TO (α=0.001) and IC (α=0.01), as well as the IC SD (α=0.001). Therefore, we can safely say that our algorithm outperformed Mannini *et al*.

Finally, when comparing with Maqbool *et al.,* no significant differences were found in the SDs of both methods, although our algorithm showed a significantly lower mean error for IC detection (α=0.01).

Overall, the classifier shows promising results, with a relatively low mean error. When compared to other gait event detection methods, it performed better or similarly to other EMG based methods as well as with IMU based methods. However, the gait event prediction algorithm, which attempted to detect gait events 30ms before their occurrence was unsuccessful. Also, the low specificity of the real-time detection algorithm is far from ideal and must be addressed. One method of doing so would consist in adopting a new loss function, which would heavily penalise incorrect classifications. Kidziński, Delp and Schwartz [90] utilised this approach in their paper, achieving a mean time delay of 18.3ms in the detection of 95% of IC events. A similar method could be used with our classifier to tackle this issue, such as using a weighted binary cross-entropy function that would heavily penalise incorrect predictions – such as a prediction outside a range of 125ms of an event. However, this process is not straightforward since the Keras API only allows custom loss function with 2 inputs, which would require the use of strategies like function closure to create the custom loss function.

Better results could also be achieved if the algorithm had been provided with more data to train more effectively. More data would also allow the testing of the algorithm in different scenarios, such as slopes and stairs, and a using video systems instead of force plates would result in more labels so the algorithm could detect other gait events.

Chapter 8

Conclusion

Gait event detection is a gait analysis technique that has a lot of useful applications such as FES and MPC. However, these applications cannot rely on non-wearable systems, such as force plates and video-systems, due to their desired ubiquitous application. As a result, there has a been a surge in attempts to perform gait event detection with wearable systems such as IMUs, footswitches and EMG.

In this dissertation, we analysed state-of-the-art gait event detection methods using wearable systems and compared their advantages and disadvantages. We found that footswitches showed the best detection performance but are only able to detect 4 gait events and are impractical in real-world applications due to their extremely short lifespans. IMUs also presented good performance, are durable and can detect all 8 phases of gait. EMG, on the other hand, showed great potential for FES and MPC applications due to its ability to detect all 8 gait phases, as well as measure the user's effort level and give a proportional response.

However, researchers using EMG to detect gait events consistently achieved worse results than researchers using other methods. Nevertheless, the aim of this dissertation is to cement the potential of EMG to perform real-time GP for applications such as FES and MPC and develop a new machine learning algorithm to tackle this matter.

For this purpose, a novel Deep Learning Algorithm combining 2 LSTM and 2 DNN layers was developed using Tensorflow 1.13.1 and Python 3.7.3. Dropout rates of 0.4 and batch standardization were applied to each layer's output, except the final layer, and ADAM was used to train the algorithm with learning rate=0.0005. Xavier-Glorot Initialization was used to initialize the algorithm's parameters. The main goal of this algorithm was to detect IC and TO events within 125ms of the actual events.

Before EMG data was fed to the algorithm, a pre-processing step was applied. The EMG data was labelled according to force plate data, and the original 2000Hz signal was resampled at 500Hz. The absolute of the resampled signal was passed through a 2^{nd} order Butterworth LP filter with a cutoff frequency of 4Hz to obtain the envelope of the signal.

The algorithm received sequences of 180 samples of EMG data from both left and right mGAS muscles and classified the final sample as "Swing" or "Stance". IC was considered the transition from Swing to Stance and TO the transition from Stance to Swing.

The mean time delays between gait events and the algorithm's detection were -4 (± 24)ms for IC and 6 ± 22 for TO. 100% of gait events were detected by the algorithm, although only 87% of detected events corresponded to real events. In addition, the time delays fell well within a 125ms tolerance window with over 99.9% confidence level. When comparing our algorithm with state-of-the-art methods using EMG, our algorithm showed similar, if not better performance overall.

Still, more research is needed in order to test the robustness of the algorithm in different scenarios, and different methods, such as using a weighted loss function, could be tested to attempt to lower the FP rate. Additional research would also result in more data for the algorithm to train, which could lead to better training.

Overall, this dissertation was able to demonstrate that EMG is a useful tool for gait event detection, capable of achieving the same level of accuracy as IMU based methods, as well as establish the capabilities of LSTM networks for EMG classification.

References

[1] W. Tao, T. Liu, R. Zheng, and H. Feng, “Gait analysis using wearable sensors,” *Sensors*, vol. 12, no. 2, pp. 2255–2283, 2012.

[2] P. Cherelle *et al.*, “The Ankle Mimicking Prosthetic Foot 3—Locking mechanisms, actuator design, control and experiments with an amputee,” *Rob. Auton. Syst.*, vol. 91, pp. 327–336, May 2017.

[3] R. T. Lauer, B. T. Smith, and R. R. Betz, “Application of a Neuro-Fuzzy Network for Gait Event Detection Using Electromyography in the Child With Cerebral Palsy,” *IEEE Trans. Biomed. Eng.*, vol. 52, no. 9, pp. 1532–1540, Sep. 2005.

[4] J. Taborri, E. Palermo, S. Rossi, and P. Cappa, “Gait Partitioning Methods: A Systematic Review,” *Sensors*, vol. 16, no. 1, p. 66, Jan. 2016.

[5] G. Balbinot, C. P. Schuch, M. A. Zaro, and M. A. Vaz, “Low-cost piezoelectric footswitch system for measuring temporal parameters during walking,” *Int. J. Eng. Technol.*, vol. 3, no. 1, p. 75, Feb. 2014.

[6] A. Mannini, V. Genovese, and A. M. Sabatini, “Online decoding of hidden markov models for gait event detection using foot-mounted gyroscopes,” *IEEE J. Biomed. Heal. Informatics*, vol. 18, no. 4, pp. 1122–1130, 2014.

[7] U. Martinez-Hernandez, M. I. Awad, I. Mahmood, and A. A. Dehghani-Sanij, “Prediction of gait events in walking activities with a Bayesian perception system,” in *2017 International Conference on Rehabilitation Robotics (ICORR)*, 2017, vol. 2017, pp. 13–18.

[8] I. P. I. Pappas, T. Keller, S. Mangold, M. Popovic, V. Dietz, and M. Morari, “A Reliable Gyroscope-Based Gait-Phase Detection Sensor Embedded in a Shoe Insole,” *IEEE Sens. J.*, vol. 4, no. 2, pp. 268–274, Apr. 2004.

[9] D. P. Ferris, K. E. Gordon, G. S. Sawicki, and A. Peethambaran, “An improved powered ankle–foot orthosis using proportional myoelectric control,” *Gait Posture*, vol. 23, no. 4, pp. 425–428, Jun. 2006.

[10] D. P. FERRIS, G. S. SAWICKI, and M. A. DALEY, “A PHYSIOLOGIST’S PERSPECTIVE ON ROBOTIC EXOSKELETONS FOR HUMAN LOCOMOTION,” *Int. J. Humanoid Robot.*, vol. 04, no. 03, pp. 507–528, Sep. 2007.

[11] M. W. Whittle, “Clinical gait analysis: A review,” *Hum. Mov. Sci.*, vol. 15, no. 3, pp. 369–387, Jun. 1996.

[12] M. Hanlon and R. Anderson, “Real-time gait event detection using wearable sensors,” *Gait Posture*, vol. 30, no. 4, pp. 523–527, Nov. 2009.

[13] T. R. Farrell and R. F. Weir, “The Optimal Controller Delay for Myoelectric Prostheses,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 15, no. 1, pp. 111–118, Mar. 2007.

[14] A. Mansfield and G. M. Lyons, “The use of accelerometry to detect heel contact events for use as a sensor in FES assisted walking,” *Med. Eng. Phys.*, vol. 25, pp. 879–885, 2003.

[15] H. T. T. Vu, F. Gomez, P. Cherelle, D. Lefeber, A. Nowé, and B. Vanderborght, “ED-FNN: A New Deep Learning Algorithm to Detect Percentage of the Gait Cycle for Powered Prostheses.,” *Sensors (Basel).*, vol. 18, no. 7, Jul. 2018.

[16] D. H. Sutherland, “The evolution of clinical gait analysis. Part II kinematics.,” *Gait*

Posture, vol. 16, no. 2, pp. 159–79, Oct. 2002.
[17] W. E. H. Harcourt-Smith and L. C. Aiello, “Fossils, feet and the evolution of human bipedal locomotion,” *J. Anat.*, vol. 204, no. 5, pp. 403–416, May 2004.
[18] R. L. Susman, “Fossil evidence for early hominid tool use.,” *Science*, vol. 265, no. 5178, pp. 1570–3, Sep. 1994.
[19] C. L. Vaughan, B. L. Davis, and J. C. O’Connor, *Dynamics of Human Gait*, 2nd ed. Western Cape, South Africa: Kiboho Publishers, 1999.
[20] R. M. Kay, S. Dennis, S. Rethlefsen, R. A. K. Reynolds, D. L. Skaggs, and V. T. Tolo, “The Effect of Preoperative Gait Analysis on Orthopaedic Decision Making,” *Clin. Orthop. Relat. Res.*, vol. 372, no. 372, pp. 217–222, Mar. 2000.
[21] P. A. DeLuca, R. B. Davis, S. Ounpuu, S. Rose, and R. Sirkin, “Alterations in surgical decision making in patients with cerebral palsy based on three-dimensional gait analysis.,” *J. Pediatr. Orthop.*, vol. 17, no. 5, pp. 608–14.
[22] D. H. Sutherland, “The evolution of clinical gait analysis part III – kinetics and energy assessment,” *Gait Posture*, vol. 21, no. 4, pp. 447–461, Jun. 2005.
[23] I. Newton, *Philosophiae Naturalis Principia Mathematica*. 1687.
[24] N. Shiba *et al.*, “Biomechanical effect and clinical application of the hip joint moment reduction brace.,” *Clin. Orthop. Relat. Res.*, no. 351, pp. 149–57, Jun. 1998.
[25] S. A. Rose, S. Õunpuu, and P. A. DeLuca, “Strategies for the Assessment of Pediatric Gait in the Clinical Setting,” *Phys. Ther.*, vol. 71, no. 12, pp. 961–980, Dec. 1991.
[26] E. Criswell, *Cram’s Introduction to Surface Electromyography*, 2nd ed. Jones and Bartlett Publishers, 2011.
[27] D. H. Sutherland, “The evolution of clinical gait analysis part l: kinesiological EMG.,” *Gait Posture*, vol. 14, no. 1, pp. 61–70, Jul. 2001.
[28] J. R. Gage, J. Perry, R. R. Hicks, S. Koop, and J. R. Werntz, “Rectus femoris transfer to improve knee function of children with cerebral palsy.,” *Dev. Med. Child Neurol.*, vol. 29, no. 2, pp. 159–66, Apr. 1987.
[29] J. Perry and M. M. Hoffer, “Preoperative and postoperative dynamic electromyography as an aid in planning tendon transfers in children with cerebral palsy.,” *J. Bone Joint Surg. Am.*, vol. 59, no. 4, pp. 531–7, Jun. 1977.
[30] C. Fleischer, A. Wege, K. Kondak, and G. Hommel, “Application of EMG signals for controlling exoskeleton robots,” *Biomed. Tech. Eng.*, vol. 51, no. 5_6, pp. 314–319, Dec. 2006.
[31] M. N. Alam, A. Garg, T. T. K. Munia, R. Fazel-Rezai, and K. Tavakolian, “Vertical ground reaction force marker for Parkinson’s disease,” *PLoS One*, vol. 12, no. 5, p. e0175951, May 2017.
[32] K. Kaczmarczyk, A. Wit, M. Krawczyk, J. Zaborski, and J. Pisudski, “Artificial Neural Networks (ANN) Applied for Gait Classification and Physiotherapy Monitoring in Post Stroke Patients,” in *Artificial Neural Networks - Methodological Advances and Biomedical Applications*, InTech, 2011.
[33] R. T. Lauer, B. T. Smith, D. Coiro, R. R. Betz, and J. McCarthy, “Feasibility of Gait Event Detection Using Intramuscular Electromyography in the Child with Cerebral Palsy,” *Neuromodulation Technol. Neural Interface*, vol. 7, no. 3, pp. 205–213, Jul. 2004.
[34] C. M. Kim and J. J. Eng, “Magnitude and pattern of 3D kinematic and kinetic gait profiles in persons with stroke: relationship to walking speed.,” *Gait Posture*, vol. 20, no. 2, pp. 140–6, Oct. 2004.
[35] C. M. O’Connor, S. K. Thorpe, M. J. O’Malley, and C. L. Vaughan, “Automatic detection of gait events using kinematic data,” *Gait Posture*, vol. 25, no. 3, pp. 469–474, Mar. 2007.
[36] A. H. Hansen, D. S. Childress, and M. R. Meier, “A simple method for determination of gait events,” *J. Biomech.*, vol. 35, no. 1, pp. 135–138, Jan. 2002.
[37] J.-Y. Jung, W. Heo, H. Yang, and H. Park, “A Neural Network-Based Gait Phase Classification Method Using Sensors Equipped on Lower Limb Exoskeleton Robots.,” *Sensors (Basel).*, vol. 15, no. 11, pp. 27738–59, Oct. 2015.
[38] C. D. Joshi, U. Lahiri, and N. V. Thakor, “Classification of gait phases from lower limb EMG: Application to exoskeleton orthosis,” in *2013 IEEE Point-of-Care Healthcare*

Technologies (PHT), 2013, pp. 228–231.
[39] C. MacDonald, D. Smith, R. Brower, M. Ceberio, and T. Sarkodie-Gyan, "Determination of Human Gait Phase Using Fuzzy Inference," in *2007 IEEE 10th International Conference on Rehabilitation Robotics*, 2007, pp. 661–665.
[40] Xiaoli Meng, Haoyong Yu, and Ming Po Tham, "Gait phase detection in able-bodied subjects and dementia patients," in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2013, pp. 4907–4910.
[41] J. Bae and M. Tomizuka, "Gait phase analysis based on a Hidden Markov Model," *Mechatronics*, vol. 21, no. 6, pp. 961–970, Sep. 2011.
[42] K. De Pauw, P. Cherelle, B. Roelands, D. Lefeber, and R. Meeusen, "The efficacy of the Ankle Mimicking Prosthetic Foot prototype 4.0 during walking: Physiological determinants," *Prosthet. Orthot. Int.*, vol. 42, no. 5, pp. 504–510, 2018.
[43] J. Rueterbories, E. G. Spaich, B. Larsen, and O. K. Andersen, "Methods for gait event detection and analysis in ambulatory systems," *Med. Eng. Phys.*, vol. 32, no. 6, pp. 545–552, Jul. 2010.
[44] H. F. Maqbool *et al.*, "Real-time gait event detection for lower limb amputees using a single wearable sensor," in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2016, pp. 5067–5070.
[45] P. Müller, T. Seel, and T. Schauer, "Experimental Evaluation of a Novel Inertial Sensor Based Realtime Gait Phase Detection Algorithm," in *Proc. of the 5th European Conference on Technically Assisted Rehabilitation - TAR*, 2015.
[46] A. J. Young, T. A. Kuiken, and L. J. Hargrove, "Analysis of using EMG and mechanical sensors to enhance intent recognition in powered lower limb prostheses," *J. Neural Eng.*, vol. 11, no. 5, p. 056021, Oct. 2014.
[47] M. Z. ur Rehman *et al.*, "Multiday EMG-Based classification of hand motions with deep learning techniques," *Sensors (Switzerland)*, vol. 18, no. 8, pp. 1–16, 2018.
[48] N. Nazmi, M. A. Abdul Rahman, S.-I. Yamamoto, and S. A. Ahmad, "Walking gait event detection based on electromyography signals using artificial neural network," *Biomed. Signal Process. Control*, vol. 47, pp. 334–343, Jan. 2019.
[49] M. Meng, Q. She, Y. Gao, and Z. Luo, "EMG signals based gait phases recognition using hidden Markov models," in *The 2010 IEEE International Conference on Information and Automation*, 2010, pp. 852–856.
[50] E. C. Wentink, V. G. H. Schut, E. C. Prinsen, J. S. Rietman, and P. H. Veltink, "Detection of the onset of gait initiation using kinematic sensors and EMG in transfemoral amputees," *Gait Posture*, vol. 39, no. 1, pp. 391–396, Jan. 2014.
[51] I. Maglogiannis, K. Karpouzis, B. A. Wallace, and J. Soldatos, *Emerging Artificial Intelligence Applications in Computer Engineering*. IOS Press, 2007.
[52] E. Keogh and A. Mueen, "Curse of Dimensionality," in *Encyclopedia of Machine Learning and Data Mining*, Boston, MA: Springer US, 2017, pp. 314–315.
[53] S. Abe, "Feature Selection and Extraction," in *Support Vector Machines for Pattern Classification. Advances in Pattern Recognition.*, Springer, London, 2010, pp. 331–341.
[54] S. Tiwari, B. Singh, and M. Kaur, "An approach for feature selection using local searching and global optimization techniques," *Neural Comput. Appl.*, vol. 28, no. 10, pp. 2915–2930, Oct. 2017.
[55] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artif. Intell.*, vol. 97, no. 1–2, pp. 273–324, Dec. 1997.
[56] Narendra and Fukunaga, "A Branch and Bound Algorithm for Feature Subset Selection," *IEEE Trans. Comput.*, vol. C-26, no. 9, pp. 917–922, Sep. 1977.
[57] N. Chaikla and Yulu Qi, "Genetic algorithms in feature selection," in *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, vol. 5, pp. 538–540.
[58] C. Xu, "Common Methods for Feature Extraction: PCA and LDA," 2018. .
[59] R. A. FISHER, "THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS," *Ann. Eugen.*, vol. 7, no. 2, pp. 179–188, Sep. 1936.
[60] V. Y. Urbakh, "Linear Discriminant Analysis: Loss of Discriminating Power When a Variate is Omitted," *Biometrics*, vol. 27, no. 3, p. 531, Sep. 1971.
[61] T. Sing, O. Sander, N. Beerenwinkel, and T. Lengauer, "ROCR: visualizing classifier

performance in R," *Bioinformatics*, vol. 21, no. 20, pp. 3940–3941, Oct. 2005.
[62] E. Gokgoz and A. Subasi, "Comparison of decision tree algorithms for EMG signal classification using DWT," *Biomed. Signal Process. Control*, vol. 18, pp. 138–144, Apr. 2015.
[63] S. Abe, "Modified Backward Feature Selection by Cross Validation," *ESANN, Comput. Sci. Eng. (CSE), 2014 IEEE 17th Int. Conf.*, no. May, pp. 27–29, 2005.
[64] J.-W. Lee and G.-K. Lee, "Gait Angle Prediction for Lower Limb Orthotics and Prostheses Using an EMG Signal and Neural Networks."
[65] A. Phinyomark, F. Quaine, S. Charbonnier, C. Serviere, F. Tarpin-Bernard, and Y. Laurillau, "EMG feature evaluation for improving myoelectric pattern recognition robustness," *Expert Syst. Appl.*, vol. 40, no. 12, pp. 4832–4840, Sep. 2013.
[66] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
[67] "Understanding LSTM Networks -- colah's blog." [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 16-Jun-2019].
[68] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Comput.*, vol. 12, no. 10, pp. 2451–2471, Oct. 2000.
[69] P. Golik, P. Doetsch, and H. Ney, "Cross-Entropy vs. Squared Error Training: a Theoretical and Experimental Comparison," 2013.
[70] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Dec. 2014.
[71] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," 2014.
[72] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent Neural Network Regularization," Sep. 2014.
[73] M. B. I. Reaz, M. S. Hussain, and F. Mohd-Yasin, "Techniques of EMG signal analysis: detection, processing, classification and applications," *Biol. Proced. Online*, vol. 8, no. 1, pp. 11–35, Dec. 2006.
[74] E. N. Kamavuako, K. B. Englehart, W. Jensen, and D. Farina, "Simultaneous and Proportional Force Estimation in Multiple Degrees of Freedom From Intramuscular EMG," *IEEE Trans. Biomed. Eng.*, vol. 59, no. 7, pp. 1804–1807, Jul. 2012.
[75] L. H. Smith and L. J. Hargrove, "Comparison of surface and intramuscular EMG pattern recognition for simultaneous wrist/hand motion classification," in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, 2013, vol. 2013, pp. 4223–4226.
[76] "Seniam.org." .
[77] V. A. Kotel'nikov, "On the Transmission Capacity of the 'Ether' and Wire in Electrocommunications," in *Modern Sampling Theory*, Boston, MA: Birkhäuser Boston, 2001, pp. 27–45.
[78] U. Cote-Allard *et al.*, "Deep Learning for Electromyographic Hand Gesture Signal Classification Using Transfer Learning," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 27, no. 4, pp. 760–771, Apr. 2019.
[79] C. J. De Luca, L. Donald Gilmore, M. Kuznetsov, and S. H. Roy, "Filtering the surface EMG signal: Movement artifact and baseline noise contamination," *J. Biomech.*, vol. 43, no. 8, pp. 1573–1579, May 2010.
[80] D. T. Mewett, H. Nazeran, and K. J. Reynolds, "Removing power line noise from recorded EMG," in *2001 Conference Proceedings of the 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 3, pp. 2190–2193.
[81] "Bertec." [Online]. Available: https://www.bertec.com/. [Accessed: 15-Jun-2019].
[82] H. J. Hermens, B. Freriks, C. Disselhorst-Klug, and G. Rau, "Development of recommendations for SEMG sensors and sensor placement procedures.," *J. Electromyogr. Kinesiol.*, vol. 10, no. 5, pp. 361–74, Oct. 2000.
[83] N. Nazmi, M. A. Abdul Rahman, S. I. Yamamoto, and S. A. Ahmad, "Walking gait event detection based on electromyography signals using artificial neural network," *Biomed. Signal Process. Control*, vol. 47, pp. 334–343, 2018.
[84] "LABIOMEP – Porto Biomechanics Laboratory." [Online]. Available: https://labiomep.up.pt/. [Accessed: 16-Jun-2019].

[85] J. Sinclair, P. J. Taylor, J. Hebron, D. Brooks, H. T. Hurst, and S. Atkins, “The Reliability of Electromyographic Normalization Methods for Cycling Analyses.,” *J. Hum. Kinet.*, vol. 46, pp. 19–27, Jun. 2015.

[86] J. F. Yang and D. A. Winter, “Electromyographic amplitude normalization methods: improving their sensitivity as diagnostic tools in gait analysis.,” *Arch. Phys. Med. Rehabil.*, vol. 65, no. 9, pp. 517–21, Sep. 1984.

[87] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.”

[88] J. S. Bridle, “Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition,” in *Neurocomputing*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 227–236.

[89] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks.”

[90] Ł. Kidziński, S. Delp, and M. Schwartz, “Automatic real-time gait event detection in children using deep neural networks,” *PLoS One*, vol. 14, no. 1, p. e0211466, Jan. 2019.

Appendix A

Developed Code

A.1. Matlab code for EMG pre-processing and labelling

```
1.  %% Abrir dados
2.  clc
3.
4.  Right_Leg_HS=zeros(60,2);
5.  Left_Leg_HS=zeros(60,2);
6.
7.  Right_Leg_emg=zeros(81,1000);
8.  Left_Leg_emg=zeros(81,1000);
9.
10.
11. for ID=1:1 %1:81
12. clearvars -
    ex-
    cept ID_tick HS_count_R ID Right_Leg_steps Left_Leg_steps HS_control_ind_R          HS_c
    on-
    trol_ind_L Right_Leg_HS Left_Leg_HS janela_size Right_Leg_emg Left_Leg_emg       Right
    _Leg_TO Left_Leg_TO TO_control_ind_R TO_control_ind_L Right_Leg_HS  Left_Leg_HS
13.
14. dados = dir('Novos Dados');
15. name = {dados.name};
```

```
16.
17. if ID<=81
18.     ID_tick=[1,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,1,1,0
    ,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0];
19.
20.     if ID_tick(ID)==0
21.         emg = [45,46,47,48]; % Alternado!!!
22.     else
23.         emg = [41,42,43,44];
24.     end
25. end
26.
27. ficheiro = load (name{ID+2});
28. field = fieldnames(ficheiro);
29. field{1}
30.
31. file = ficheiro.(field{1});
32. analog = file.Analog;
33. labels = analog.Labels;
34. data = analog.Data;
35.
36. %% Retirar Nans
37.
38. for i=1:50
39.     canal(i,:) = data(i,~isnan(data(i,:)));
40. end
41. %% Dados Gerais (vetor t)
42. fs=analog.Frequency;
43. ts= 1/fs;
44. samples = size(canal,2)-1;
45. t= 0:ts:samples*ts;
46. t_rs= 0:ts*4:samples*ts;
47. t_deriv=linspace(0,(samples-1)*ts,(samples/4));
48.
49. %% Normalizar força e detetar eventos
50. plates = [3,9,15,21];
```

```
51. eventos=zeros(4,2);
52. thresh = 0.005; % Limite para baseline
53. 
54. for i=1:4 % Para cada force plate
55.     average_canal(i)=mean(canal(plates(i),:));
56.     index=find(canal(plates(i),:)<thresh); % Indíces para fazer a média apenas do baseline
57.     new=canal(plates(i),index);
58.     average_baseline=mean(new);
59.     sd(i)=std(new);
60.     if average_canal(i)>0.01  %Se o canal nao estiver "morto"
61.         evento = find(canal(plates(i),:) > (average_baseline+10*sd(i))); %Todos os pontos aci
    ma do baseline (+10DP)
62.         eventos(i,:) = [evento(1),evento(end)]; % Primeiro ponto acima e último ponto acima
     do baseline (+10DP)%evento TO
63.     end
64.     canal(plates(i),:)=canal(plates(i),:)-average_baseline;
65. end
66. 
67. %% Retificar EMG
68. 
69. retificado(:,:) = canal(emg,:)- mean(canal(emg,:),2);
70. absoluto = abs(retificado);
71. 
72. %% Filtrar O Retificado
73. fc=300;
74. 
75. [b,a]=butter(2,fc/(fs/2),'low');
76. filtrado = filter(b,a,retificado,[],2);
77. 
78. %% Resample de Retificado (500 HZ) e envelope
79. 
80. resampled = resample(filtrado',1,4);
81. resampled = abs(resampled');
82. frs = 500;
83. fc=4;
84. 
```

```
[b,a]=butter(2,fc/(frs/2),'low');

for i=1:4
    envel(i,:) = filtfilt(b,a,resampled (i,:));
end

%% Normalizar sinal
norm_max=max(envel,[],2);
norm_min=min(envel,[],2);
envel_norm=((envel-norm_min)./(norm_max-norm_min));

%% Janela
cutoff=300; % Cortar 300 samples (300*ts = 150ms)
eventos_ind= eventos(:,:) > 0; % Quando é que eventos é >0
eventos_nonzero=eventos(eventos_ind);
    eventos_nz = reshape(eventos_nonzero,[3,2]);

    janela_min=min(eventos_nonzero)-cutoff;
    janela_max=max(eventos(:,1))+cutoff*2;
    janela_ind=round((janela_min:4:janela_max)/4,0);
    janela = envel(:,janela_ind);
    janela_t=t_rs(janela_ind);
    janela_size(ID)=length(janela);

    Right_Leg_emg(ID,[1:length(janela)]) = janela(1,:);
    Left_Leg_emg(ID,[1:length(janela)]) = janela(2,:);

    %% Normalizar Janela
    norm_max=max(janela,[],2);
    norm_min=min(janela,[],2);
    janela_norm=((janela-norm_min)./(norm_max-norm_min));

    Right_Leg_emg(ID,1:length(janela_norm)) = janela_norm(1,:);
    Left_Leg_emg(ID,1:length(janela_norm)) = janela_norm(2,:);
    %% Detetar Eventos - norm.
```

```
120.    thresh1 = 0.56;
121.    gate=1;
122.
123.    for j=1:2
124.        count=1;
125.        for i=2:length(janela_ind)
126.            if janela_norm(j,i)>thresh1
127.
128.                if janela_norm(j,i)<janela_norm(j,i-1)
129.                    if gate==1
130.                        HS(j,count)=janela_t(1,i);
131.                        HS_ind(j,count) = i-1;
132.                        count = count+1;
133.                        gate=0;
134.                    end
135.                end
136.            else
137.                gate=1;
138.            end
139.        end
140.    end
141.    HS_count_R = length(nonzeros(HS(1,:)));
142.    HS_count_L = length(nonzeros(HS(2,:)));
143.    HS_control = eventos_nz(:,1)'*ts;
144.    HS_control_ind = round(((eventos_nz(:,1)-(janela_min-
    1))/4),0); % (HS_control_ind + (janela_min-1))*4 para transformar em tempo.
145.    TO_control_ind = round(((eventos_nz(:,2)-(janela_min-1))/4),0);
146.
147.    if HS_count_R ==2 && HS_count_L ==1
148.        HS_control_ind_R=[HS_control_ind(1),HS_control_ind(3)];
149.        TO_control_ind_R=[TO_control_ind(1),TO_control_ind(3)];
150.        HS_control_R=[HS_control(1),HS_control(3)];
151.        HS_control_ind_L=[HS_control_ind(2),0];
152.        TO_control_ind_L=[TO_control_ind(2),0];
153.        HS_control_L=[HS_control(2),0];
154.        fprintf('Right Foot two stomps')
```

```
155.    elseif HS_count_R ==1 && HS_count_L ==2
156.        HS_control_ind_R=[HS_control_ind(2),0];
157.        TO_control_ind_R=[TO_control_ind(2),0];
158.        HS_control_R=[HS_control(2),0];
159.        HS_control_ind_L=[HS_control_ind(1),HS_control_ind(3)];
160.        TO_control_ind_L=[TO_control_ind(1),TO_control_ind(3)];
161.        HS_control_L=[HS_control(1),HS_control(3)];
162.        fprintf('Right Foot one stomp')
163.    else
164.        fprintf('Something is Wrong!')
165.        HS_count_R =0;
166.        HS_control_ind_R = [0,0];
167.        TO_control_ind_R = [0,0];
168.        HS_control_R =[0,0];
169.        HS_count_L =0;
170.        HS_control_ind_L = [0,0];
171.        TO_control_ind_L = [0,0];
172.        HS_control_L =[0,0];
173.    end
174.
175.    %% Final Touches
176.
177.    Right_Leg_steps(ID,1)=HS_count_R;
178.    Left_Leg_steps(ID,1)=HS_count_L;
179.
180.    Right_Leg_HS(ID,:) = HS_control_ind_R;
181.    Right_Leg_TO(ID,:) = TO_control_ind_R;
182.    Left_Leg_HS(ID,:) = HS_control_ind_L;
183.    Left_Leg_TO(ID,:) = TO_control_ind_L;
184.
185.    end
186.    %% END
187.
188.    %% Final Vectors
189.
```

```
Right_Leg_vetor= zeros(81,max(janela_size));
Left_Leg_vetor= zeros(81,max(janela_size));

for i=1:81
    if Right_Leg_steps(i)==2
        Right_Leg_vetor(i,(Right_Leg_HS(i,1)-15):Right_Leg_TO(i,1)-16) = 1;
        Right_Leg_vetor(i,(Right_Leg_HS(i,2)-15):Right_Leg_TO(i,2)-16) = 1;
        Left_Leg_vetor (i,(Left_Leg_HS(i,1)-15):Left_Leg_TO(i,1)-16) = 1;
    elseif Right_Leg_steps(i)==1
        Right_Leg_vetor (i,(Right_Leg_HS(i,1)-15):Right_Leg_TO(i,1)-16) = 1;
        Left_Leg_vetor(i,(Left_Leg_HS(i,1)-15):Left_Leg_TO(i,1)-16) = 1;
        Left_Leg_vetor(i,(Left_Leg_HS(i,2)-15):Left_Leg_TO(i,2)-16) = 1;

    end

end

Right_Leg_emg = Right_Leg_emg(:,1:max(janela_size));
Left_Leg_emg = Left_Leg_emg(:,1:max(janela_size));
Right_Leg_vetor = Right_Leg_vetor(:,1:max(janela_size));
Left_Leg_vetor = Left_Leg_vetor(:,1:max(janela_size));

sprintf('Done!')
%% Write Files
canais_bons=1:81;
canais_bons(Right_Leg_steps(canais_bons)==0)=[];


% fid = fopen('RightLegEMG.txt','wt');
% for i = canais_bons
%     fprintf(fid,'%.4f, ',Right_Leg_emg(i,:));
%     fprintf(fid,';\n');
% end
% fclose(fid);
%
```

```
% fid = fopen('RightLegVector(-30ms).txt','wt');
% for i = canais_bons
%     fprintf(fid,'%d, ',Right_Leg_vetor(i,:));
%     fprintf(fid,';\n');
% end
% fclose(fid);
%
% fid = fopen('LeftLegEMG.txt','wt');
% for i = canais_bons
%     fprintf(fid,'%.4f, ',Left_Leg_emg(i,:));
%     fprintf(fid,';\n');
% end
% fclose(fid);
%
% fid = fopen('LeftLegVector(-30ms).txt','wt');
% for i = canais_bons
%     fprintf(fid,'%d, ',Left_Leg_vetor(i,:));
%     fprintf(fid,';\n');
% end
% fclose(fid);
```

A.2. Python code for training and testing the Deep Learning algorithm

```
# Imports
import glob
import numpy as np
from collections import deque
import pandas as pd
import random
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, CuDNNLSTM
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.callbacks import ModelCheckpoint
```

```
12.     from tensorflow.keras.utils import to_categorical
13.
14.     # STEP 1 - IMPORTING DATA INTO DATAFRAME
15.
16.     def prepare_data(filename): # Reads filename and returns them in array. Array is a lis
    t where each element
17.
18.         array = []
19.         f = open(filename, "r")
20.         lines=f.read().split(', ;\n')           # Defines that ', ;\n' indicates the end of a line.
21.         del(lines[-1])
22.         for line in lines:
23.             ele-
    ments=(line.split(', '))         # Defines that each element in a line is divided by ', '.
24.             elements=list(map(float, elements))  # Stores the values as float
25.             array.append(elements)
26.         f.close
27.         return array
28.
29.
30.     txt_files = [i for i in glob.glob('*.txt')] # Reads all .txt files in current directory (create
    d in Matlab)
31.     print(txt_files)
32.     del txt_files[2]                        # Delete non-important files from txt_files
33.     del txt_files[2]
34.     del txt_files[4]
35.     del txt_files[4]
36.
37.     print(txt_files)
38.
39.     ar-
    ray_all = []                                # Create empty list that will store all arrays
40.     for filename in txt_files:
41.         ar-
    ray_all.append(prepare_data(filename))              # In array_all, store all arrays returned
     from prepare_data
42.
43.     df = pd.DataFrame()                              # Create an empty dataframe
```

```
44.
45.     for i in range(len(array_all[0])):                    # Append the elements in array_all
46.         if df is None:                                    # The result is a Dataframe with 292 collu
    mns
47.             df = df.append([array_all[0][i], array_all[1][i]])  # Collumns 0-
    146 alternate between LeftLegGM and Left Leg Vector
48.         else:
49.             df = df.append([array_all[0][i], array_all[1][i]])
50.
51.     for i in range(len(array_all[0])):
52.         df = df.append([array_all[2][i], array_all[3][i]])      # Collumns 147-
    292 alternate between RightLegGM and Right Leg Vector
53.
54.     # STEP 2 - CREATING SEQUENCES
55.
56.     SEQ_LEN = 180                   # Defines the length of the sequences.
57.     sequential_data = []             # This is a list that will contain the sequences
58.     se-
    quence = deque(maxlen=SEQ_LEN)    # These will be our actual sequences. They are made
     with deque, which keeps the maximum length by popping out older values as new ones c
    ome in
59.     sequence2= deque(maxlen=SEQ_LEN)    # Sequence2 will contain the second feature.
60.
61.     ind=(np.arange(0,292,2))          # Vector with indices of collumns with EMG data
62.     for i in ind:                   # Create the sequences. Append them in sequential_data
63.         count=0
64.         for n in df.values[i]:
65.             sequence.append([n])
66.             if i<145:
67.                 sequence2.append([df.values[i+146][count]])
68.             else:
69.                 sequence2.append([df.values[i-146][count]])
70.             if len(sequence) == SEQ_LEN:                    # make sure sequences have leng
    th 180.
71.                 if sum(sequence[-1]+sequence[-2]+sequence[-
    3])==0: # Don't Append sequences that end in 3 zeros, since EMG channels
72.                     continue
73.                 else:
```

```
74.                     sequen-
    tial_data.append([np.array(sequence),np.array(sequence2),df.values[i+1][count]])  # App
    end a label with each pair of features.
75.             count=count+1
76.         sequence = []
77.         sequence = deque(maxlen=SEQ_LEN)
78.         sequence2 = []
79.         sequence2 = deque(maxlen=SEQ_LEN)
80.
81.     print(len(sequential_data))
82.
83.     # STEP 3 TEST AND TRAIN SPLIT, BALANCING TRAIN DATA
84.     test=int(len(sequential_data)*0.2)  # Define A
85.     print('Train:', test)
86.     train_data=sequential_data[test:]   # Rest of values for training
87.     test_data=sequential_data[:test]    # First A values for testing
88.
89.
90.     Stance = []  # list that will store our buy sequences and targets
91.     Swing = []   # list that will store our sell sequences and targets
92.
93.     for i in train_data:                            # iterate over the train_data
94.         if i[2] == 0:                                   # if it's a "not stance"
95.             Swing.append([i[0], i[1],i[2]])       # append to swing list
96.         elif i[2] == 1:                              # if it's a "stance"
97.             Stance.append([i[0], i[1],i[2]])     # append to swing list
98.
99.     lower = min(len(Stance), len(Swing))      # what's the shorter length?
100.
101.    Stance = Stance[:lower]     # make sure both lists are only up to the shortest length.
102.    Swing = Swing[:lower]       # make sure both lists are only up to the shortest length.
103.    train_data = Stance+Swing   # add them together
104.
105.    random.shuffle(train_data)  # shuffle, so mod-
    el doesn't get confused with all 1 class then the other.
106.    print('Db:',len(train_data))
107.
```

```
108.
109.     X_train = []
110.     X_train_temp=[]
111.     y_train = []
112.     X_test = []
113.     X_test_temp=[]
114.     y_test = []
115.
116.     # STEP 4 - DIVIDE TRAIN_DATA AND TEST_DATA INTO SEQUENCES AND LABELS
117.     for i in train_data:
118.         for element in range(SEQ_LEN):                          # going over our new train_data
119.             X_train_temp.append(np.array([i[0][element], i[1][element]]))       # X is the seq
    uences
120.         X_train.append(X_train_temp)
121.         X_train_temp=[]
122.         y_train.append(i[2])                                    # y is the targets/labels (swing/s
    tance)
123.     X_train = np.squeeze(np.array(X_train))
124.
125.
126.     for i in test_data:                                         # going over test_data
127.         for element in range(SEQ_LEN):
128.             X_test_temp.append(np.array([i[0][element], i[1][element]]))       # X is the seq
    uences
129.         X_test.append(X_test_temp)
130.         X_test_temp=[]
131.         y_test.append(i[2])                                     # y is the targets/labels (swing
    /stance)
132.     X_train = np.squeeze(np.array(X_train))
133.     X_test = np.squeeze(np.array(X_test))
134.
135.
136.     # PART 5 - THE CLASSIFIER
137.     gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.75)
138.     sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))
139.
140.     EPOCHS = 150          # how many passes through our data
```

```
141.    BATCH_SIZE = 500       # how long are the batches? Smaller batch
142.                           # if getting OOM (out of memory) errors.
143.    learning_rate=0.0005   # define Learning Rate
144.    dropout=0.4            # Define droupout rate
145.    NAME = f"LSTM-(Adam-30)-128-64 layers- dropout0.4 -
    Ultimo"  # a unique name for the model
146.
147.    model = Sequential()    # Create the Model, add layers sequentially
148.    model.add(CuDNNLSTM(128, input_shape=(SEQ_LEN,2), return_sequences=True))
149.    model.add(Dropout(dropout))
150.    model.add(BatchNormalization())
151.
152.    model.add(CuDNNLSTM(64))
153.    model.add(Dropout(dropout))
154.    model.add(BatchNormalization())
155.
156.    model.add(Dense(32, activation='relu'))
157.    model.add(Dropout(dropout))
158.    model.add(BatchNormalization())
159.
160.    model.add(Dense(2, activation='softmax'))
161.
162.    #Choose optimizer
163.    opt = tf.keras.optimizers.Adam(learning_rate)
164.
165.    # Compile model
166.    model.compile(
167.        loss='categorical_crossentropy',  # Choose Loss
168.        optimizer=opt,                  # Tell model which opt to use
169.        metrics=['accuracy']            # Tell model which metric to use
170.    )
171.
172.    tensorboard = TensorBoard(log_dir="logs/{}".format(NAME))
173.
174.    # unique file name that will include the epoch and the validation acc for that epoch
175.    filepath = "LSTM_Adam-32-32layers-ultimo{epoch:02d}-{val_acc:.3f}"
```

```
176.    checkpoint = ModelCheckpoint("models/{}.model".format(filepath,
177.                                                          monitor='val_acc',
178.                                                          verbose=1,
179.                                                          save_best_only=True,
180.                                                          mode='max'))
181.
182.    y_trainb = to_categorical(y_train)
183.    y_testb = to_categorical(y_test)
184.
185.    # Train model
186.    history = model.fit(    # Batchsize and Epochs were define previously.
187.        X_train, y_trainb,
188.        batch_size=BATCH_SIZE,
189.        epochs=EPOCHS,
190.        validation_data=(X_test, y_testb),
191.        callbacks=[tensorboard, checkpoint],
192.    )
193.
194.    # Score model
195.    score = model.evaluate(X_test, y_testb, verbose=0)
196.    print('Test loss:', score[0])
197.    print('Test accuracy:', score[1])
198.    # Save model
199.    model.save("models/{}".format(NAME)) # Save the model
```

A.3. Python code for calculating mean event detection time delays, standard deviation and testing normality of time delay

```
1.  import os
2.  import glob
3.  import numpy as np
4.  from collections import deque
5.  import pandas as pd
6.  import tensorflow as tf
7.  from tensorflow.keras.models import Sequential
```

```
8.  from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, CuDNNLSTM
9.  from tensorflow.keras.utils import to_categorical
10. from scipy.stats import shapiro
11. #from playsound import playsound
12. #import matplotlib.pyplot as plt
13. from matplotlib import pyplot
14.
15.
16. cwd = os.getcwd()
17. SEQ_LEN = 180
18.
19. def prepare_data(filename):
20.
21.     array = []
22.     f = open(filename, "r")
23.     lines=f.read().split(', ;\n')
24.     del(lines[-1])
25.
26.     for line in lines:
27.         elements=(line.split(', '))
28.         elements=list(map(float, elements))
29.         array.append(elements)
30.     f.close
31.
32.     return array
33.
34.
35. txt_files = [i for i in glob.glob('*.txt')]
36. del txt_files[1]
37. del txt_files[1]
38. del txt_files[3]
39. del txt_files[3]
40.
41. array_all = []
42. for filename in txt_files:
```

```
43.     array_all.append(prepare_data(filename))
44. print(txt_files)
45. 
46. #print ((np.shape(array_all)))
47. #print (np.shape(array_all[0]))
48. #print (np.shape(array_all[1]))
49. 
50. df = pd.DataFrame() # begin empty
51. 
52. for i in range(len(array_all[0])):
53.     if df is None:  # if the dataframe is empty
54.         df = df.append([array_all[0][i], array_all[1][i]])  # then it's just the current df
55.     else:  # otherwise, join this data to the main one
56.         df = df.append([array_all[0][i], array_all[1][i]])
57. for i in range(len(array_all[0])):
58.     df = df.append([array_all[2][i], array_all[3][i]])
59. 
60. #df=df.T
61. 
62. #print (df.values)
63. 
64. sequential_data = []  # this is a list that will CONTAIN the sequences
65. se-
    quence = deque(maxlen=SEQ_LEN)  # These will be our actual sequences. They are made
    with deque, which keeps the maximum length by popping out older values as new ones co
    me in
66. sequence2 = deque(maxlen=SEQ_LEN)
67. length=[]
68. ind=(np.arange(0,292,2)) #146
69. 
70. 
71. for i in ind:  # iterate over the values
72.     count=0
73.     count2=0
74.     for n in df.values[i]:
75.         sequence.append([n])  # store all but the target
76.         if i<145:
```

```
77.             sequence2.append([df.values[i+146][count]])
78.         else:
79.             sequence2.append([df.values[i-146][count]])
80.         if len(sequence) == SEQ_LEN:  # make sure we have 180 long sequences!
81.             if sum(sequence[-1]+sequence[-2]+sequence[-
    3])==0: # Don't Append sequences that end in 3 zeros
82.                 continue
83.             else:
84.                 sequen-
    tial_data.append([np.array(sequence),np.array(sequence2),df.values[i+1][count]])
85.                 count2=count2+1
86.         count=count+1
87.     length.append(count2)
88.     sequence = []
89.     sequence = deque(maxlen=SEQ_LEN)
90.     sequence2 = []
91.     sequence2 = deque(maxlen=SEQ_LEN)
92.
93. length_cum=np.cumsum(length)
94.
95. ## DIVIDIR EM TEST E TRAIN
96. train=len(sequential_data)
97. print(len(sequential_data))
98. test_data=sequential_data[:train]
99.
100.     X_test = []
101.     X_test_temp=[]
102.     y_test = []
103.
104.     for i in test_data:  # going over our new sequential data
105.         for element in range(SEQ_LEN):
106.             X_test_temp.append(np.array([i[0][element], i[1][element]]))       # X is the seq
    uences
107.         X_test.append(X_test_temp)
108.         X_test_temp=[]
109.         y_test.append(i[2])      # y is the targets/labels
110.     X_test = np.squeeze(np.array(X_test))
```

```
111.
112.    y_testb = to_categorical(y_test)
113.
114.    #Load model
115.    learning_rate=0.0005
116.    dropout=0.4
117.    model = Sequential()
118.    model.add(CuDNNLSTM(128, input_shape=(X_test.shape[1:]), return_sequences=True)
119.    model.add(Dropout(dropout))
120.    model.add(BatchNormalization())  #normalizes activation outputs, same reason
    #                                    why we want to normalize your input data.
121.    model.add(CuDNNLSTM(64))
122.    model.add(Dropout(dropout))
123.    model.add(BatchNormalization())
124.
125.    model.add(Dense(32, activation='relu'))
126.    model.add(Dropout(dropout))
127.    model.add(BatchNormalization()) error
128.
129.    model.add(Dense(2, activation='softmax'))
130.
131.
132.    #Choose optimizer
133.    opt = tf.keras.optimizers.Adam(lr=learning_rate) #Optimizer
134.    # Compile model
135.    model.compile(
136.        loss='categorical_crossentropy',
137.        optimizer=opt,
138.        metrics=['accuracy']
139.    )
140.
141.    path= 'LSTM_Adam-32-32layers-ultimo14-0.956.model'      # Choose model to run
142.    model.load_weights(cwd+'\models\\'+path)              # Load parameters
143.
144.    #print(model.summary())
145.
```

```
146.
147.    # Show prediction
148.    Y_pred = model.predict([X_test], verbose=1)
149.
150.    pred_list=[]
151.    for pred in Y_pred:
152.        if pred[0]>0.9:
153.            pred[0]=0
154.        else:
155.            pred[0]=1
156.        pred_list.append(pred[0])
157.
158.
159.    vetor_temp=[]
160.    vetor=[]
161.    prediction_temp=[]
162.    prediction=[]
163.    for trial in range(146):
164.        if trial==0:
165.            index=np.arange(0,length_cum[trial])
166.    #        print(trial)
167.        else:
168.            index=np.arange(length_cum[trial-1],length_cum[trial])
169.        for i in index:
170.            if pred_list[i] != y_test[i]:
171.                vetor_temp.append(pred_list[i])
172.            else:
173.                vetor_temp.append(pred_list[i])
174.            prediction_temp.append(pred_list[i])
175.        vetor.append(vetor_temp)
176.        vetor_temp=[]
177.        prediction.append(prediction_temp)
178.        prediction_temp=[]
179.
180.    HS_temp=[]
```

```
TO_temp=[]
HS_control_temp=[]
TO_control_temp=[]
HS=[]
TO=[]
HS_control=[]
TO_control=[]
diferenca=[]
FP=0
needle = [0,1]
for j in range(0, 146):
    for i in range(len(needle),len(prediction[j])):
        if prediction[j][i-1]==0 and prediction[j][i]==1:
            HS_temp.append(i)
        if prediction[j][i-1]==1 and prediction[j][i]==0:
            TO_temp.append(i)
        if df.values[ind[j] + 1][i+179-1]==0 and df.values[ind[j] + 1][i+179]==1:
            HS_control_temp.append(i)
        if df.values[ind[j] + 1][i+179-1]==1 and df.values[ind[j] + 1][i+179]==0:
            TO_control_temp.append(i)
    HS.append(HS_temp)
    TO.append(TO_temp)
    HS_temp=[]
    TO_temp=[]
    HS_control.append(HS_control_temp)
    TO_control.append(TO_control_temp)
    HS_control_temp=[]
    TO_control_temp=[]
    if j<27:
        print('Trial:',j, HS[j],HS_control[j], np.array(HS[j]) - (HS_control[j]), TO[j],
TO_control[j],np.array(TO[j]) - (TO_control[j]))
        diferenca.append(np.array(TO[j]) - TO_control[j])
        FP=FP+len(diferenca[j])-1


for x in range(len(diferenca)):
```

```
216.        if len(diferenca[x]) == 0:
217.            diferenca[x]=30
218.        else:
219.            diferenca[x]= min(diferenca[x], key=abs)
220.
221.    print('Abs delay= ', np.mean(diferenca[:27]), 'Std=', np.std(diferenca[:27]), 'FP=',FP)
222.    #playsound('effect.mp3')
223.    stat, p = shapiro(diferenca[:27])
224.    print('Statistics=%.3f, p=%.3f' % (stat, p))
225.    pyplot.hist(diferenca[:27])
226.
227.    pyplot.show()
```