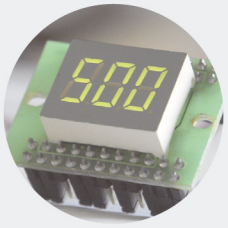
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Sistema baseado em inteligência artificial para deteção do estado de locomoção de uma pessoa

CAROLINA COSTA DE JESUS DA SILVA
novembro de 2022

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

System based on Artificial Intelligence for detecting the state of locomotion of a person

Carolina Costa de Jesus da Silva

Master in Electrical and Computer Engineering
Specialization Area of Automation and Systems

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

November, 2022

This dissertation partially satisfies the requirements of the Thesis/Dissertation course of the program Master in Electrical and Computer Engineering, Specialization Area of Automation and Systems.

Candidate: Carolina Costa de Jesus da Silva, No. 1170654,
1170654@isep.ipp.pt

Scientific Guidance: Luiz Faria, lef@isep.ipp.pt

Scientific Co-Guidance: Arcelina Marques, Lino Figueiredo,
mmr@isep.ipp.pt,lbf@isep.ipp.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

November, 2022

I would like to dedicate this dissertation to all my family , colleagues and professors who accompanied me throughout my academic journey.

Acknowledgements

First, I would like to thank my parents, Zulmira Silva and Pedro Silva, and my brother Rui Pedro Silva for having accompanied me in all the important stages of my life and for always believing in me. Also to my family for their constant support and love.

To "SMART-HEALTH-4-ALL – Smart medical technologies for better health and care" project (POCI-01-0247-FEDER-046115; LISBOA-01-0247-FEDER-046115), which was co-financed by Portugal 2020, under the Operational Program for Competitiveness and Internationalization (COMPETE 2020) through the European Regional Development Fund (ERDF).

I especially thank my supervisors Professor Luiz Faria, Professor Arcelina Marques, and Professor Lino Figueiredo, for their constant availability, attention, knowledge, and understanding/trust placed in me.

To my colleagues, Rui Ferreira, and Pedro Martins, for their help in granting the tests carried out with the different participants and for sharing their knowledge.

To INEGI and its collaborators for providing the treadmill and participants to carry out the tests.

To all my colleagues and friends who supported me in carrying out this project.

Abstract

Currently, there are more and more systems using *Artificial Intelligence* (AI), since one of its characteristics is to provide the ability for systems to work in a similar way to a human. The development of various technologies [1] [2] such as sensors, *Internet-of-Things* (IoT), data analysis, etc. allows the application of AI through the existence of a large number of data [3]. Artificial Intelligence is used in several applications, one of the main ones being healthcare [4].

In this perspective, the project proposal presented in this dissertation aims at using Artificial Intelligence to classify a user's motion activities (slow or fast gait), with the data obtained through a developed plantar pressure measurement system.

The developed project can be divided into three parts, the first being the development of the plantar pressure monitoring system, the second the development of the AI models, and the third the processing of the input data of the models (datasets). Two datasets from different systems were used, the first being obtained through a public repository and the second obtained through the monitoring system developed for the project. The purpose of using the two datasets was to later compare the accuracy of the models for input data from different systems.

At the end of the project, when analyzing the accuracy values of each of the models for the two datasets, it was possible to conclude that, with the data from the monitoring system developed for the project, greater accuracy of the classification model was obtained than with the data from the public dataset.

Keywords: Artificial Intelligence, dataset, accuracy.

Resumo

Atualmente, existem cada vez mais sistemas que utilizam Inteligência Artificial, pois uma das suas características de funcionamento é proporcionar a capacidade dos sistemas funcionarem de forma semelhante a um ser humano. O desenvolvimento de várias tecnologias [1] [2] como sensores, Internet-of-Things (IoT), análise de dados, etc. permite a aplicação de Inteligência Artificial (IA) através da existência de um grande número de dados [3]. A Inteligência Artificial é utilizada em diversas aplicações, sendo uma das principais a área da saúde [4].

Nesta perspetiva, a proposta de projeto apresentada nesta dissertação visa utilizar Inteligência Artificial para classificar as atividades de movimento de um utilizador (marcha lenta ou rápida), com os dados obtidos através de um sistema de medição de pressão plantar desenvolvido.

O projeto desenvolvido pode ser dividido em três partes, sendo a primeira o desenvolvimento do sistema de monitorização de pressão plantar, a segunda o desenvolvimento dos modelos de IA e a terceira o processamento dos dados de entrada dos modelos (*datasets*). Foram utilizados dois *datasets* de sistemas diferentes, sendo o primeiro obtido através de um repositório público e o segundo obtido através do sistema de monitorização desenvolvido para o projeto. O objetivo de usar os dois *datasets* foi comparar posteriormente a precisão dos modelos para dados de entrada de sistemas diferentes.

No final do projeto, ao analisar os valores de precisão de cada um dos modelos para os dois *datasets*, foi possível concluir que, com os dados do sistema de monitorização desenvolvido para o projeto, obteve-se maior precisão do modelo de classificação do que com os dados do *dataset* público.

Palavras-Chave: Inteligência Artificial, *dataset*, precisão.

Contents

List of Figures

List of Tables

List of Acronyms

1D	*One-Dimensional*
2D	*Two-Dimensional*
3D	*Three-Dimensional*
AGI	*Artificial General Intelligence*
AI	*Artificial Intelligence*
ANI	*Artificial Narrow Intelligence*
ANN	*Artificial Neural Networks*
API	*Application Programming Interface*
BLE	*Bluetooth Low Energy*
BOSS	*Bag-of-SFA-Symbols*
BW	*Butterworth*
CNN	*Convolutional Neural Networks*
COTE	*Collective of transform-based ensembles*
CPU	*Central Processing Unit*
CSV	*Comma-Separated Values*
DFT	*Discrete Fourier transforms*
DL	*Deep Learning*
DNN	*Deep Neural Network*
DTW	*Dynamic Time Warping*
ESN	*Echo State Networks*
FFT	*Fast Fourier Transforms*
FSR	*Force Sensitive Resistor*

FT	*Fourier transforms*
GPIOTE	*General purpose input/output tasks and events*
HCP	*Health Cluster Portugal*
I2S	*Serial Bus Interface*
IIR	*Infinite Impulse Responde*
IMU	*Inertial measurement unit*
IoT	*Internet-of-Things*
LED	*Light-Lmitting Diode*
LSTM	*Long Short-Term Memory*
ML	*Machine Learning*
MLP	*Multi Layer Perceptrons*
MTS	*M-dimensional time series (MTS)*
NARX	*Nonlinear Autoregressive Exogenous Model*
PCA	*Principal component analysis*
PCB	*Printed circuit board*
PPI	*Programmable Peripheral Interface*
RNN	*Recurrent Neural Network*
SH4ALL	*Smart-Health-4-All*
SPI	*Serial Peripheral Interface*
TSC	*Times Series Classification*
TWI	*Two Wire Interface*
UART	*Universal Asynchronous Receiver/transmitter*
USB	*Universal Serial Bus*
WDT	*Watchdog timer*

Chapter 1

Introduction

This chapter provides a context to the theme presented in this dissertation, as well as its objectives, and contribute to the project in which it is inserted, complementing with planning and organization of this dissertation.

1.1 Context and Motivation

Over the last few years, it has been witnessed the constant evolution of Artificial Intelligence, which has changed the way people work and live. According to a McKinsey Global Institute study on the use of *Artificial Intelligence* (AI) in different organizations, it is estimated that AI could generate additional economic savings of around $13 trillion by 2030 [5]. AI is already creating a tremendous amount of value in the software industry, however, a lot of the value that it will be created in the future will come from other fields, such as retail, transportation, automotive, healthcare, etc [5]. In this follow-up, this document aims to present a system based on AI that is included in the healthcare area.

One of the lines of the *Smart-Health-4-All* (SH4ALL) project, promoted by the *Health Cluster Portugal* (HCP) involves the development of an instrumented shoe with two components:

- **Sensor component**: which consists in a set of integrated sensors to measure plantar pressures, dorsal pressures, temperature and humidity;

- **Actuator component**: which is used to make the compensation of excessive pressures, circulation stimulation, activity alerts, etc.

As part of the project, an Android mobile application was developed to receive information from the shoe's sensors to monitor pressure levels and activate the actuation devices existing in the shoe. The actuation orders must be adapted to the shoe user's profile and according to pressure levels, considering clinical recommendations.

The AI-based system developed aims to use the data collected by the monitoring system for further analysis and detection of a person's locomotion status.

1.2 Objectives

The main objective of this study was the development of an AI model for classifying a person's state of locomotion, through their plantar pressure data. In order to achieve the general objective described above, the following specific objectives were required:

- Development of a monitoring system that records the user's plantar pressure values. The monitoring system consists of 9 piezoresistive sensors placed in the plantar region of the foot;
- Processing of data obtained by the monitoring system to create a dataset;
- The design, implementation, and training of a model based on machine learning to detect the state of locomotion of a person, being the model training performed with the plantar data collected by the shoe monitoring system;

1.3 Project Schedule

To ensure that all project objectives were achieved, all goals were defined and planned (Figure 1.1), which are:

- Development of the hardware to be used for the monitoring system;
- Development of the software for the functioning of the monitoring system as expected;
- Public dataset processing;
- Trials with participants for data collection with the developed monitoring system;
- Processing of the dataset formulated through data from trials with participants;

- Preparation of the dissertation.

It is important to mention that since the monitoring system is part of one of the developments of the SH4ALL project, the timing of its development was considered so that to ensure that the AI models were developed and tested with real data, it was used a public dataset that later also served for comparison with a dataset obtained through the monitoring system after its finishing.

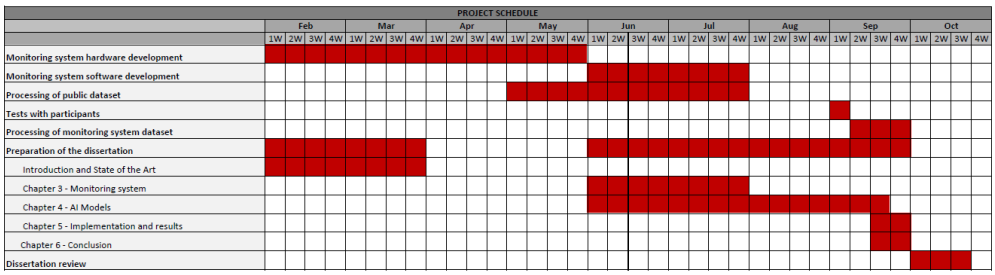


Figure 1.1: Project scheduling

1.4 Organization of dissertation

This dissertation is organized into six chapters, in which the first chapter describes the main motivations and also provides a contextualization of the developed project. Finally, the objectives of the work are characterized, as well as the system requirements.

In the second chapter, a study is carried out on Artificial Intelligence and human gait and its phases, finally giving some examples of projects developed in the same context as this project.

The third chapter addresses the development of the plantar pressure monitoring system, referring to the components that are part of the hardware and also referring to the flowcharts that represent the development of the code for recording plantar pressure data for later use in laboratory tests.

The fourth chapter describes all the data processing that took place for two types of datasets, one public and the other obtained through tests with the monitoring system developed. Additionally, three AI models developed to classify the type of human gait through plantar pressure data are described.

In the fifth chapter, the results obtained with each of the datasets are discussed, for each AI model developed, and finally, in chapter six, a conclusion of the entire project is made, where possible improvements are mentioned.

Chapter 2

State of the art

Since the main focus of this dissertation is the use of Artificial Intelligence in healthcare, this chapter aims to introduce AI, as well as several concepts that are related to it. In addition, a study of human gait and the phases that constitute it is carried out, concluding with some examples of systems previously developed for motion detection and classification of gait phases.

2.1 Artificial Intelligence (AI)

Nowadays, Artificial Intelligence can be associated with several terms, however, AI can be divided into two types: *Artificial Narrow Intelligence* (ANI), normally used to be applied in specific areas, such as agriculture, industry, etc., and *Artificial General Intelligence* (AGI) that is based on doing anything that a human can do [6]. Of these two types of AI, almost no progress has been made in AGI since it is difficult to create an application that can do everything that a human can do, for example, a robot that works in a hairdresser can perform some typical haircuts, however, it is not possible to guarantee that it will be able to perform all the haircuts that any client wants. In the field of ANI, there have been developments and the growth of AI is largely due to a field called *Machine Learning* (ML) [7].

Different approaches defend that ML can be defined, as coaching computers to perform different tasks, and to do these tasks the computer has to learn about the environment in which is inserted through repeated examples [8] [9]. Different ML methods can be used, depending on the nature of the data to be analyzed, which

are divided into supervised learning, unsupervised learning, and semi-supervised learning [9]. The most common type of ML is supervised learning which consists of mapping certain inputs corresponding to certain outputs and the learning of this method is executed through already known samples, in which the outputs are already labeled. In unsupervised learning, only inputs are used to train the system. Semi-supervised learning is a combination of the last two methods, where part of the data is partially labeled and these data are used to infer the remaining part of the data that is not labeled [9]. These three approaches are shown in Figure 2.1.

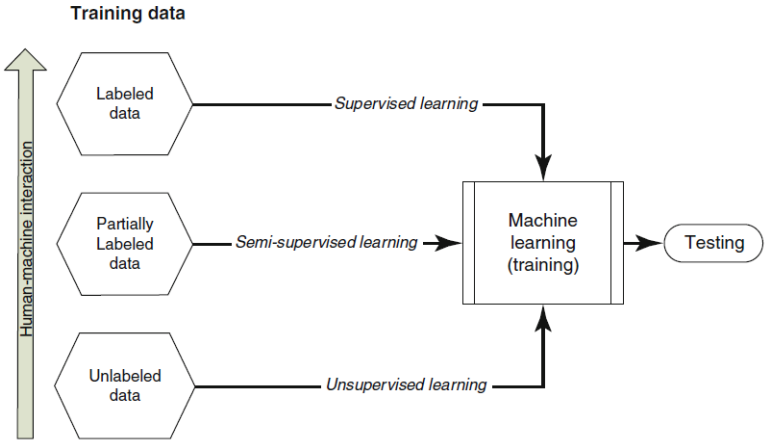


Figure 2.1: Categories of ML algorithms [9]

Some examples of day-to-day ML applications are the creation of algorithms to detect malicious emails, recommend content on online shopping sites, translate, etc.

In this way, ML requires a large amount of data for the created algorithms to work as expected. If these data are not of quality, the performance of the intended task will be compromised, which in turn requires the extraction and selection of the data to be used [10]. These limitations are overcome through *Deep Learning* (DL), which is a method that independently performs the extraction and selection of data to be used and then uses the collected data to perform their classification and regression.

For the creation of algorithms in DL, the so-called *Artificial Neural Networks* (ANN) are used. By its name, the ANNs are associated with the neural networks present in the human being, the human brain is constituted by neurons, and as such the ANN present a similar structure to neurons, Figure 2.2, helping the algorithm to make the right decisions and obtain the intended outputs. This method is divided into different layers [10] [11] (Figure 2.2), one of which is the input layer, which represents the type of data to be used, another is the output layer, which represents the information to be obtained when using the neural network and finally the intermediate layers, which represent the processing of data carried out through software. The entire neural network can be represented by a mathematical equation.

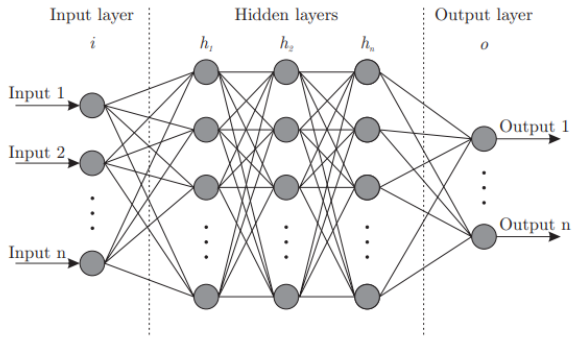


Figure 2.2: ANN example [11]

Currently, DL has been increasingly used, namely for medical image analysis applications, speech recognition, etc. In the field of electronics, the development of systems based on various types of sensors (which represent a network of sensors) contributes to the collection of a large volume of data (Big Data) [12]. Access to a large volume of data has its advantages, one of which is the ease of training the network to implement, however, there are also some challenges related to the search for patterns (data mining) and also data processing due to its wide range, collection speed, etc. [12].

In the healthcare area, the systems developed to monitor certain activities have the characteristic of obtaining a large volume of data along with time records. The introduction of these times brought a challenge called *Times Series Classification* (TSC) [13].

2.1.1 Time Series classification

TSC can be defined as a sequence of data obtained during a given time [**?**]. The use of TSC allows the classification of data to be carried out through the time recorded together with data from other variables, for example, in the case of the pressure monitoring system used in this project, the time when the pressure values of the sensors are registered allows the analysis and classification of the type of gait in which the user is. The TSC can be used through the *Deep Neural Network* (DNN), and there are complex models specifically developed for the same [**?**] [14].

There are, however, other important concepts to take into account, they are univariate time series, *M-dimensional time series (MTS)* (MTS), and datasets. Univariant time series can be defined as a set of ordered real values, whereas an MTS uses a set of different univariate times series [13]. A dataset is a set of pairs (X, Y), where X can be a univariate or M-dimensional times series and Y is a vector of size equal to the number of classes to use [13].

Traditional approaches

In 1978, an approach was devised in the context of TSC for speech recognition, called *Dynamic Time Warping* (DTW) [15]. This technique makes it possible to determine the distance between two non-punctual time series, using a mapping between similar points that are defined as time-invariant [15]. Some authors concluded that two of the techniques that showed the best conclusions were *Bag-of-SFA-Symbols* (BOSS) and*Collective of transform-based ensembles* (COTE) [15]. The BOSS approach was inspired by the bag-of-words model applied in text processing. This model is normally oriented towards the representation of the content of an image, where typically it starts by identifying local spots represented by vectors [16]. These vectors are identified as key points that are then quantified and organized in a histogram of visual words allowing their classification [16]. The COTE approach, on the other hand, consists of the use of 35 classifiers, structured in time, frequency, change, etc. that perform the processing of different types of time series [17].

DNN models

Currently, this TSC problem has been increasingly studied, and many researchers have even proposed several methods to solve it. An interesting detail regarding many of these methods is that several studies were carried out that found the non-use of DL models for their formulation [13] [18]. To perform the TSC task with DL, there are specific frameworks with a structure similar to that shown in Figure 2.3. As mentioned earlier, a DNN is made up of several layers, which in turn are made up of neurons that handle the necessary processing to get the output element [13]. In this follow-up, the most recent approaches to TSC are presented below, describing the corresponding architectures.

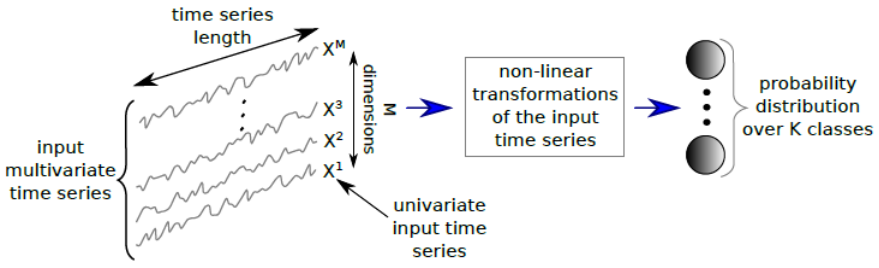


Figure 2.3: A DL framework for TSC [13]

- ***Multi Layer Perceptrons* (MLP)**

  An MLP presents the traditional architecture of DNN models, where the neurons of a certain layer connect with the neurons of the previous and next layer, therefore it can also be defined as a fully connected network [13]. Typically,

the construction of this network is carried out in three or more layers, where the first is the input layer that contains all the variables to be considered, the second or more hidden layers constituted by neurons for computation, and the output layer that contains a node for each desired class [19], Figure 2.4.

In Figure 2.4, an example of an MLP used for speech recognition is presented, where the input data are organized in a pattern and the neurons present in the network do the processing/calculation in the layers that follow until reaching the nodes of the output layer, which in turn performs the classification of the data [19]. The behavior of the network is also defined by the weight of each connection, these weights being later adjusted through training a supervised algorithm [13] [19] [20].

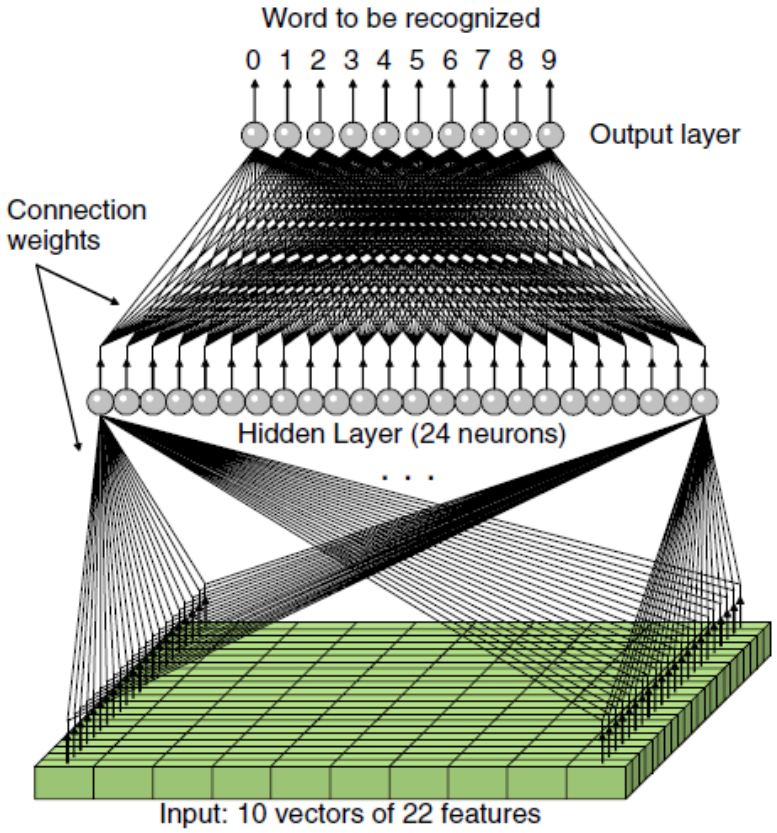


Figure 2.4: Example of an MLP structure [20]

- ***Convolutional Neural Networks* (CNN)**

  CNN's are based on the aforementioned approach, MLP, however, the way of dividing the weight of the various connections is different, which reduces the complexity of the network and also the number of weights to consider [13] [21] [22] [23]. A CNN is structured to use *Two-Dimensional* (2D) input data and each layer of the network consists of several 2D planes, which makes these

networks widely used in terms of image recognition. The planes are in turn composed of several independent neurons, which are connected to layers above and below but are not connected to each other [23].

In this context, a convolution presents a process similar to the application of a filter along the time series to analyze [13]. Figure 2.5 shows a simplified structure of a CNN. In this version, two convolutional layers (C1 and C2) and two subsampling layers (S1 and S2) are used. Using the image recognition example, when applying the CNN to it, three trained filters are applied to the input image (convolution layer C1), which is then added to a vector [23]. Still, in convolution layer 1, three resource maps are generated, where each of the localized regions is weighted, which generates three new resource maps in layer S1 [23]. For layers C2 and S2, the process is similar to the previous one, however, the output of layer S2 is vectorized to later serve as input for the training neural network [23].

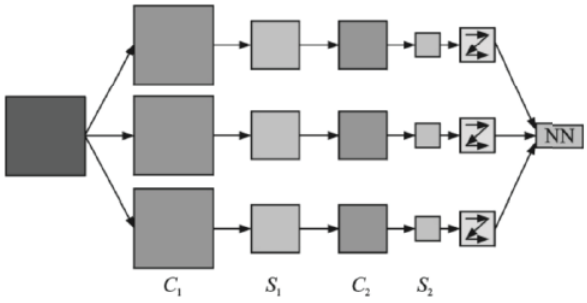


Figure 2.5: Simplified example of an CNN structure [24]

- ***Echo State Networks*** **(ESN)**

  ESN comes from DL models called *Recurrent Neural Network* (RNN). RNN networks have characteristics that distinguish them from the others, one of them being the ability to use input data referring to past events [24]. Several researchers have found some limitations of this type of network, some of which are: a limited number of input variables, architecture designed to obtain an output for each element (timestamp), difficulty in training the network, etc. [13] [24]. To respond to these problems, ESN emerged that sought to reduce the training time of the network, using dispersed random connections in a hidden layer called a *reservoir*, and the only parameters to consider are the output weights [13] [25]. Inside the *reservoir*, some neurons will be activated from the arrival signal [13], Figure 2.6 represents the architecture of an ESN for TSC.

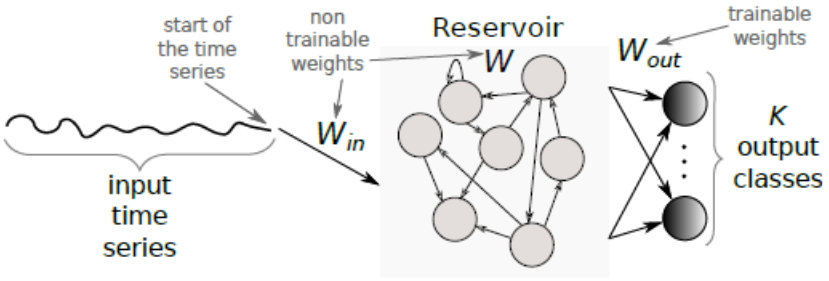


Figure 2.6: ESN architecture for TSC [13]

2.2 AI in gait analysis

AI has several areas of application, being healthcare one of the most recurrent. In the field of medicine, nowadays, AI contributes to disease diagnosis, medical care, biomedical information processing, etc [26]. Since the project presented is an application of AI to detect the state of locomotion of a user, it can be integrated into the healthcare area, where a study on human gait is carried out.

2.2.1 The gait

Although walking is a habitual activity for humans, a deeper analysis of gait can be performed since walking is a periodic movement that activates different segments of the body [27][28].

Visually analyzing the gait cycle of the human being, it can be divided into different phases. These phases were identified through the different movements generated in the segments [28]. A gait cycle begins with the initial contact of the heel of one foot on the ground and ends with the next contact of the heel of the same foot, which completes a gait cycle (two steps - one each foot) [27]. The gait cycle has two phases: swing and stance (Figure 2.7). The stance phase starts with the initial contact of one foot and ends with the toe-off of the same foot, lasting around 60% of the normal gait cycle [27][28][29]. The swing phase represents movements in which the foot is not on the ground and lasts approximately 40% of the normal gait cycle [27][28][29]. Figure 2.7, shows the two phases addressed and the specific movements of each one, as well as a brief description [29].

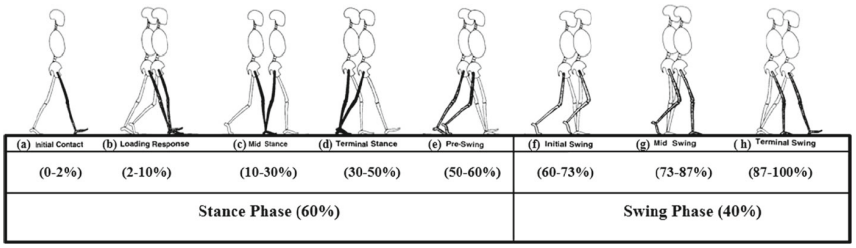


Figure 2.7: Phases of human gait and respective durations [28]

- **Initial Contact** - moment when the foot first touches the ground, through heel contact.
- **Loading Response** - phase in which the weight of the body is delivered to the front limb.
- **Mid Stance** - represents the interval in which the body weight is distributed on the foot contact area, from the rear foot to the forefoot. At the same time, the other foot begins to lift off the ground.
- **Terminal Stance** - the moment when the heel begins to lift off the ground.
- **Pré-Swing** - the moment when the forefoot is lifting off the ground and the heel of the other foot is already on the ground, gaining balance. The stance phase ends with toe-off (60%).
- **Initial Swing** - the phase when the foot lifts off the ground (acceleration phase).
- **Mid Swing** - maximum knee flexion.
- **Terminal Swing** - vertical positioning of the tibia (deceleration phase).

Currently, there is already a diversity of technological systems developed for the study of human gait and beyond. Many of these systems are mainly based on wearable sensors so that it is possible to collect real data from the user. The most used sensors for these studies are divided between inertial sensors and piezoresistive sensors [30]. Several studies demonstrate the use of inertial sensors for gait analysis, as they can detect variations in speed and acceleration. The main examples of these types of sensors are accelerometers and gyroscopes, and some devices combine these two sensors in an only, they are the so-called *Inertial measurement unit* (IMU) [31].

The IMUs allow access to many important variables for the characterization of a person's movement in addition to speed and acceleration, some of them being the number of steps, activity time, balance, cadence, length of a step, distance covered, etc. [32]. In some cases is also possible to determine the angle of the joint along the

movement of the person, being able to then identify the different phases of the gait through the variation of this value [30].

Piezoresistive sensors are normally used as pressure sensors that are then integrated into a shoe (for example, in an insole), and the pressure distribution on the foot is then studied in the different phases of gait [30].

2.2.2 AI in the study of the human gait

The existence of motion detection systems and the study of human gait via the collection of data from sensors makes it possible to apply AI in a form of prevention and data analysis. There are already projects developed where DL is used to classify human gait types, based on several sensors integrated into a smart insole. In a previous study, it was used a commercial smart insole, 'FootLogger', that has incorporated an array of eight pressure sensors, an array for the three-axis accelerometer, and an array for a three-axis gyroscope [33]. In this system, the authors defined different levels for the measured pressure intensity (0,1 or 2), which allowed identifying the swing and stance phases [33]. The measured values were sent via Bluetooth to an Android application.

Subsequently, the authors organized the values/variables they received into two matrices, one for the pressure sensors of each foot and another for the accelerometer and gyroscope of each foot, these matrices were then converted to vectors [33]. The organization of these variables aimed to use them in a DNN. Several networks were then created, Figure 2.8, one for each existing array, after which each network was trained individually [33].

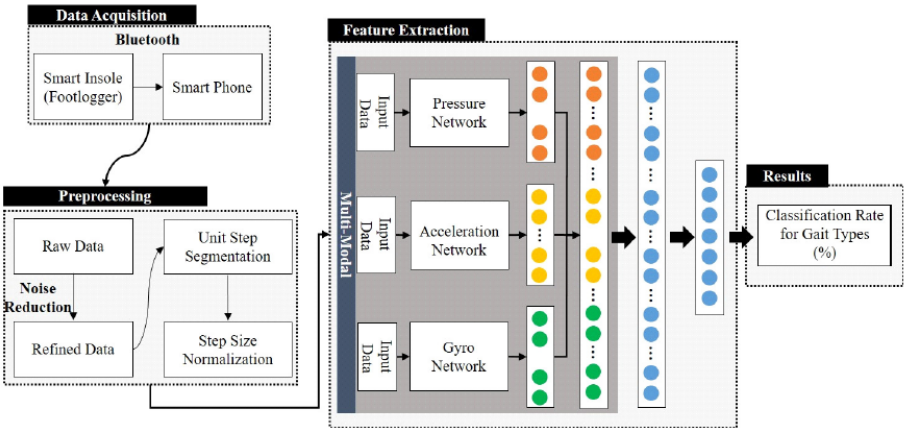


Figure 2.8: Structure of the method to classify gait types [34]

From Figure 2.8, it is possible to observe that the network consists of several layers, the first being the data obtained by the system (input data), which comes in vectors that were previously processed, the second is a layer called by *Convolutional*

layer that applies a set of filters according to the number of steps to be analyzed [33]. The neural network was trained to create a feature map for each sensor and this map is then converted to a vector that is used as input to a fully connected neural network [33]. The final layer is called *Output* which in turn represents all the classes to be classified.

Another study developed to classify gait phases consisted of the use of a lower limb exoskeleton robot, Figure 2.9, equipped with two IMU sensors and eight *Force Sensitive Resistor* (FSR) sensors to detect forces generated when the feet make contact with the ground [34]. The robot is attached to the user's lower limbs to carry out the tests. In this study, eight force sensors are used to perform gait classification.

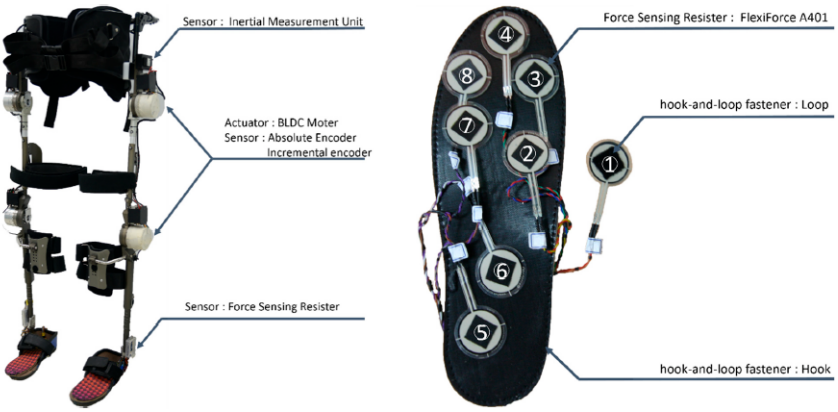


Figure 2.9: Robot and insole used for testing [35]

As in the previous study, the authors pre-process the acquired data before implementing them in the neural network, in this case, this was done using Matlab. This processing is important to identify which data correspond to the activity to be analyzed, for example, the authors remove some data referring to the beginning and end of the test, which correspond to a phase in which the user is standing still [34]. For training the neural network, supervised learning was used, in which, as mentioned in subchapter 2.1, there is a set of input data and a set of intended data for the output [34]. In this case, input parameters like pitch orientations, and angular velocities of each leg were used, being these preventive data from the IMUs coupled to the robot. In addition to this information, the force values measured by the FSR sensors were analyzed, where it was defined that if more than 5 kg of force were measured on the z-axis, the user would be considered to be in the stance phase [34].

Two distinct methods of neural networks were used, one of them being MLP and the other *Nonlinear Autoregressive Exogenous Model* (NARX). MLP networks have been addressed previously and they are normally used for pattern recognition and classification [34] [35]. The NARX networks, on the other hand, are normally used in dynamic systems applications, having limited prediction capabilities [34] [36].

In another research article, the authors followed another approach for recognizing gait phases, using inertial sensors, which were positioned on the user's foot and legs, and IMU's sensors which were handheld devices [37]. Once again, before the data are used in the neural network, they were pre-processed, since a great diversity of data is obtained, which can result in greater complexity in the neural model. For data processing, the *Principal component analysis* (PCA) method was used, which is a data analysis and dimensionality tool [37] [38].

For the input data in the neural network, vectors related to standard deviation, mean, maximum values, minimum values, etc. were organized into a single input vector to improve recognition accuracy. The construction of the neural network was based on a DNN, which can be defined as a set of ML algorithms that use inputs in the form of models with multiple layers (one input layer, an output layer, and several hidden layers) [39]. It may happen that some sub-neuronal networks do not draw the right conclusions, which will lead to a bad classification of the final data [40], since the output of the neuronal network is influenced by the outputs of the sub-neuronal networks that constitute it [37]. In this way, the authors proposed the construction of a structure for a general neural network, Figure 2.10, which allows each node to make different decisions, thus increasing the capacity of the network. This network consists of three DNNs, all three having an input layer and two hidden layers in common.

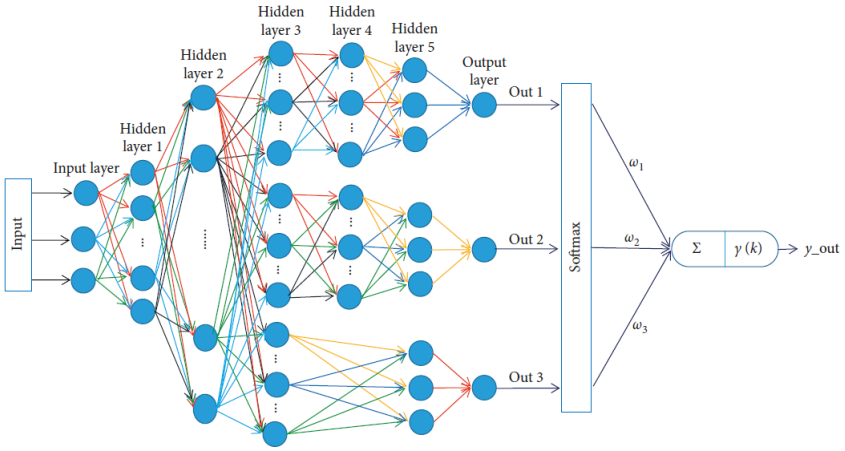


Figure 2.10: Proposed structure of neural network diagram [38]

2.3 Conclusion

After an introduction to what AI is and how it fits into the health area, referring to the traditional approaches of the types of networks used, as well as the systems present on the market, the monitoring system for data collection was developed, which is described in the next chapter.

Chapter 3

Monitoring system

This chapter is intended to present the first phase of the project, which consisted of the creation and development of a data acquisition system of plantar pressures. The acquisition system can be divided into three components, in which the first is the heart of the system, the microcontroller, which through its programming will perform the reading of analog channels and will convert the values obtained into pressure values (kPa) and in turn will send them via serial port. The second component is the plantar pressure sensors that will respond to the pressure applied to them. The third and last component is the communication via *Universal Serial Bus* (USB) to the computer. The diagram in Figure 3.1 shows the interconnection of all these system components.

Figure 3.1: General architecture of the pressure monitoring system

The hardware used to build the monitoring system is described below, as well as electric scheme and the flowcharts corresponding to the system programming (Firmware) accompanied by a detailed description.

3.1 Hardware

The components that make up the monitoring system were chosen through the SH4ALL project since there were requirements to be fulfilled. Since the monitoring system to be built for this project would not need all the components of the SH4ALL project monitoring system, only a few components were used for what was intended. The monitoring system can then be summarized into three components: the microcontroller, a multiplexer, and a sensing insole.

3.1.1 Microcontroller

The microcontroller represents the core of the monitoring system since it will be from there that the readings of the pressure sensors are carried out and these values are sent via *Bluetooth Low Energy* (BLE). In this way, it was necessary to make a responsible choice taking into account the requirements of the project. The microcontroller used was the nRF52832 (Figure 3.2), which includes the latest version of BLE 5.3 and also a varied set of peripherals [41] [42]. It supports various wireless protocols in addition to BLE and is built around an Arm Cortex-M4 *Central Processing Unit* (CPU) with floating point unit [41] [42]. It also allows the use of a set of digital interfaces such as *Serial Peripheral Interface* (SPI), *Two Wire Interface* (TWI), *Serial Bus Interface* (I2S), *Universal Asynchronous Receiver/transmitter* (UART), etc [42].

Contains 8 analog pins that can have a resolution of 8, 10, or 12-bits and as peripherals, it has 4 32-bit timers/counters, 2 24-bit real-time counters, *General purpose input/output tasks and events* (GPIOTE), a temperature sensor, *Watchdog timer* (WDT), *Programmable Peripheral Interface* (PPI), etc.

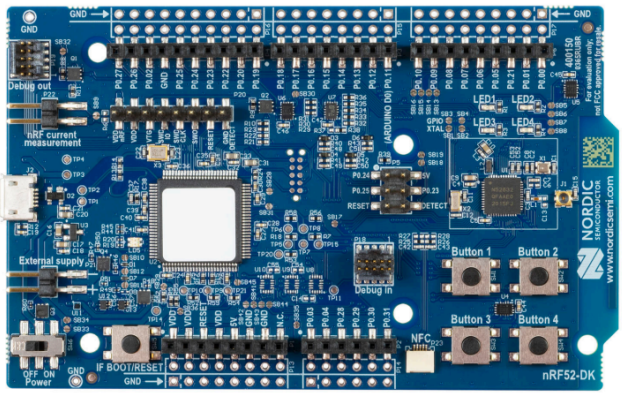


Figure 3.2: Board nrf52832 [43]

3.1.2 Pressure Sensors

Since the main objective of the SH4ALL project is the detection of areas of the foot with the potential risk of ulceration, the number of sensors to be used, as well as their location, were important parameters to be defined for the subsequent construction of the monitoring system.

The risk of ulceration is usually higher in diabetic patients since this problem is associated with constant pressure in a certain place [43] [44], without them realizing it, due to their lack of sensitivity. There are already some studies that quantify typical human plantar pressures, with values ranging from 100 kPa to 500 kPa in the Hallux region, approximately 700 kPa in the first metatarsal and 950 kPa in the second metatarsal [45] [46] [47] [48]. Given this variety of values, it is inappropriate to define a pressure threshold value, from which the diabetic patient may be at risk of ulceration, however, some authors define that this threshold can vary between 500 kPa and 1000 kPa [44] [49] [43], according to the foot area.

According to the indications for the diabetic foot, several areas of the foot where the risk of ulceration is high are identified, Figure 3.3, so these were the areas chosen for the placement of pressure sensors. Thus, in the final solution, the need to place 9 sensors in the plantar region was foreseen, as indicated in Figure 3.3. In the plantar region, two sensors were placed in the area of the phalanges of the first and fifth toes (sensors 1 and 2, in Figure 3.3), three sensors on the heads of the first, second/third and fourth/fifth metatarsals (sensors 3, 4 and 5) and three sensors located in the heel region (sensors 7, 8 and 9).

Figure 3.3: High risk zones for ulceration of the plantar region (left) and location of sensors in the plantar regions (right)

In this way, 9 pressure sensors were used in total, which led to the use of a 16-channel multiplexer, Figure 3.4, to read the sensors since the nRF52832 board has only 8 analog pins.

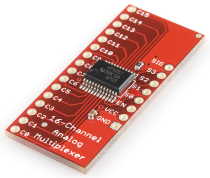


Figure 3.4: 16 channel multiplexer used [51]

The sensors used for pressure measurement were produced using a piezoresistive ink PR2NPr and a conductive extensible silver ink Sink01NPr [50] by the company Nanopaint Lda. The sensors were printed on a textile substrate through a screen printing process on a double-sided insole, that is, on one side of the insole it was printed with piezoresistive ink and on the other face it was printed with conductive silver ink, Figure 3.5 (developed within the scope of the SH4ALL project). After printing the insole, it was integrated into the insole of the shoe to be used for the tests, as shown in Figure 3.6.

Figure 3.5: Insole constructed for later integration into the diabetic shoe

Figure 3.6: Integration of insole into the diabetic shoe

In chapter one, it was mentioned that the sensor component that is part of the instrumented shoe consists of a set of sensors positioned in the plantar area of the foot and the dorsal area of the foot. Just as the sensorized insole was obtained, the sensorized upper was also developed, Figure 3.7 (developed within the scope of the SH4ALL project), however by manually integrating the instrumented upper in the shoe, as was done for the insole, Figures 3.8 and 3.9, it was noticed that the shoe did not have its initial structure, which would compromise later tests. In this way, only pressure data from the plantar region were collected.

Figure 3.7: Dorsal sensors constructed for later integration into the diabetic shoe

Figure 3.8: Integration of dorsal sensors in the shoe

Figure 3.9: Integration of dorsal sensors in the shoe

3.1.3 Electric Scheme

When interconnecting all the hardware components, it was necessary to define a way to position the system on the user's leg, for later tests. To make the system more comfortable to use and to avoid the use of wires that can be disconnected as the user walks, a *Printed circuit board* (PCB) was developed. This PCB is a board (Figures 3.10 and 3.11) that fits directly into the microcontroller and has all the necessary connections for the system to function, thus avoiding the use of wires.

It is important to point out that this PCB was developed not only for the tests of this project, but also for the SH4ALL project and therefore includes some connections that were not used for the tests of this work. The PCB has a connector for the multiplexer and another for all plantar sensors plus upper sensors, it also contains a connector for an IMU and two other connectors to support a battery charger. Additionally, it contains a connector for a digital potentiometer.

For this project, only the connections referring to the multiplexer, the pressure sensors, and the digital potentiometer were used. However, the digital potentiometer component was not used, being replaced by a resistor of fixed value (39 kOhm), since, when carrying out different tests of reading the analog value of certain channels of the multiplexer, it was detected noise in ports that were not being subjected to any voltage and it was concluded that this noise was caused by the digital potentiometer.

In Figure 3.10, the electrical schematic of the PCB plugged into the microcontroller is shown. Connectors Conn_A, Conn_B, Conn_C, Conn_D and Conn_E, are the pins that connect the PCB to the pins of the nrf52832. The *Palmilha* and *Gaspea* connectors are for connecting the sensorized upper-shoe and insole. The Conn_carregador connectors are for the charger and the Conn_IMU connector fits the IMU pins and has the connections with connector Conn_C, for connection to the microcontroller. Still in Figure 3.10, the connections of the multiplexer (U1) and the potentiometer (U2) to the microcontroller are shown.

Figures 3.11 and 3.12 shows the front and back of this PCB, which fits into the microcontroller. And for an overview of the entire system, Figure 3.13 shows the PCB already coupled to the microcontroller and it is inside a box (printed on a *Three-Dimensional* (3D) printer), so that the tests could be carried out with the system attached to the user's leg.

A Velcro tape was also added to wrap around the participant's leg the system, to secure the 3D box and to not interfere with the person's gait. It was also considered the position of the board with the USB input facing away from the user's foot so that the USB cable would not interfere with the activity. Then the code was developed to program the board for reading and recording the plantar pressure values.

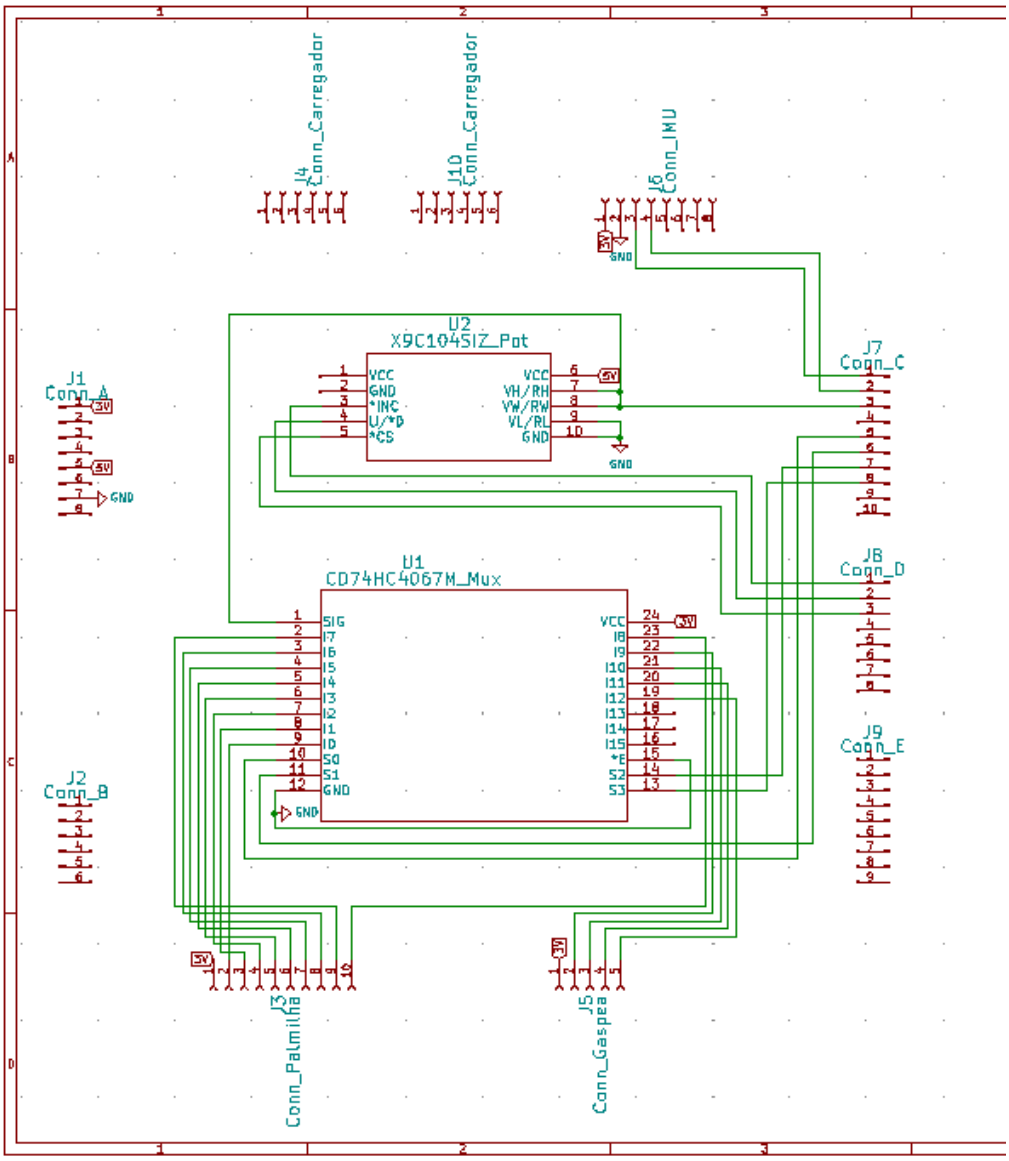


Figure 3.10: Electrical scheme of the pressure monitoring system

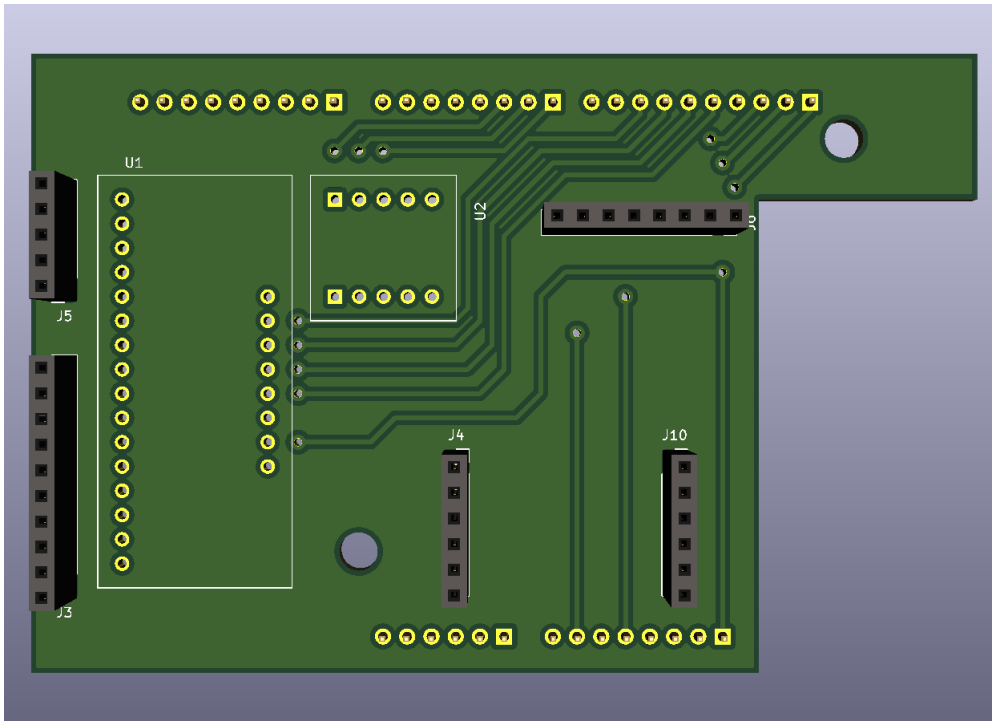


Figure 3.11: Front view of the developed PCB

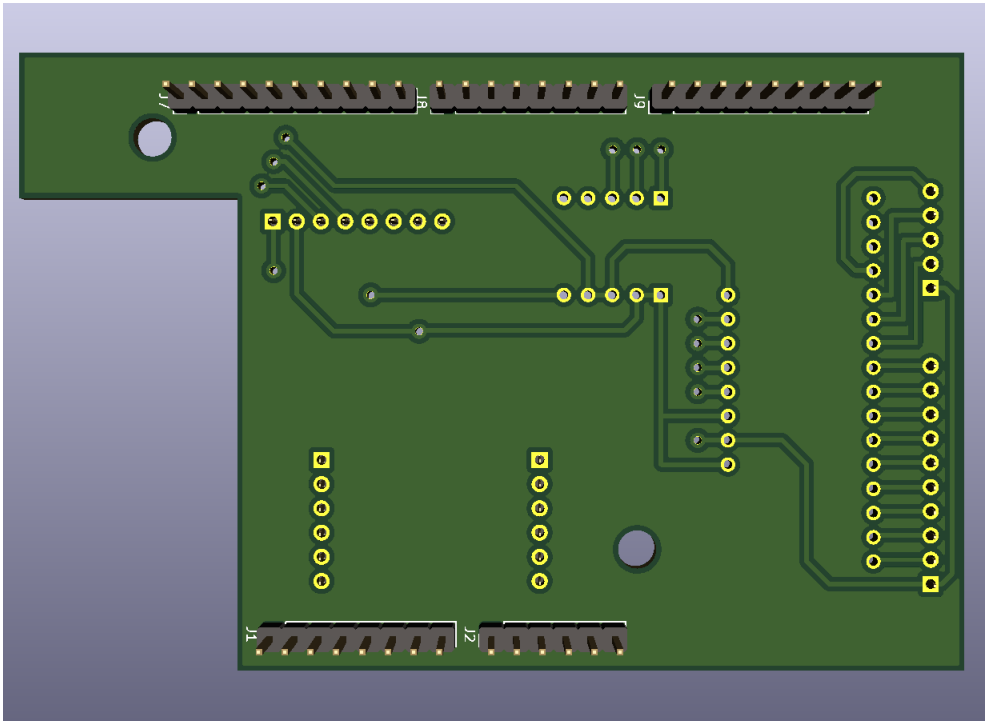


Figure 3.12: Rear view of the developed PCB

Figure 3.13: Final monitoring system - PCB coupled to the microcontroller

3.2 Firmware

The software development focused on the acquisition of data from pressure sensors, at a defined rate of 100 Hz. Thus, in sections 3.2.1 and 3.2.2, is a description of how the entire code works, referring to functions and *Timers* used to make the system work as expected. It is also important to point out that the *software* was developed in the SEGGER development tool in C programming language since its use was recommended by the Nordic board manufacturers.

3.2.1 Main and Timer

In *main()* function (Figure 3.14 and Appendix A.1), the main execution code of the program is carried out, where a set of initializations are made, they are: output terminal, board power management system, initialization of analog pins (function *ADC_init ()*, Figure 3.15 and Appendix A.1) and Timer. Additionally, pins 22 to 25 of the nrf52832 board are configured as output pins, for controlling the multiplexer. In the *ADC_init()* function, the ADC peripheral is initialized, for later configuration and initialization of the channel used to read the ADC value.

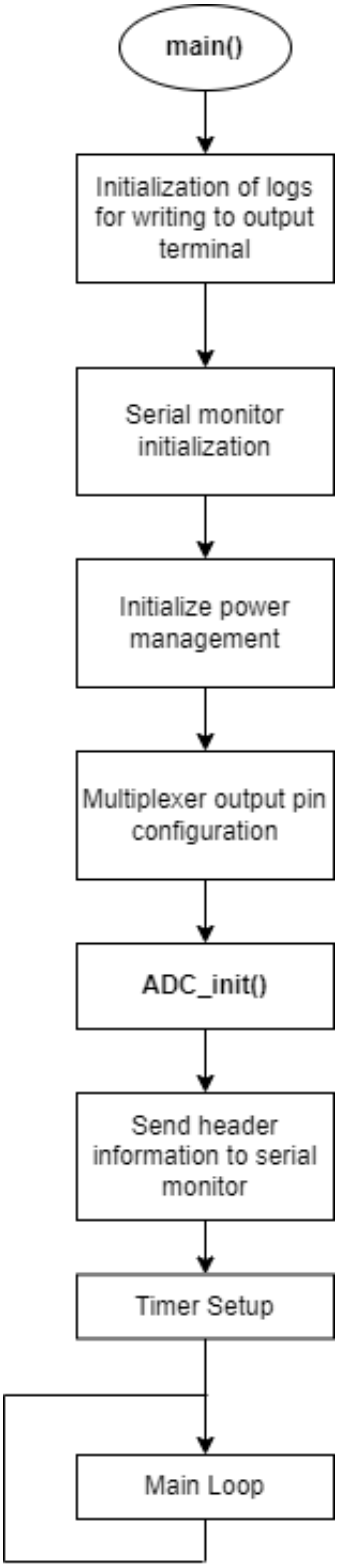


Figure 3.14: main() function flowchart

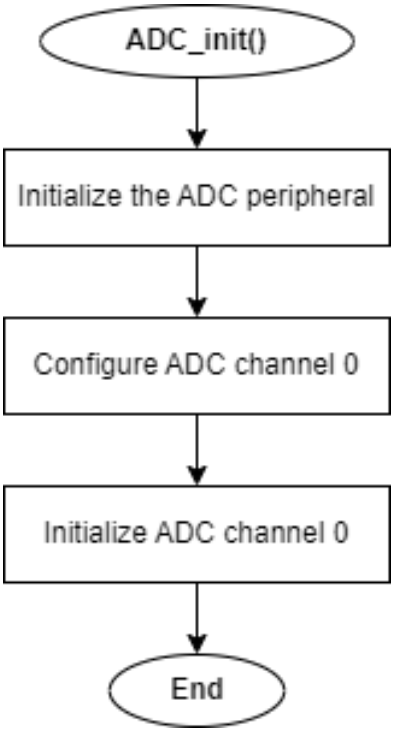


Figure 3.15: ADC_init() function flowchart

Using the Timer started by defining an instance that represents it and defining its configuration parameters: frequency, operating mode, number of bits, and task priority. After configuring the Timer, it is initialized through the *nrfx_timer_init()* function of the *nrfx_timer.h* library, which has three parameters: the created timer instance, timer configuration structure and the event handler, which is a callback function that will be invoked when the timer is running, called *timer1_callback()* (Figure 3.16 and Appendix A.1).

The desired time in ms for the timer is also defined, which is converted to the number of ticks using the *nrfx_timer_ms_to_ticks()* function. Then the timer is defined in extended compare mode using the function *nrfx_timer_extended_compare()* (also from the library *nrfx_timer.h*), that is, when the timer is running, its current number of ticks is constantly compared with the total number of ticks corresponding to the defined elapsed time ms.

When the current number of ticks is equal to the number of ticks of the defined time, an event called *NRF_TIMER_EVENT_COMPARE1* is generated. The function *timer1_callback()* identifies whether an event of this type is generated, which will make the readings of all sensors take place.

Finally, the timer is enabled through the function *nrfx_timer_enable()* (from the *nrfx_timer.h* library).

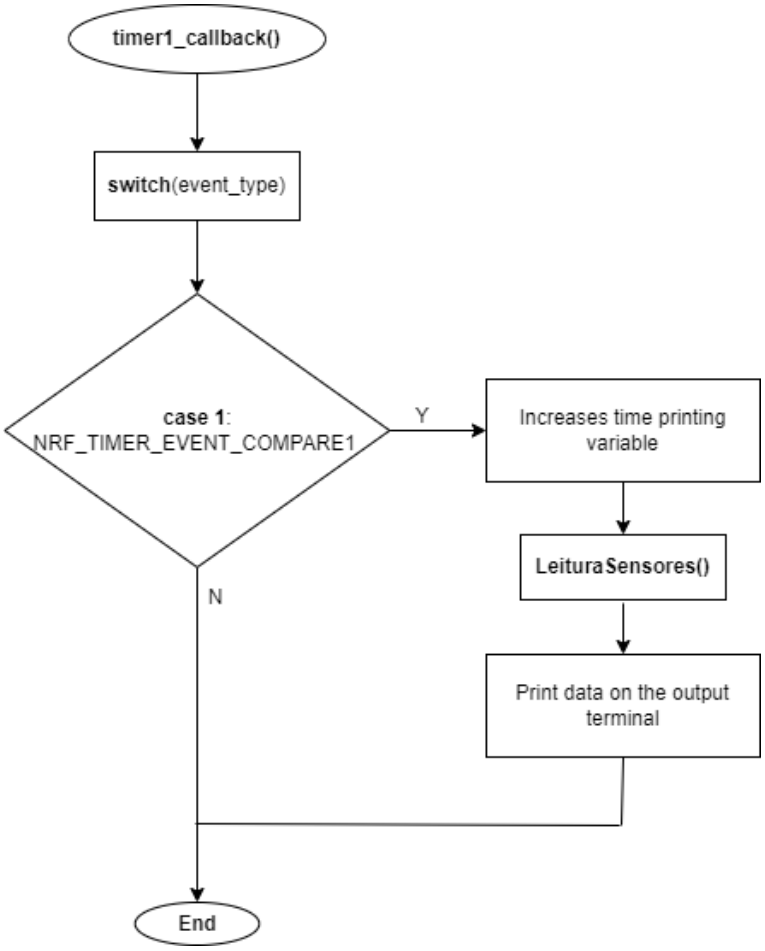


Figure 3.16: timer1_callback() function flowchart

3.2.2 Sensors reading

To read the sensors it was developed the function *LeituraSensores()*, Figure 3.17 and Appendix A.1, which has a *while()* loop that will make the transition of the nine channels of the multiplexer used after each reading. After selecting each channel of the multiplexer, a function called *MediaLeiturasADC_DP()* (Appendix A.1) is called. This function, Figure 3.18, also has a *while()* loop, which will count the number of readings being taken, until it takes 50 readings on the specified analog channel. The 50 readings are summed together, and then averaged to obtain a single value. This average value is then used in the *LeituraSensores()* function to be converted into a pressure value (kPa) through the calibration curves of each sensor.

The values of each sensor are stored in a vector of nine positions to be sent later by the serial port. Figures 3.17 and 3.18 show the flowcharts of the functions *LeituraSensores()* and *MediaLeiturasADC_DP()*, respectively.

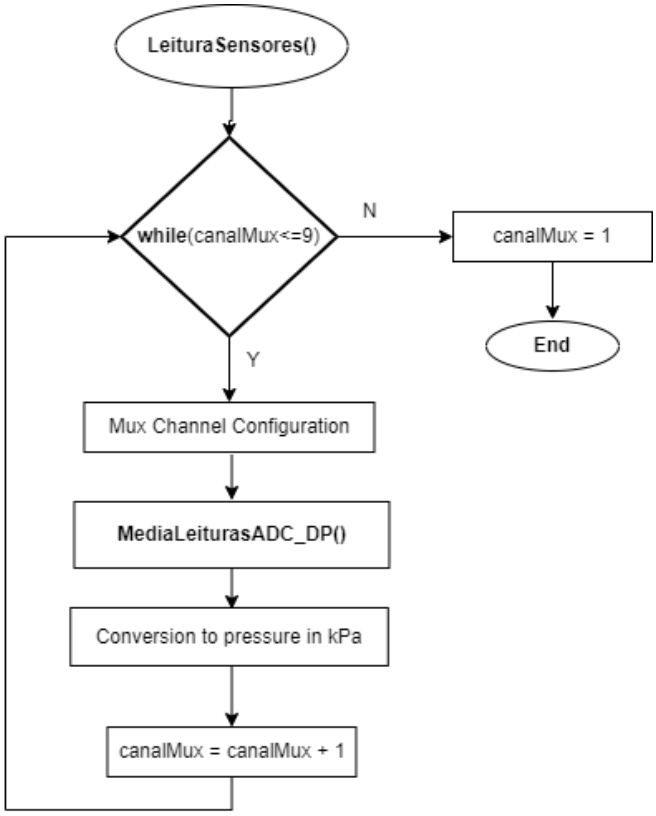


Figure 3.17: LeituraSensores() function flowchart

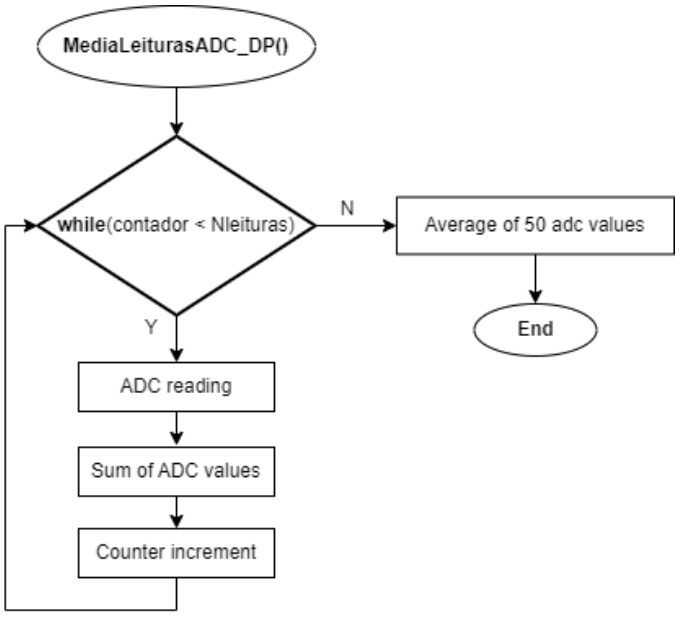


Figure 3.18: MediaLeiturasADC_DP() function flowchart

3.3 Conclusion

In summary, the development of the monitoring system programming focused on the reading of the analog channel of the nrf52832 board, accompanied by the change of the multiplexer channel for reading the different sensors, using a timer to guarantee an acquisition rate of 100 Hz.

The next chapter describes the AI models developed for classifying the type of movement of the person. The type of data used in the AI models is also specified, as well as their treatment.

Chapter 4

AI models

The construction of an AI model is performed from the dataset that will be used to train it. In this context, the following chapter discusses two datasets used and the AI models developed, detailing the processing of the data before using them in the model. Since the monitoring system was developed in the SH4ALL project period, initially it was important to consider a public dataset for the development of the model, while the monitoring system was being built. When the system was ready, different tests were carried out, with different participants to record plantar pressure data and for later construction of the second dataset.

4.1 First Dataset

As mentioned, initially a public dataset was used, which was constructed using plantar pressure data from 15 healthy young participants, who walked on a treadmill at three different speeds [51]. For the tests to be controlled more easily and to obtain a great diversity of data, several acquisition systems were used (Figure 4.1), such as seven IMUs positioned in the lower body, Zebris system FDMT-HQ, OptoGait system, and two RGB cameras.

The Zebris system consists of a pressure distribution treadmill that integrates an array of capacitive force sensors [51] [52] [53] [54]. The Optogait system is equipped with two bars, one transmitting and the other receiving, made up of optical sensors, infrared *Light-Lmitting Diode* (LED), that work at a frequency of 1000 Hz and record parameters such as space and time for any type of tests (walking, running, etc.) [51]

[55] [56]. This system is important for recording participant contact times and foot positions.

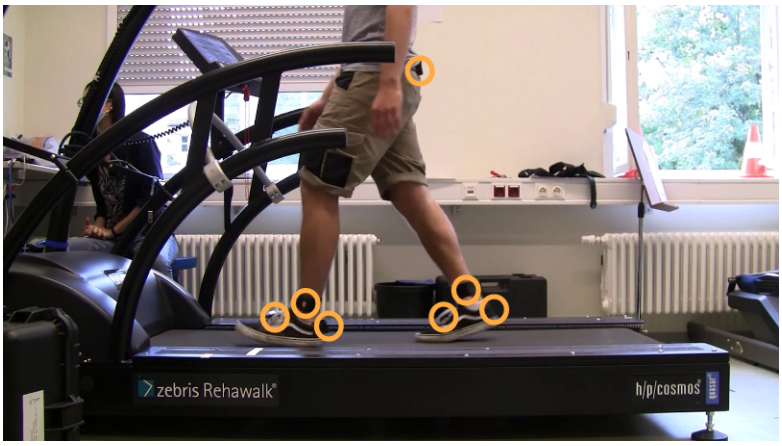


Figure 4.1: All systems setup [52]

An experimental protocol was defined to be followed for carrying out the tests. As indicated above, each participant performed three trials (2 minutes each) at three different speeds (preferred walking speed, 20% fast, 20% slow) [51] [57]. Before performing the final tests, each participant had the opportunity to walk on the treadmill for 6 minutes at the chosen speed so that they could become familiar with it [51] [58]. The preferred speed of each participant was also defined, by slowly increasing the speed of the treadmill, until the participant reported that he was at his normal walking speed [51].

The Zebris system was configured to record at 128 Hz and the OptoGait system was configured to record at 1000 Hz. In the final trials, data were obtained from both feet which were then recorded separately, left and right foot. For each foot, data from each of the IMUs were stored, as well as data from the two systems OptoGait and Zebris. However, since the objective of the AI model is to classify the type of gait of a person through the values of plantar pressure, only the pressure data recorded by the Zebris system were used to train the model.

The data of this system were recorded in a *Comma-Separated Values* (CSV) file that contains the plantar pressure data throughout each test and each pressure reading recorded with the timestamp previously set, represents the sum of all force sensors that are pressed on the treadmill.

4.2 Second Dataset

In chapter 3, the monitoring system developed to collect plantar pressures was described and it was with this system that were acquired the data for the construction of a second dataset. It was also necessary to create an experimental protocol

(Appendix B) to be followed by the participants. Tests were performed with seven healthy participants, who walked on a treadmill (Figure 4.2) at three different speeds and also performed a test at rest (standing still). In all tests, the values of each sensor were recorded as well as the reading time, using a python script (Appendix A.3) to read the Universal Serial Bus (USB) port and record the information in a .txt file (the monitoring system worked at 100 Hz).

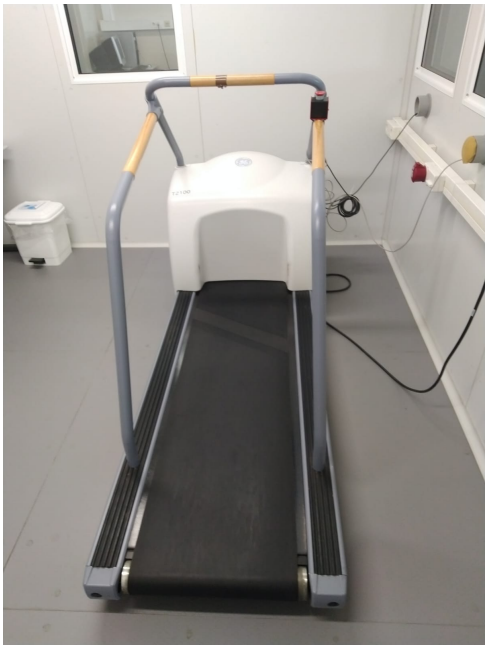


Figure 4.2: Treadmill used in the tests

In the experimental protocol, four tests were defined that the participant would have to do at four different speeds (normal preferred speed, slow preferred speed, fast preferred speed and rest). Before starting the tests, some steps were taken into account, in which the first was the registration of the participant's metrics (Name, Age, Gender, Weight, Height), the second step was to determine and record the reference speeds of the participant, being carried out a trial where the speed was increased until the participant detected that he was walking at his normal, slow and fast speed. There were also some comments that were registered about, for example, the type of foot of the person, way of walking, etc.

Each participant performed one trial at rest for 40 seconds, and another at their normal, slow, and fast speed for 2 minutes (Figure 4.3). Only one sensorized shoe (right foot) was used, so only data from the right foot were collected, however, the two Orthomedical shoes were worn to ensure that there was no difference in the way the participant positioned the foot when walking. Four .txt files were recorded for each participant for further processing.

Figure 4.3: Test with a participant

4.3 Classes

The AI model is built from the type of data that is intended to be used for training the model, these data are called model input data and in this case, there will only be one input class which is the sum of the pressure values in all sensors. Regarding the output classes, since the objective of the neural network would be to classify the type of gait of the user, the output class of the model is the type of gait of the person. As mentioned earlier, plantar pressures were recorded for each individual at three different gait speeds, and of these three speeds, only two were selected: slow walking speed (which is the participant's slow speed) and fast walking speed (which is the participant's fast speed). Thus, as output variables, there are fast and slow walking.

4.4 Network models

There is a wide variety of approaches to build AI models, as mentioned earlier. In this way, three different models were built and later standardization and normalization methods were applied to the model's input data. This methods were implemented to analyze if there was any increase in the percentage of accuracy in the classification of the different types of activities. When building the models, it was necessary to process the input data and define them according to the input structure of the AI models. Therefore the treatment performed is described below and then a description of each of the AI models is presented (it is important to note that the models were all developed in the Jupyter Notebook tool).

4.4.1 Prior data processing of first dataset

As mentioned before, the data from the Zebris pressure mat were recorded in CSV files and therefore for each test of each participant a CSV file was obtained for the right foot and another for the left foot, with two columns in which the first corresponds to time (seconds) and the second to force (Newton). By analyzing each of these CSV files it was possible to denote a certain pattern of the data in each foot. Figure 4.4 is a graph obtained through data from a CSV file of the left foot of one of the participants in slow gait, where the butterfly pattern of the person's gait cycle can be observed. Analyzing this figure, it is possible to observe that in a part of the gait cycle, there are values that are at 0 N, which means that the foot is not pressing the treadmill and therefore corresponds to the swing phase of this cycle.

Something important to mention is the difference between the gait cycle at a person's normal gait speed and the gait cycle at a person's accelerated speed (eg. running). When a person is at a higher speed, the duration of the gait cycle is shorter, the peak force value (in the stance phase) is higher and the graph format is no longer similar to a butterfly having only one peak. Since these are the characteristics that distinguish the two types of activities intended to be classified, the CSV file of each foot was divided into several CSV files, where each one presents the data of only one gait cycle. Thus, if in a test the participant performs 95 gait cycles, a CSV file of one foot with all the gait cycles is obtained, which is then divided into 95 CSVs. A python script (Appendix A2.1) was developed to divide the CSV files into several gait cycles.

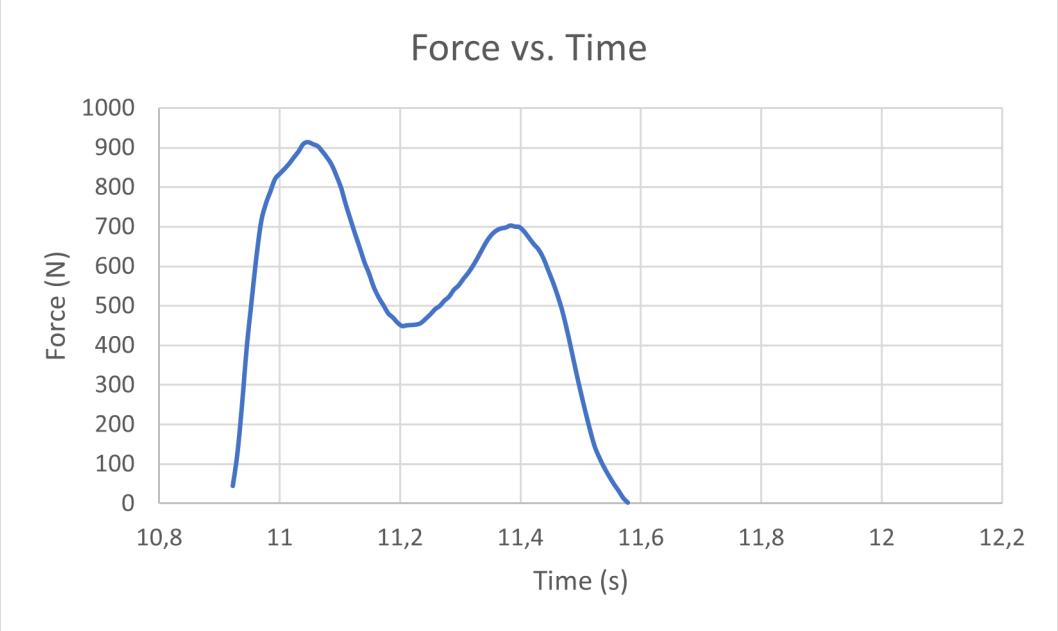


Figure 4.4: Walking cycle taken from a CSV file corresponding to a slow gait test of a random participant

After dividing each test (each CSV) into several cycles (several CSVs), it was necessary to aggregate all this information in a single CSV file (Appendix A2.2). This new file aims to present the input data of the AI model, so that, when developing the model, it was necessary to understand which input data format it would accept.

Following this chapter, the various types of models developed are discussed, however, for now, it is important to mention that all models will use the same input vector format and it is also important to mention that the models used require that the number of samples of input (samples that characterize each cycle) is constant.

In this way, the new CSV file is made up of 228 columns, where the first and second columns correspond to the participant's ID and type of activity and the others present the pressure values throughout each gait cycle: [ID_PARTI.], [ACTIVITY], [PRESSURE1], [PRESSURE2]...[PRESSURE226].

That is, each line of this file is a gait cycle of a participant in a certain type of test. 226 columns were destined for the recorded pressure data since when analyzing the number of pressure values that were recorded in a single cycle, it was noticed that not all cycles had the same number of samples different from zero (since they are different people, at different speeds).

4.4.2 Prior data processing of second dataset

As mentioned before, the data obtained from the monitoring system was recorded in a .txt file constituted of 10 columns, in which the first column shows the time in seconds of each reading performed and the other columns represent the pressure values recorded in each sensor.

In the case of the public dataset, the pressure values recorded in the CSV files are already the sum of the forces measured by the force sensors that are pressed on the treadmill, but in the case of this second dataset, the values recorded were from each sensor and not the sum. Thus, since the sum of pressures of all sensors was analyzed for the first dataset, for this second dataset the sum of the values of the 9 sensors was also used.

A python script was developed (Appendix A5.3) that performs the sum of the values of all sensors for each trial, considering for each reading the sum of all sensors, as in the first dataset. With these values, the evolution of pressure throughout the test was analyzed, where it was possible to detect the presence of noise in the signal obtained, as shown in Figure 4.5 (example of five gait cycles of a random participant).

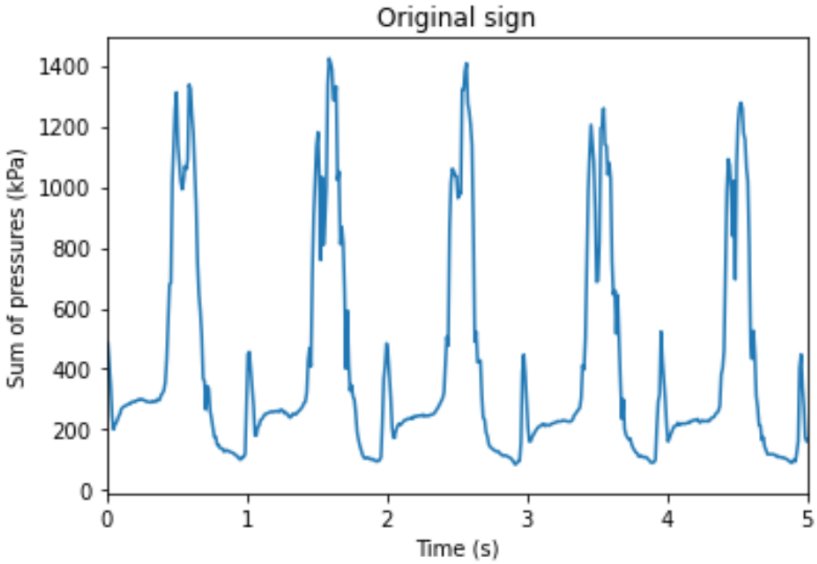


Figure 4.5: Evolution of the sum of pressures throughout the test corresponding to the first five gait cycles

It was then necessary to study some types of filters that could be applied to remove the noise present in the original signal, considering the *Butterworth* (BW) filter and the Fourier transforms. The Butterworth filter was developed by Stephen Butterworth in 1930 [59] and has the characteristic of having been designed to have a frequency response as smooth as possible, that is, the bandpass ripple and the slope of the transfer function present low values, the frequency response being flat in the bandpass approaching zero in the stop band [59][60][61].

Thus, a low-pass Butterworth filter was implemented through a python script (Appendix A5.2) using the SciPy library that provides functions that perform signal processing. It was created a function *butter_lowpass()*, which has as input parameters, the data array to be filtered, the desired cut-off frequency (in this case 10 Hz), the sampling frequency of the data, the filter order and finally the Nyquist frequency. This function implements another function from the SciPy library, *butter()* [62], which has as parameters: filter order, critical frequency from which the gain drops, type of filter intended (in this case low-pass) and finally the indication that a digital filter is intended to be implemented. The function *butter()* returns two arrays that are the polynomial numerator and denominator to be used later in an *Infinite Impulse Responde* (IIR) filter. Finally, through the function *filtfilt()* an IIR filter is applied to the signal twice, once to the front and once to the back [63]. This function has three parameters, the filter numerator, the filter denominator, and the data array to be filtered and its output is the already filtered data array [63].

Fourier transforms (FT) are based on transformations of a given continuous signal in time, between the time and frequency domains [64]. To apply FT calculations on a computer, it is necessary to discretize the signal [64] (decomposition of the

signal into a sum of periodic components). This process can be carried out through the so-called *Discrete Fourier transforms* (DFT), which, to be applied, require a large amount of computation time, especially if the number of samples to be filtered is very large (which happens in this case) [64] [65] [66]. To overcome this problem, *Fast Fourier Transforms* (FFT) [64] [65] [66] were developed, which are the implementation of an algorithm that reduces the number of calculations for all points signal [64] [65], thus accelerating the computation time of the DFT.

To implement the FFTs to the pressure signal obtained, a Python script was also developed (Appendix A.4) with a function, *fft_denoiser()*, which receives as parameters the data array to be filtered and the size of the same array and using the numpy library it is possible to apply a function called *fft()*, which calculates the one-dimensional DFT using the FFT algorithm [67] and returns the Fourier coefficients (complex values). A cut-off frequency defined by the number of elements that there is in the signal is then applied for later application of the inverse fast transform (ifft()), which returns the filtered input array.

After developing the codes for each of the filters, it was necessary to compare the results for later choosing the filter that best applies to the signal. Figure 4.5 shows the original signal and Figures 4.6 and 4.7 show the same signal filtered with the BW filter and FFTs, respectively. By analyzing both signals obtained, it is possible to observe that the BW filter smoothes the original signal better than the applied FFTs, making the start and end points of each cycle more noticeable so that the BW filter was chosen for filtering each signal obtained from each participant.

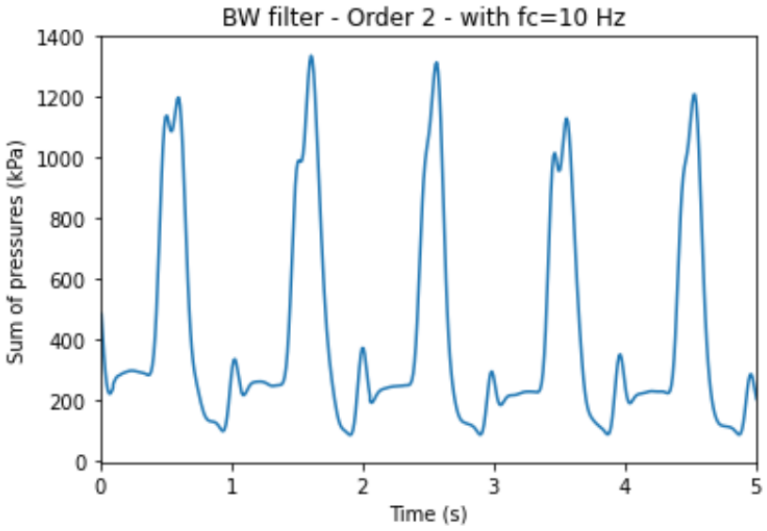


Figure 4.6: Signal filtered with BW filter

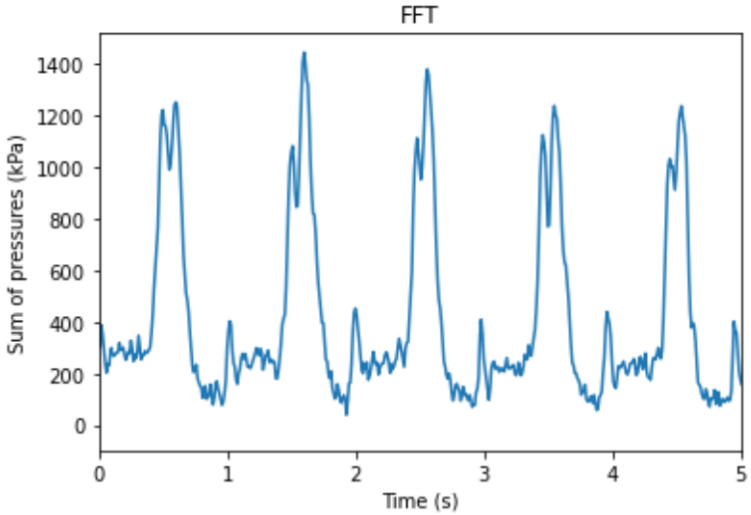


Figure 4.7: Signal filtered with FFts

After filtering the signal obtained, it was necessary to develop a method (Appendix A.5) that allowed the data to be divided into the various gait cycles, that each participant performed throughout their tests. In the previous dataset, a CSV file was obtained for training and testing the model, in that each line corresponded to a gait cycle of a participant at a given speed. One of the characteristics of human gait is that despite detecting a pattern in the pressure signal obtained during gait, the pressures measured in each sensor are not always the same, which leads to each gait cycle presenting different values of pressure relative to others.

In this way, it was necessary to detect the start and end points of each gait cycle, for this, it was used the function *find_peaks()* (also belongs to the Scipy library, Appendix A.5.5) to detect the minimum and maximum points of the signal, Figure 4.8. It was also necessary to analyze each signal of each trial of each participant to understand if the first recorded cycle would be complete, if not, a cut of the first values was performed and it were only considered the values that correspond to the first complete cycle and others (Appendix 5.6). It is also known that the human plantar pressure is normally the highest when the foot strikes the ground until the moment it leaves the ground (foot in the air) so that the maximums detected by the *find_peaks()* function correspond to the moment when the foot is on the ground and the minimum corresponds to the moment when the foot is in the air. After obtaining the maximum and minimum values of the filtered signal (Appendix 5.7), it was also verified that several minimum and maximum values were obtained for each cycle, which led to the development of another script (Appendix A5.9) that allowed the identification of the specific minimum value that identifies the end of the cycle and start of the next.

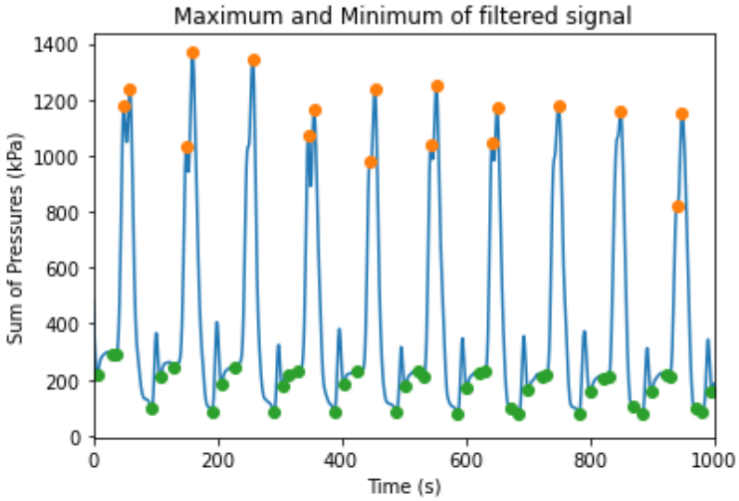


Figure 4.8: Maximums and minimums of the filtered signal

Figure 4.9 shows an example of the beginning and end of a cycle, where the orange dots represent the maximums and the green dots the minimums. It is observed that before the minimum corresponding to the end of the cycle, two more minimums are registered and this number of minimums is not always constant, that is, in a certain cycle one can only register three minimums, but for the following cycle, it can be registered four minimums for example.

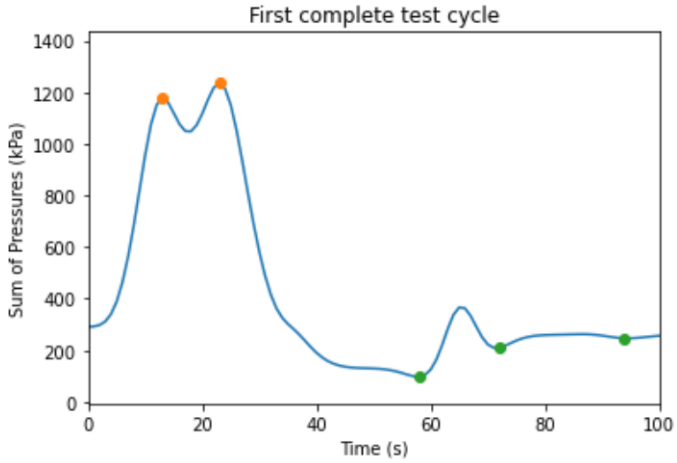


Figure 4.9: First complete test cycle

So, the next step was to count the minimums present in each cycle (Attachment A5.8), that is, the number of minimums that exist between each maximum was counted. With the number of minimums, a vector was then created that includes all the minimum points at which each cycle is to be cut, that is, in the case of the

cycle shown in Figure 4.9, it is known that the gait cycle ends at the third minimum recorded before the next maximum and therefore it is intended to cut the data from that minimum since the following data already correspond to the next cycle that is supposed to be recorded in another CSV file. Thus, it was possible to create a vector with the specific point at which each cycle of the entire signal is to be cut (Appendix 5.8).

After obtaining the minimum cut-off points for each cycle, another python script was developed (Appendix A5.10) that allowed recording the data of each cycle in a new CSV file. After dividing the filtered pressure signal into several CSV files for each cycle, it was necessary to formulate the final CSV (Appendix A5.11) that served as a dataset for training and testing the developed AI models. As in the previous dataset, a CSV was created where all the cycles obtained from all participants were added and as before, it was necessary to define a fixed number of columns for the input vector format. This new CSV has 202 columns where the first and second columns correspond to the participant ID and type of activity and the remaining ones correspond to the pressure values for each gait cycle: [ID_PARTI.], [ACTIVITY], [PRESSURE1], [PRESSURE2]...[PRESSURE200]. Each line of this file then corresponds to a gait cycle of a participant in a certain type of test. 200 columns were destined for the pressure data because each gait cycle presents a different amount of measured pressures from the others, so to standardize the dataset the 200 columns are filled with the cycle pressure data and if the cycle does not have 200 values of measured pressure, the rest of the data is taken as 0.

4.4.3 *Long Short-Term Memory* (LSTM) Model

The first model developed was based on an LSTM model. This type of networks are a particular type of RNNs, which are networks that have feedback connections between nodes and layers that allow the processing of random-length input sequences and, as mentioned in the state-of-the-art chapter, RNNs have the particularity of learning using past information. However, if a large amount of passed information is required (long-term dependencies), RNNs have difficulties in their training [68] [69], which generated the problem called exploding/vanishing gradient problem [70] [71]. These difficulties were overcome when the emergence of LSTM [72] networks, which are networks that have internal memory and multiplicative ports [68] [69].

Thus, for the practical development of the LSTM model, the Jupyter Notebook tool was used, which is based on a web application with a wide variety of features, from document creation to code development, supporting several programming languages. In this case, this tool was used, integrated into Visual Studio Code and Python (version 3.7.13) was used as the programming language, since in addition to being a high-level language it has a wide variety of libraries of artificial intelligence and beyond.

The first step for the development of the model was the loading of the data present in the previously treated dataset. A Python function called *load_dataset3()* (Appendix A6.1) was developed, where the *numpy* library [73] was used to store the pressure data in a numpy array, and the velocity data in another numpy array. The Keras *Application Programming Interface* (API) is also used to encode the output vector, which consists of integers, in classes so that the data are suitable for fitting a neural network multiclass classification model. To finish loading the dataset, the arrays were divided into 80% for training and the remaining 20% for test, obtaining four new arrays: pressure array for training, velocity array for training, pressure array for testing, and speed array for testing.

To define, adjust and evaluate the LSTM model, a function called *evaluate_model()* was developed (Appendix A6.1 and code below), which has as parameters the training and test arrays obtained by the *load_dataset3()* function. To create the LSTM model it is first necessary to define it through the Keras deep learning library called TensorFlow [74]. Firstly, the dimensions of the input and output of the model are defined: the number of time steps (226), which is the quantity of pressure data in a window, number of variables or features that exist in each window (in this case it is only 1, the pressure) and number of outputs (in this case 2 outputs which is the type of activity - slow or fast). Next, a Keras Sequential model is created by importing the Sequential template from the Keras library [75], which is a model made up of a set of simple [75] layers. In this case, a hidden LSTM layer is defined, followed by a dropout layer (which was also imported from the Keras library [76]) intended to reduce *overfitting*. Overfitting occurs when training the model with the corresponding training data, it presents an excellent performance, however, when using the data intended for testing the result is not good [76]. To interpret the results obtained in the dropout layer, a dense layer is used. A dense layer is a fully connected layer that can be defined by the number of neurons that are intended to use and an activation function [77]. The activation function will decide whether each neuron will be used or not, that is, through a mathematical function, it is concluded whether or not a particular neuron is important in the prediction of the model [77]. Finally, another dense layer is used to define the dimension of the output array from the number of outputs that the model intends to have.

Still in the *evaluate_model()* function, after creating the LSTM model, it was then configured for training via the *compile()* method belonging to the Keras library models API [78]. Then the model is trained for a fixed number of *epochs* (repetitions), which in this case are 25, through another method of models API called *fit()* [78]. After training, it is time to evaluate the classification accuracy of the model through the previously obtained test dataset (test arrays) via the *evaluate()* method of models API [78].

```
1
2  import csv
3  import numpy as np
4  from numpy import mean
5  from numpy import std
6  import tensorflow as tf
7  from tensorflow.keras.models import Sequential
8  from tensorflow.keras.layers import Dense
9  from tensorflow.keras.layers import Dropout
10 from tensorflow.keras.layers import LSTM
11
12 def load_dataset3():
13     P = np.genfromtxt(C:/Users/carol/OneDrive/Ambiente
14         de Trabalho/Ensaios_Finais/Final_CSV.csv,
15         delimiter=',',
16         missing_values=None, filling_values=0.0)
17
18     d = P.shape[0]
19
20     P = np.append(P, np.zeros((d, 256 - P.shape[1] + 2)),
21                   axis=1)
22     np.random.shuffle(P)
23
24     V = P[:,1:2]
25     P = P[:,2:]
26
27     P = P.reshape((P.shape[0], P.shape[1], 1))
28
29     V = V.reshape(V.shape[0])
30     V = tf.keras.utils.to_categorical(V)
31
32     divide = round(d*0.80)
33
34     trainX = P[:divide]
35     testX = P[divide:]
36     trainY = V[:divide]
37     testY = V[divide:]
38
39     return trainX, testX, trainY, testY
40
41 def evaluate_model (trainX, trainY, testX, testY):
42     verbose, epochs, batch_size=0,25,64
43     n_timesteps, n_features, n_outputs = trainX.shape[1],
44                                          trainX.shape[2],
45                                          trainY.shape[1]
46     model = Sequential()
47     model.add(LSTM(100,input_shape=(n_timesteps,
48                     n_features)))
49     model.add(Dropout(0.55))
```

```
50      model.add(Dense(100, activation='relu'))
51      model.add(Dense(n_outputs, activation = 'softmax'))
52      model.compile(loss='categorical_crossentropy',
53                    optimizer='adam', metrics=['accuracy'])
54
55      model.fit(trainX, trainY, epochs=epochs,
56                batch_size=batch_size, verbose=verbose)
57      _, accuracy = model.evaluate(testX, testY,
58                                   batch_size=batch_size,
59                                   verbose = 1)
60
61      return  accuracy
```

4.4.4 CNN LSTM Model

The second model is based on the LSTM network created, however, to improve the performance of the previous model, layers based on CNN networks were used, which combined with LSTM layers allow the extraction of more resources, which increases the model's accuracy.

The dataset loading and handling of training and test arrays for this model is the same as described in the previous model. In this model, the function *evaluate_model()* (Appendix A7 and code below) has the same parameters as before, that are the training and test arrays, however, this model will process blocks taken from the input arrays (training and testing pressure array), to be later interpreted by the LSTM layer. As mentioned in the first model, the input array has 226 timesteps in each window, however, in order for the blocks to be more easily divided, it was considered to have a window of 256 timesteps, instead of 226, and thus to divide this window into four subsequences of 64 timesteps.

A Keras Sequential model was created again, however in this case two encapsulated convolution layers were first defined so that the model interprets each of the four blocks of the input window [79]. This is followed by an encapsulated dropout layer and a max pooling layer that reduces the input resolution [80], with the features obtained later being flattened (transformation of multidimensional input to one-dimensional input) by a Flatten layer [81]. The Flatten layer is used for later transaction to the LSTM model (mentioned in the previous chapter). All these new layers used are also part of the Keras library.

```
1
2  import csv
3  import numpy as np
4  from numpy import mean
5  from numpy import std
6
```

```
7  import tensorflow as tf
8  from tensorflow.keras.models import Sequential
9  from tensorflow.keras.layers import Dense
10 from tensorflow.keras.layers import Flatten
11 from tensorflow.keras.layers import Dropout
12 from tensorflow.keras.layers import LSTM
13 from tensorflow.keras.layers import TimeDistributed
14 from tensorflow.keras.layers import Conv1D
15 from tensorflow.keras.layers import MaxPooling1D
16
17
18 def evaluate_model (trainX, trainY, testX, testY):
19     verbose, epochs, batch_size=0,25,64
20     n_steps = 4
21     n_length = 64
22     n_timesteps, n_features, n_outputs = trainX.shape[1],
23                                          trainX.shape[2],
24                                          trainY.shape[1]
25
26     trainX = trainX.reshape((trainX.shape[0], n_steps,
27                              n_length, n_features))
28     testX = testX.reshape((testX.shape[0], n_steps,
29                            n_length, n_features))
30
31     model = Sequential()
32     model.add(TimeDistributed(Conv1D(filters=64,
33                               kernel_size=3,
34                               activation='relu'),
35               input_shape=(None,n_length,n_features)))
36
37     model.add(TimeDistributed(Conv1D(filters=64,
38                               kernel_size=3,
39                               activation='relu')))
40
41     model.add(TimeDistributed(Dropout(0.5)))
42     model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
43     model.add(TimeDistributed(Flatten()))
44     model.add(LSTM(100))
45     model.add(Dropout(0.5))
46     model.add(Dense(100, activation='relu'))
47     model.add(Dense(n_outputs, activation = 'softmax'))
48     model.compile(loss='categorical_crossentropy',
49                   optimizer='adam', metrics=['accuracy'])
50
51     model.fit(trainX, trainY, epochs=epochs,
52               batch_size=batch_size, verbose=verbose)
53
54     _, accuracy = model.evaluate(testX, testY,
55                                  batch_size = batch_size,
```

```
56                                  verbose = 1)
57
58      return accuracy
```

4.4.5 Convolutional LSTM Model

In this third model, the junction of CNN networks with LSTM networks was explored again, however this time the objective was to use the CNN convolutions as part of the LSTM, this combination is called Convolutional LSTM. The big difference between this model and the previous model is that instead of using a CNN layer and an LSTM layer, it is used only a layer that contains the convolution operation inside the LSTM cell used.

Once again, loading the dataset and handling the model input arrays is the same, however the function *evaluate_model()* (code snippet below) has a different model structure and the model input arrays also have different dimensions. A Keras Sequential model was created again in which the first layer of this model is a Convolutional LSTM layer that supports 2D data [82] and expects input data with the following shape: samples, time, rows, columns, and channels. For this problem, the parameter *samples* is the number of windows present in the dataset, the parameter *time* is the number of blocks into which the dataset is divided (in this case there are four), the parameter *rowws* represents the *One-Dimensional* (1D) format of each line that constitutes each subsequence, the parameter *columns* which is the number of timesteps present in each input block and finally, the parameter *channels* is the number of input variables, which in this case is just one, the pressure. This is followed by a dropout layer, a flattened layer, another dropout layer, and two dense layers.

```
1
2  import csv
3  import numpy as np
4  from numpy import mean
5  from numpy import std
6  import tensorflow as tf
7  from tensorflow.keras.models import Sequential
8  from tensorflow.keras.layers import Dense
9  from tensorflow.keras.layers import Flatten
10 from tensorflow.keras.layers import Dropout
11 from tensorflow.keras.layers import LSTM
12 from tensorflow.keras.layers import ConvLSTM2D
13
14
15 def evaluate_model(trainX, trainY, testX, testY):
16   verbose, epochs, batch_size = 0, 25, 64
```

```
17
18  n_timesteps, n_features, n_outputs = trainX.shape[1],
19                                  trainX.shape[2],
20                                  trainY.shape[1]
21
22  n_steps, n_length = 4, 64
23  trainX = trainX.reshape((trainX.shape[0], n_steps,
24                           1, n_length, n_features))
25  testX = testX.reshape((testX.shape[0], n_steps,
26                         1, n_length, n_features))
27  modelo = Sequential()
28  modelo.add(ConvLSTM2D(filters=64,
29                        kernel_size=(1,3),
30                        activation='relu',
31                        input_shape=(n_steps, 1,
32                                     n_length,
33                                     n_features)))
34
35  modelo.add(Dropout(0.55))
36  modelo.add(Flatten())
37  modelo.add(Dropout(0.5))
38  modelo.add(Dense(100, activation='relu'))
39  modelo.add(Dense(n_outputs, activation='softmax'))
40  modelo.compile(loss='categorical_crossentropy',
41                 optimizer='adam', metrics=['accuracy'])
42
43  modelo.fit(trainX, trainY, epochs=epochs,
44             batch_size=batch_size, verbose=verbose)
45
46  _, accuracy = modelo.evaluate(testX, testY,
47                                  batch_size=batch_size,
48                                  verbose=1)
49
50  return accuracy
```

4.5 Standardization and Normalization

After the development of these three models, two methods/techniques of resource sizing were applied, Normalization and Standardization, for further analysis and comparison of the model's accuracy with and without the application of these techniques. These two methods, process the data that will serve as input to the additional AI model and apply cleaning techniques. Normalization is about making the data homogeneous, where it creates a connection between the input data by calculating a new point. This new point is obtained by applying two formulas (Equations

4.1 and 4.2) that scale each value to be within a certain range, for example between zero and one [83].

$$X\_std = (X - X.min(axis = 0))/(X.max(axis = 0) - X.min(axis = 0)) \quad (4.1)$$

$$X\_scaled = X\_std * (max - min) + min \quad (4.2)$$

, where *min* and *max* are minimum and maximum values.

Standardization is a technique normally used when the data obtained do not behave as expected, that is, they do not approach a certain model and therefore can be defined as the process of putting different data on the same scale. The data are resized to obtain a mean value of 0 and a unit standard deviation, from Equation 4.3 [84].

$$z = (x - u)/s \quad (4.3)$$

, where u represents the mean of the training data and s is the standard deviation of the training data.

The application of these methods was carried out in the processing of data from the CSV which has all the cycles that will serve as training and test data for the AI model. As mentioned in chapter 4.4.1, the data were aggregated in a single CSV file that has all the gait cycles obtained from the participants at different speeds, and in the chapters that describe the models developed, it is also explained that these pressure data are converted to a numpy array. After obtaining this array, it is used the *MinMaxScaler()* function to obtain the normalized data [85] or the *StandardScaler()* function to obtain the standardized data [86]. These two functions belong to an ML module for Python called *sklearn.preprocessing* [85] [86]. The two code snippets below show how to apply these functions, where the P array is the pressure data array taken from the CSV file. The *fit_transform* function (*sklearn.preprocessing* method) performs the fitting and transformation of the [83] data.

```
1
2 from sklearn.preprocessing import  MinMaxScaler
3 import pandas as pd
4
5 P = P.reshape((P.shape[0], P.shape[1]))
6
7 scaler = MinMaxScaler()
8 P = scaler.fit_transform(P)
9
```

```
scaled_P = pd.DataFrame(data=P)
new_P = scaled_P.to_numpy()
```

```

from sklearn.preprocessing import StandardScaler
import pandas as pd

P = P.reshape((P.shape[0], P.shape[1]))

scaler = StandardScaler()
P = scaler.fit_transform(P)
scaled_P = pd.DataFrame(data=P)
new_P = scaled_P.to_numpy()
```

4.6 Conclusion

After defining the various AI models to be applied to classify the type of human gait, the implementation of these models was made with the two datasets created. In the next chapter, a record of the percentages of accuracy that each model obtained is made, where the set of model and dataset that obtained the best accuracy results is concluded.

Chapter 5

Implementation and results

In the previous chapter, three neural networks were developed with different architectures and types of layers mentioned and, in the end, it is discussed two methods that are used as cleaning techniques for the data coming from the datasets. These methods were applied so that it was possible to obtain greater accuracy in the classification of the model.

Thus, after developing the three models and defining the two datasets, 18 implementations were carried out, nine with the public dataset and another nine with the monitoring system dataset. These nine implementations for each dataset have been organized as follows:

- Model I (LSTM) without standardization and without data normalization;
- Model II (CNN LSTM) without standardization and without data normalization;
- Model III (ConvLSTM) without standardization and without data normalization;
- Model I (LSTM) with data standardization;
- Model II (CNN LSTM) with data standardization;
- Model III (ConvLSTM) with data standardization;
- Model I (LSTM) with data normalization;

- Model II (CNN LSTM) with data normalization;
- Model III (ConvLSTM) with data normalization;

For each implementation with each dataset, the classification accuracy results of each model were then recorded. These values were obtained by averaging the accuracy values recorded in five model evaluation repetitions.

5.1 Public Dataset

In Table 5.1, the results obtained for each model without data normalization/standardization are registered and in Tables 5.2 and 5.3 are recorded the results obtained with standardization and normalization, respectively.

Table 5.1: Accuracy percentage registration of each model for the public dataset without data standardization or normalization

	Accuracy(%)
Model I (LSTM)	54.81
Model II (CNN LSTM)	89.17
Model III (ConvLSTM)	98.40

Table 5.2: Accuracy percentage registration of each model for the public dataset with data standardization

	Accuracy(%)
Model I (LSTM)	53.27
Model II (CNN LSTM)	99.17
Model III (ConvLSTM)	99.29

Table 5.3: Accuracy percentage registration of each model for the public dataset with data normalization

	Accuracy(%)
Model I (LSTM)	52.72
Model II (CNN LSTM)	95.14
Model III (ConvLSTM)	97.59

Through the results presented, it is possible to conclude that in the three test situations, Model III (ConvLSTM) is the one with the highest percentage of precision, and among the three test situations, the one with the highest precision in Model III is the one with standardization of the model's input data.

5.2 Monitoring system dataset

In Table 5.4, the results obtained for each model without data normalization/standardization are registered and in Tables 5.5 and 5.6 are recorded the results obtained with standardization and normalization, respectively. These values were also obtained by averaging the accuracy values recorded in five model evaluation repetitions.

Table 5.4: Accuracy percentage registration of each model for the monitoring system dataset without data standardization or normalization

	Accuracy(%)
Model I (LSTM)	58.30
Model II (CNN LSTM)	94.76
Model III (ConvLSTM)	98.38

Table 5.5: Accuracy percentage registration of each model for the monitoring system dataset with data standardization

	Accuracy(%)
Model I (LSTM)	56.83
Model II (CNN LSTM)	98.89
Model III (ConvLSTM)	99.63

Table 5.6: Accuracy percentage registration of each model for the monitoring system dataset with data normalization

	Accuracy(%)
Model I (LSTM)	56.09
Model II (CNN LSTM)	97.20
Model III (ConvLSTM)	98.60

With the dataset obtained by the monitoring system, it is possible to conclude that in the three test situations, Model III, is also the model that presents the highest percentage of precision, and among the three situations, the one that reveals a greater precision of Model III it is also the one with standardization of the input data.

5.3 Comparison

It is therefore possible to conclude that Model III is the model with the highest percentage of accuracy in all test situations, both for the public dataset and the system dataset. Furthermore, Model III presented a better percentage of accuracy with both datasets for the test situation that uses the standardization of the input data. Additionally, when comparing the absolute precision value of Model III in this same situation, it is possible to conclude that with the dataset formulated through the monitoring system, a greater accuracy (99.6%) is obtained than with the public dataset (99.3%).

One of the important points to note is that the public dataset was built from pressure data recorded for both feet, while the project dataset only included pressure data from one foot, since only one sensorized shoe was available. Another point to be mentioned is the number of sensors present in each of the systems, since for the public dataset a pressure measurement treadmill was used that included a large number of force sensors, which makes the system has greater precision in the data obtained and in the case of the monitoring system developed (SH4ALL project) only nine pressure reading sensors were used. It is therefore interesting to conclude that despite the large difference in the number of sensors between the two systems, the percentage of classification accuracy of the models is similar for both systems. This may be due to the choice of sensor locations in the insole of the developed monitoring system, since these locations are considered the most critical points of pressure exercised on the plantar region.

5.4 Conclusion

After analyzing and comparing the results for each of the models with the two different datasets, a conclusion of the project is made in the next chapter, where the results obtained are presented again along with some conclusions and the limitations encountered in the development of the project are registered. It is also projected future developments of the project and an general appreciation of the work is given.

Chapter 6

Conclusions

As mentioned before, the SH4ALL project has two components that are still being developed within project planning time, they are the pressure monitoring system and the android application, which in turn has the function of data processing and user interface of the product. Regarding the work described in this dissertation, in the initial phase, it was necessary to contemplate the definition and implementation of the monitoring system and, in parallel, an investigation was carried out on possible public data that could be used to test the AI models, to carry out a later comparison between the data obtained by the system of the SH4ALL project and the commercial systems existing in the market. During the development and testing of the monitoring system, the AI models were elaborated and the form of the input data of the models (processing of the public dataset) was also studied. After the finishing of the monitoring system, an experimental protocol was defined that was followed to collect data on people with the monitoring system. Finally, the data collected from the different participants were processed and a new dataset was formulated.

6.1 Results achieved

The main objective of the work, which was to identify the locomotion status of the shoe user, was achieved, since through the data collected by the monitoring system, it was obtained a percentage of accuracy of the classification model of 99.6%. It is important to mention that the monitoring system used to collect the data had

a different constitution than the system used for the construction of the public dataset, having fewer sensors. It is also important to refer that the project dataset only presents data from one foot of each participant, while the public dataset consists of the data obtained from both feet of each participant. Additionally is also relevant to mention that only seven participants were used in the tests carried out with the monitoring system developed, while data from 15 participants were used for the public dataset. That is, despite the use of a dataset with less data from the monitoring system, it was obtained a better accuracy of the classification model with this same dataset.

6.2 Limitations

During the development of the project, some limitations were identified, such as the placement of the sensorized upper in the Ortomedical shoe, which, as it is a manual process, makes the shoe not adhere to the foot normally, not allowing the use of upper data; the collection of data carried out via USB cable and the use of the monitoring system attached to the participant's leg, that could have caused some disturbances, when collecting data from the participants and the acquisition of the treadmill, which ended up conditioning the planning of the project, since it was only possible to use it in September.

6.3 Future work

There are, however, some improvements and additions that could be included in the project, such as the inclusion of more input variables like:

- Instead of using the sum of pressures as an input variable, use the value of each sensor as an input variable, that is, as there are nine sensors, there would be nine input variables;
- Adding six more variables from the three-axis IMU that is also integrated into the monitoring system (Accelerometer and Gyroscope - AcX, AcY, AcZ, GiX, GiY and GiZ);
- Another feature that could be included is the pressure data recorded by the sensorized upper-shoe, being necessary first to carry out an analysis of the upper pressure data for the different activities to see if there are different pressure variations.

Regarding the dataset to be used for training and testing the model, data from both feet could be recorded to double the amount of data. It could also be carried

out tests of more types of activities in addition to those already analyzed, such as for example: climbing stairs, descending stairs, running, sitting, etc.

An improvement that could be applied to the monitoring system would be to send data via Bluetooth, to facilitate the process of data collection in a laboratory environment, both for the user of the shoes and for the ones who are controlling the reception of the data, since that in the tests carried out the data were collected via USB, forcing to have some cares. Still in the monitoring system, another improvement to consider would be to fit the monitoring system inside the shoe so that the user can move freely, since in the tests carried out the system was positioned around the user's leg.

6.4 Appreciation

That said, this work allowed consolidating knowledge of hardware and software, obtained during the master's degree and degree in Electrical and Computer Engineering, such as the interconnection of electrical components and their programming through the C programming language. In addition it was possible to explore a little of the area of AI, both the theory that accompanies it and its applications. Throughout the project it was also possible to acquire more knowledge about the Python programming language.

References

[1] X. He, W. Shi, W. Li, H. Luo, and R. Zhao, "Reliability enhancement of power electronics systems by big data science," *Proc. Chin. Soc. Elect. Electron. Eng.*, vol. 37, pp. 209–221, 2017. [Cited on pages i and iii]

[2] K. L. TSUI, Y. ZHAO, and D. WANG, "Big data opportunities: System health monitoring and management," *IEEE Access*, vol. 7, pp. 68853–68867, 2019. [Cited on pages i and iii]

[3] S. Zhao, F. Blaabjerg, and H. Wang, "Big data opportunities: System health monitoring and management," *IEEE TRANSACTIONS ON POWER ELECTRONICS*, vol. 36, pp. 4633–4658, 2021. [Cited on pages i and iii]

[4] R. Manne and S. C. Kantheti, "Application of artificial intelligence in healthcare: Chances and challenges," *Current Journal of Applied Science and Technology*, vol. 40, pp. 78–89, 2021. [Cited on pages i and iii]

[5] J. Bughin, J. Seong, J. Manyika, M. Chui, and R. Joshi, "Notes from the ai frontier modeling the impact of ai on the world economy," 2018. [Cited on page 1]

[6] T. B. C. Society, "Living with ai: A personal perspective," *ITNOW*, pp. 32–33, 2019. [Cited on page 5]

[7] A. Bundy, "Preparing for the future of artificial intelligence," no. 285-287, 2017. [Cited on page 5]

[8] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM JOURNAL*, pp. 211–229, 1959. [Cited on page 5]

[9] I. E. Naqa and M. J. Murphy, *Machine Learning in Radiation Oncology: Theory and Applications*. Springer, 2015. [Cited on pages 5 and 6]

[10] F. Lussier, V. Thibault, B. Charron, G. Q. Wallace, and J. Masson, "Deep learning and artificial intelligence methods for raman and surface-enhanced raman scattering," *Trends in Analytical Chemistry*, vol. 124, 2020. [Cited on page 6]

[11] F. Brea, J. M. Gimenez, and V. D. Fachinotti, "Prediction of wind pressure coefficients on building surfaces using artificial neural networks," *Energy and Buildings*, vol. 158, pp. 14429–1441, 2018. [Cited on page 6]

[12] Q. Zhanga, L. T. Yang, Z. Chenc, and P. Lic, "A survey on deep learning for big data," *Information Fusion*, vol. 42, pp. 146–157, 2018. [Cited on page 7]

[13] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, p. 917–963, 2019. [Cited on pages 7, 8, 9, and 10]

[14] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," 2016. [Cited on page 7]

[15] D. Smirnov and E. M. Nguifo, "Time series classification with recurrent neural networks," 2016. [Cited on page 8]

[16] Y. Zhang, R. Jin, and Z. Zhou, "Understanding bag-of-words model: a statistical framework," *International Journal of Machine Learning and Cybernetics*, vol. 1, pp. 43–52, 2010. [Cited on page 8]

[17] A. Bagnall, J. Lines, J. Hills, and A. Bostrom, "Time-series classification with cote: The collective of transformation-based ensembles," *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, vol. 27, pp. 2522–2535, 2015. [Cited on page 8]

[18] J. LINES, S. TAYLOR, and A. BAGNALL, "Time series classification with hive-cote: The hierarchical vote 2 collective of transformation-basedensembles," *ACM Transactions on Knowledge Discovery from Data*, vol. 12, p. 917–963, 2018. [Cited on page 8]

[19] J. LINES, S. TAYLOR, and A. BAGNALL, "Hardware description of multilayer perceptrons with different abstraction levels," *Microprocessors and Microsystems*, vol. 30, pp. 435–444, 2006. [Cited on page 9]

[20] B. Widrow and M. A. Lehr, "Artificial neural networks of the perceptron, madaline, and backpropagation family," *Proc. IEEE*, vol. 78, p. 1415–1442, 1990. [Cited on page 9]

[21] J. YANG and J. LI, "Application of deep convolution neural network," *International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pp. 229–232, 2017. [Cited on page 9]

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, p. 436–444, 2015. [Cited on page 9]

[23] A. Al-Saffar, H. Tao, and M. Talab, "Review of deep convolution neural network in image classification," *International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications*, pp. 26–31, 2017. [Cited on pages 9 and 10]

[24] T. Donkers, B. Loepp, and J. Ziegler, "Sequential user-based recurrent neural network recommendations," *RecSys '17: Proceedings of the Eleventh ACM Conference on Recommender Systems*, p. 152–160, 2017. [Cited on page 10]

[25] Q. Ma, L. Shen, W. Chen, J. Wang, J. Wei, and Z. Yu, "Functional echo state network for time series classification," *Information Sciences*, vol. 373, pp. 1–20, 2016. [Cited on page 10]

[26] G. Rong, A. Mendez, E. B. Assi, B. Zhao, and M. Sawan, "Artificial intelligence in healthcare: Review and prediction case studies," *Engineering*, vol. 6, pp. 291–301, 2020. [Cited on page 11]

[27] C. Prakash, R. Kumar, and N. Mittal, "Recent developments in human gait research: parameters, approaches, applications, machine learning techniques, datasets and challenges," *Artif Intell Rev*, vol. 49, pp. 1–40, 2018. [Cited on page 11]

[28] W. Tao, T. Liu, R. Zheng, and H. Feng, "Gait analysis using wearable sensors," *Sensors*, vol. 12, pp. 2255–2283, 2012. [Cited on page 11]

[29] A. Kharb, V. Saini, Y. Jain, and S. Dhiman, "A review of gait cycle and its parameters," *International Journal of Computational Engineering Management*, vol. 13, pp. 78–83, 2011. [Cited on page 11]

[30] S. Potluri, S. Ravuri, C. Diedrich, and L. Schega, "Deep learning based gait abnormality detection using wearable sensor system," *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 3613–3619, 2019. [Cited on pages 12 and 13]

[31] A. I. Cuesta-Vargas, A. Galán-Mercant, and J. M. Williams, "The use of inertial sensors system for human motion analysis," *Physical Therapy Reviews*, vol. 15, pp. 462–473, 2010. [Cited on page 12]

[32] M. Fusca, F. Negrini, P. Perego, L. Magoni, F. Molteni, and G. Andreoni, "Validation of a wearable imu system for gait analysis: Protocol and application to a new system," *Applied sciences*, vol. 8, pp. 1–16, 2018. [Cited on page 12]

[33] S. Lee, S. T. Choi, and S. Choi, "Classification of gait type based on deep learning using various sensors with smart insole," *Sensors*, vol. 19, 2019. [Cited on pages 13 and 14]

[34] J. 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*, vol. 15, pp. 27738–27759, 2015. [Cited on page 14]

[35] M. A. Ghorbani, H. A. Zadeh, M. Isazadeh, and O. Terzi, "A comparative study of artificial neural network (mlp,rbf) and support vector machine models for river flow prediction," *Environ Earth Sci*, p. 476, 2016. [Cited on page 14]

[36] J. Menezes and G. A. Barreto, "A new look at nonlinear time series prediction with narx recurrent neural network," *Proceedings of the Ninth Brazilian Symposium on Neural Networks*, 2006. [Cited on page 14]

[37] L. Yan, T. Zhen, J. Kong, L. Wang, and X. Zhou, "Walking gait phase detection based on acceleration signals using voting-weighted integrated neural network," *Hindawi Complexity*, 2020. [Cited on page 15]

[38] J. Kima, S. Hwanga, R. Sohna, Y. Leeb, and Y. Kima, "Development of an active ankle foot orthosis to prevent foot drop and toe drag in hemiplegic patients: A preliminary study," *Applied Bionics and Biomechanics*, vol. 8, pp. 377–384, 2011. [Cited on page 15]

[39] Z. Ahmad, A. S. Khan, C. W. Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study ofmachine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, pp. 1–29, 2020. [Cited on page 15]

[40] A. I. Georgevici and M. Terblanche, "Performance evaluation of dnn with other machine learning techniques in a cluster using apache spark and mllib," *Journal of King Saud University – Computer and Information Sciences*, vol. 34, pp. 1311–1319, 2022. [Cited on page 15]

[41] "nrf52832." `https://www.nordicsemi.com/products/nrf52832`. Accessed: 2022-07-02. [Cited on page 18]

[42] N. Semiconductor, "nrf52832 product specification v1.4," *Datasheet*, 2017. [Cited on page 18]

[43] D. V. Rai and L. Aggarwal, "The study of plantar pressure distribution in normal and pathological foot," *Journal of Medical Physics And Engineering*, vol. 12, pp. 25–34, 2006. [Cited on page 19]

[44] P. Cavanagh and J. Ulbrecht, "Clinical plantar pressure measurement in diabetes: rationale and methodology," *The Foot*, vol. 4, pp. 123–135, 1994. [Cited on page 19]

[45] S. Ostadabbas, A. Saeed, M. Nourani, and M. Pompeo, "Sensor architectural tradeoff for diabetic foot ulcer monitoring," *Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. IEEE Eng. Med. Biol. Soc. Annu. Int. Conf.*, vol. 2012, p. 6687–6690, 2012. [Cited on page 19]

[46] M. J. Mueller, "Efficacy and mechanism of orthotic devices to unload metatarsal heads in people with diabetes and a history of plantar ulcers," *Phys. Ther.*, vol. 86, p. 833–842, 2006. [Cited on page 19]

[47] H. Penny, "Comparison of two pixelated insoles using in-shoe pressure sensors to determine percent offloading: case studies," *J. Wound Care*, vol. 29, pp. 18–26, 2020. [Cited on page 19]

[48] A. Martinez-Nova, J. P. Huerta, and R. Sanchez-Rodriguez, "Cadence, age, and weight as determinants of forefoot plantar pressures using the biofoot in-shoe system," *J. Am. Podiatr. Med. Assoc.*, vol. 98, pp. 302–310, 2008. [Cited on page 19]

[49] A. Boulton, C. Hardisty, R. Betts, C. Franks, R. Worth, J. Ward, and T. Duckworth, "Dynamic foot pressure and other studies as diagnostic and management aids in diabetic neuropathy," *Diabetes Care*, vol. 6, pp. 26–33, 1983. [Cited on page 19]

[50] P. Martins, C. Silva, J. Oliveira, and A. Marques, "Preliminary tests with screen-printed piezoresistive pressure sensors on pet and textile substrates," *IEEE International Conference on Flexible, Printable Sensors and Systems*, 2022. [Cited on page 20]

[51] J. Trautmann, L. Zhou, C. M. Brahms, C. Tunca, C. Ersoy, U. Granacher, and B. Arnrich, "Tripod—a treadmill walking dataset with imu, pressure-distribution and photoelectric data for gait analysis," *Data*, vol. 6, p. 95, 2021. [Cited on pages 31 and 32]

[52] "Fdm-t technical specifications and user manual." `https://www.zebris.de/fileadmin/Editoren/zebris-PDF-Manuals/Medizin/Hardware/Aktuelle_Version/FDM-T_Hardware-Manual_Med_191204_en.pdf`. Accessed: 2020-01-22. [Cited on page 31]

[53] "Dynamic gait analysis on the treadmill." `https://www.zebris.de/en/medical/dynamic-gait-analysis-on-the-treadmill`. Accessed: 2022-06-17. [Cited on page 31]

[54] "The zebris fdm-t system for stance and gait analysis." `https://www.zebris.de/fileadmin/Editoren/zebris-PDF/zebris-Prospekte-EN/FDM-T_Prospekt_en_120901_72dpi.pdf`. Accessed: 2022-06-17. [Cited on page 31]

[55] "Optogait user manual." `http://www.optogait.com/optogaitportal/media/manuals/manual-en.pdf`. Accessed: 2020-01-22. [Cited on page 32]

[56] "System for gait analysis." http://optogait.com/. Accessed: 2022-06-17. [Cited on page 32]

[57] N. Jordan K., Challis J.H. and K.M, "Walking speed influences on gait cycle variability," *Gait Posture*, vol. 26, pp. 128–134, 2007. [Cited on page 32]

[58] C. Meyer, T. Killeen, C. Easthope, A. Curt, M. Bolliger, M. Linnebank, B. Zörner, and L. Filli, "Familiarization with treadmill walking: How much is enough?," *Sci. Rep.*, vol. 9, p. 5232, 2019. [Cited on page 32]

[59] X. Zhan and S. Jiang, "Application of fourier transform and butterworth filter in signal denoisingy," *IEEE 6th International Conference on Intelligent Computing and Signal Processing (ICSP 2021)*, pp. 1277–1281, 2021. [Cited on page 37]

[60] S. K. Manjit Sandhu and J. Kaur, "A study on design and implementation of butterworth, chebyshev and elliptic filter with matlab," *International Journal of Emerging Technologies in Engineering Research (IJETER)*, vol. 4, pp. 111–114, 2016. [Cited on page 37]

[61] P. R. Babu, "Design and implementation of butterworth , chebyshev-1 cic filters for speech signal processing," *Scitech Publication(India) Pvt. Ltd*, vol. 4, 2008. [Cited on page 37]

[62] "scipy.signal.butter." https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html. Accessed: 2022-09-17. [Cited on page 37]

[63] "scipy.signal.filtfilt." https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.filtfilt.html. Accessed: 2022-09-17. [Cited on page 37]

[64] L. Sevgi, "Numerical fourier transforms: Dft and fft," *IEEE Antennas and Propagation Magazine*, vol. 49, pp. 238–243, 2007. [Cited on pages 37 and 38]

[65] J. COOLEY, P. LEWIS, and P. WELCH, "Historical notes on the fast fourier transform," *PROCEEDINGS OF THE IEEE*, vol. 55, pp. 1675–1677, 1967. [Cited on page 38]

[66] U. OBERST, "The fast fourier transform," *SIAM J. CONTROL OPTIM*, vol. 46, pp. 496–540, 2007. [Cited on page 38]

[67] "Fourier transforms (scipy.fft)." https://docs.scipy.org/doc/scipy/tutorial/fft.html. Accessed: 2022-09-17. [Cited on page 38]

[68] G. V. Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model," *Artificial Intelligence Review*, vol. 53, p. 5929–5955, 2020. [Cited on page 41]

[69] K. Smagulova and A. P. James, "A survey on lstm memristive neural network architectures and applications," *Eur. Phys. J. Special Topics*, vol. 228, pp. 2313–2324, 2019. [Cited on page 41]

[70] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," *Technische Universität München*, vol. 91, 1991. [Cited on page 41]

[71] J. Kolen and S. Kremer, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. in: Kremer sc, kolen jf (eds) a field guide to dynamical recurrent networks," *Wiley-IEEE Press*, p. 237–243, 2001. [Cited on page 41]

[72] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, p. 1735–80, 1997. [Cited on page 41]

[73] "Numpy." `https://numpy.org/`. Accessed: 2022-10-04. [Cited on page 42]

[74] "Tensorflow." `https://www.tensorflow.org/`. Accessed: 2022-10-04. [Cited on page 42]

[75] "The sequential model." `https://keras.io/guides/sequential_model/`. Accessed: 2022-07-10. [Cited on page 42]

[76] "Dropout layer." `https://keras.io/api/layers/regularization_layers/dropout/`. Accessed: 2022-07-10. [Cited on page 42]

[77] "Dense layer." `https://keras.io/api/layers/core_layers/dense/`. Accessed: 2022-10-04. [Cited on page 42]

[78] "Model training apis." `https://keras.io/api/models/model_training_apis/`. Accessed: 2022-10-04. [Cited on page 42]

[79] "Conv1d layer." `https://keras.io/api/layers/convolution_layers/convolution1d/`. Accessed: 2022-10-04. [Cited on page 44]

[80] "Maxpooling1d layer." `https://keras.io/api/layers/pooling_layers/max_pooling1d/`. Accessed: 2022-10-04. [Cited on page 44]

[81] "Flatten layer." `https://keras.io/api/layers/reshaping_layers/flatten/`. Accessed: 2022-10-04. [Cited on page 44]

[82] "Convlstm2d layer." `https://keras.io/api/layers/recurrent_layers/conv_lstm2d/`. Accessed: 2022-10-04. [Cited on page 46]

[83] "sklearn.preprocessing.minmaxscaler." https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. Accessed: 2022-08-01. [Cited on page 48]

[84] "sklearn.preprocessing.standardscaler." https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. Accessed: 2022-08-01. [Cited on page 48]

[85] "sklearn.preprocessing.minmaxscaler." https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. Accessed: 2022-10-04. [Cited on page 48]

[86] "sklearn.preprocessing.standardscaler." https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. Accessed: 2022-10-04. [Cited on page 48]

Appendix A

Codes and scripts developed

A.1 Code for monitoring system (nrf52832 board)

```
//inclusion of libraries for the board NRF52832 - PCA10040
#include "app_error.h"
#include "app_util_platform.h"
#include "boards.h"
#include "nrf.h"
#include "nrf_delay.h"
#include "nrf_drv_gpiote.h"
#include "nrf_drv_saadc.h"
#include "nrf_gpio.h"
#include "nrf_log.h"
#include "nrf_log_ctrl.h"
#include "nrf_log_default_backends.h"
#include "nrf_pwr_mgmt.h"
#include <math.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>

// libraries for TIMER
#include "nrfx_ppi.h"
#include "nrfx_timer.h"
#include "nrf_drv_timer.h"
```

```
24
25
26 //Timer
27 #define  TIMER_INTERVAL_MS  10
28 const nrfx_timer_t timer1 = NRFX_TIMER_INSTANCE(1);
29 double teste=0.00; // to count the acquisition time
30
31 // mux delay value in microseconds
32 int delay = 1;
33
34 //-------------------------------------------------//
35
36 // Control pins for the CD74 HC4067 multiplexer
37 #define canal_s0 25
38 #define canal_s1 24
39 #define canal_s2 23
40 #define canal_s3 22
41
42 // work variables
43 int contador = 0;     // to counter
44 int somaADC = 0;      // variable that sums the values
45                       // obtained in the readings
46                       // by the ADC
47 int mediaADC = 0;     // variable that calculates the
48                          average of the ADC readings
49 int Nleituras = 50;   // number of samples for the mean
50 int canalMux = 1;     // variable to change the channel
51                          of the multiplexer (starts at 1)
52 int plantar[9];       // vector for the values
53                          of the 9 insole sensors
54 float pressao = 0.0; // to calculate the pressure
55
56 // Mandatory function for ADC call
57 void saadc_callback(nrf_drv_saadc_evt_t const *p_event) {
58   NRF_LOG_INFO("ADC event");
59 }
60
61 // function to initialize the
62 //ADC peripheral (called in main code below)
63 void ADC_init(void) {
64   ret_code_t err_code;
65   err_code = nrf_drv_saadc_init(NULL, saadc_callback);
66   APP_ERROR_CHECK(err_code);
67
68   // Configure the ADC channel 0
69   //(it is reading through the digital
70   //pin P0.02 of the board)
71   // AIN0 = pin P0.02
72   nrf_saadc_channel_config_t channel_config_0 =
```

```
73    NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(
74    NRF_SAADC_INPUT_AIN0);
75
76    err_code = nrf_drv_saadc_channel_init(
77    0, &channel_config_0);
78    APP_ERROR_CHECK(err_code);
79  } // end function ADC_init
80
81  // function for ADC readings (called in main code below)
82  void MediaLeiturasADC_DP(void) {
83
84    contador = 0; // reset to new cycle
85    somaADC = 0;
86    mediaADC = 0;
87
88    while (contador < Nleituras) // takes x readings
89                                 // from the ADC,
90                                 // from the input pin
91    {
92      ret_code_t err_code;         // tests runtime errors
93      nrf_saadc_value_t valorADC; // variable for ADC value
94                                   //(nrf_saadc_value_t-->
95                                   //type variable defined
96                                   //by nrf_drv_saadc.h)
97      // ADC reading
98      err_code = nrfx_saadc_sample_convert(0, &valorADC);
99      APP_ERROR_CHECK(err_code);
100
101     if (valorADC < 0) {
102       valorADC = 0;
103     }
104
105     somaADC += valorADC; // sum of the current value
106                          // with the previous one
107                          // for the average
108     contador++; // increment the counter
109   }
110   mediaADC = somaADC / Nleituras; //calculates the average
111                                   //without decimals
112                                   //because it is int
113 } // end function readADC
114
115
116 void LeituraSensores(void) {
117   // cycle that will read all sensors
118      in articulation with the multiplexer
119   while (canalMux <= 9) {
120
121     // for sensor 1
```

```
122     if (canalMux == 1) {
123
124       // Multiplexer channel 0 - 0000
125       nrf_gpio_pin_clear(canal_s0);
126       nrf_gpio_pin_clear(canal_s1);
127       nrf_gpio_pin_clear(canal_s2);
128       nrf_gpio_pin_clear(canal_s3);
129       nrf_delay_us(delay);
130
131       // ADC reading
132       MediaLeiturasADC_DP();
133       pressao = powf(2.7183, (mediaADC + 45.574)/378.88);
134       if (pressao < 1) {
135         pressao = 0;
136       }
137       plantar[0] = pressao;
138     }
139
140     // for sensor 2
141     if (canalMux == 2) {
142
143       // Multiplexer channel 1 - 1000
144       nrf_gpio_pin_set(canal_s0);
145       nrf_gpio_pin_clear(canal_s1);
146       nrf_gpio_pin_clear(canal_s2);
147       nrf_gpio_pin_clear(canal_s3);
148       nrf_delay_us(delay);
149
150       // ADC reading
151       MediaLeiturasADC_DP();
152       pressao = powf(2.7183, (mediaADC + 88.2) / 495.73);
153       if (pressao < 1) {
154         pressao = 0;
155       }
156       plantar[1] = pressao;
157     }
158
159     // for sensor 3
160     if (canalMux == 3) {
161
162       // Multiplexer channel 2 - 0100
163       nrf_gpio_pin_clear(canal_s0);
164       nrf_gpio_pin_set(canal_s1);
165       nrf_gpio_pin_clear(canal_s2);
166       nrf_gpio_pin_clear(canal_s3);
167       nrf_delay_us(delay);
168
169       // ADC reading
170       MediaLeiturasADC_DP();
```

```
171       pressao = powf(2.7183, (mediaADC - 28.4)/394.3);
172       if (pressao < 1) {
173         pressao = 0;
174       }
175       plantar[2] = pressao;
176     }
177
178     // for sensor 4
179     if (canalMux == 4) {
180
181       // Multiplexer channel 3 - 1100
182       nrf_gpio_pin_set(canal_s0);
183       nrf_gpio_pin_set(canal_s1);
184       nrf_gpio_pin_clear(canal_s2);
185       nrf_gpio_pin_clear(canal_s3);
186       nrf_delay_us(delay);
187
188       // ADC reading
189       MediaLeiturasADC_DP();
190       pressao = powf(2.7183, (mediaADC + 117.68)/372.77);
191       if (pressao < 1) {
192         pressao = 0;
193       }
194       plantar[3] = pressao;
195     }
196
197     // for sensor 5, the only one that is +- linear
198     if (canalMux == 5) {
199
200       // Multiplexer channel 4 - 0010
201       nrf_gpio_pin_clear(canal_s0);
202       nrf_gpio_pin_clear(canal_s1);
203       nrf_gpio_pin_set(canal_s2);
204       nrf_gpio_pin_clear(canal_s3);
205       nrf_delay_us(delay);
206
207       // ADC reading
208       MediaLeiturasADC_DP();
209       if (mediaADC < 137.83) {
210         pressao = 0;
211       } else {
212         pressao = (mediaADC - 137.83)/4.0516;
213       }
214       if (pressao < 1) {
215         pressao = 0;
216       }
217       plantar[4] = pressao;
218     }
219
```

```
220     // for sensor 6
221     if (canalMux == 6) {
222
223       // Multiplexer channel 5 - 1010
224       nrf_gpio_pin_set(canal_s0);
225       nrf_gpio_pin_clear(canal_s1);
226       nrf_gpio_pin_set(canal_s2);
227       nrf_gpio_pin_clear(canal_s3);
228       nrf_delay_us(delay);
229
230
231       // ADC reading
232       MediaLeiturasADC_DP();
233       pressao = powf(2.7183, (mediaADC - 43.779)/407.99);
234       if (pressao < 1) {
235         pressao = 0;
236       }
237       plantar[5] = pressao;
238     }
239
240     // for sensor 7
241     if (canalMux == 7) {
242
243       // Multiplexer channel 6 - 0110
244       nrf_gpio_pin_clear(canal_s0);
245       nrf_gpio_pin_set(canal_s1);
246       nrf_gpio_pin_set(canal_s2);
247       nrf_gpio_pin_clear(canal_s3);
248       nrf_delay_us(delay);
249
250       // ADC reading
251       MediaLeiturasADC_DP();
252       pressao = powf(2.7183, (mediaADC + 115.28)/359.68);
253       if (pressao < 1) {
254         pressao = 0;
255       }
256       plantar[6] = pressao;
257     }
258
259     // for sensor 8
260     if (canalMux == 8) {
261
262       // Multiplexer channel 7 - 1110
263       nrf_gpio_pin_set(canal_s0);
264       nrf_gpio_pin_set(canal_s1);
265       nrf_gpio_pin_set(canal_s2);
266       nrf_gpio_pin_clear(canal_s3);
267       nrf_delay_us(delay);
268
```

```
269       // ADC reading
270       MediaLeiturasADC_DP();
271       pressao = powf(2.7183, (mediaADC - 26.41)/495.33);
272       if (pressao < 1) {
273         pressao = 0;
274       }
275       plantar[7] = pressao;
276     }
277
278     // for sensor 9
279     if (canalMux == 9) {
280
281       // Multiplexer channel 8 - 0001
282       nrf_gpio_pin_clear(canal_s0);
283       nrf_gpio_pin_clear(canal_s1);
284       nrf_gpio_pin_clear(canal_s2);
285       nrf_gpio_pin_set(canal_s3);
286       nrf_delay_us(delay);
287
288       // ADC reading
289       MediaLeiturasADC_DP();
290       pressao = powf(2.7183, (mediaADC - 250.52)/495.27);
291       if (pressao < 1) {
292         pressao = 0;
293       }
294       plantar[8] = pressao;
295     }
296     // end of channel reading
297     canalMux = canalMux + 1; //increment channel
298                              //counter --> n+1
299   }
300   canalMux = 1; // resets the channel (starts at 1)
301                 //to enter another cycle of 9 sensors
302 } // end function ReadingSensors
303
304
305 static void timer1_callback(nrf_timer_event_t event_type,
306 void* p_context){
307     switch(event_type)
308     {
309       case NRF_TIMER_EVENT_COMPARE1:
310
311         teste = 10;
312         teste = teste/1000;
313         tempo = tempo + teste;
314         LeituraSensores();
315
316
317         // print data
```

```
318         NRF_LOG_RAW_INFO(NRF_LOG_FLOAT_MARKER,
319         NRF_LOG_FLOAT(tempo));
320
321
322         NRF_LOG_RAW_INFO(  %d  %d  %d  %d  %d,
323         plantar[0], plantar[1], plantar[2], plantar[3],
324         plantar[4]);
325
326         NRF_LOG_RAW_INFO(  %d  %d  %d  %d  \n,
327         plantar[5], plantar[6], plantar[7], plantar[8]);
328
329         break;
330       default:
331         break;
332     }
333
334 }
335
336
337 // main program execution code
338 int main(void) {
339
340   // for error detection
341   uint32_t err_code = NRF_LOG_INIT(NULL);
342   APP_ERROR_CHECK(err_code);
343
344   nrf_delay_ms(500);
345
346   // start the output terminal serial monitor
347   NRF_LOG_DEFAULT_BACKENDS_INIT();
348
349   nrf_delay_ms(500);
350
351   // for detecting errors in the board's
352   //power management system
353   ret_code_t ret_code = nrf_pwr_mgmt_init();
354   APP_ERROR_CHECK(ret_code);
355
356   nrf_delay_ms(500);
357
358   // configure these pins as output (22 to 25) for the mux
359   nrf_gpio_range_cfg_output(22, 25);
360
361   // delays to stabilize the system and not crash
362   nrf_delay_ms(500);
363
364   // order to start ADC (function defined above)
365   ADC_init();
366
```

```
367     nrf_delay_ms(1000);
368
369     NRF_LOG_INFO(Testes marcha.\n);
370     NRF_LOG_FLUSH();
371     nrf_delay_ms(1000);
372     NRF_LOG_FLUSH();
373     NRF_LOG_INFO(A iniciar...\n);
374     NRF_LOG_FLUSH();
375     nrf_delay_ms(1000);
376     NRF_LOG_FLUSH();
377     NRF_LOG_RAW_INFO(Tempo(s) Sensor1 Sensor2 Sensor3
378     Sensor4 Sensor5 Sensor6 Sensor7 Sensor8 Sensor9  \n);
379     NRF_LOG_FLUSH();
380
381     /////FOR TIMER///////
382     uint32_t time_ticks;
383
384     nrfx_timer_config_t timer1_config =
385     NRFX_TIMER_DEFAULT_CONFIG;
386     timer1_config.frequency = NRF_TIMER_FREQ_1MHz;
387     timer1_config.mode = NRF_TIMER_MODE_TIMER;
388     timer1_config.bit_width = NRF_TIMER_BIT_WIDTH_32;
389     timer1_config.interrupt_priority = 6;
390     APP_ERROR_CHECK(nrfx_timer_init(&timer1, &timer1_config,
391     timer1_callback));
392
393     time_ticks = nrfx_timer_ms_to_ticks(&timer1,
394     TIMER_INTERVAL_MS);
395
396     nrfx_timer_extended_compare(&timer1,
397     NRF_TIMER_CC_CHANNEL1, time_ticks,
398     NRF_TIMER_SHORT_COMPARE1_CLEAR_MASK, true);
399
400     nrfx_timer_enable(&timer1);
401
402     // here enter the program loop --> ***LOOPING CYCLE***
403     while (true) {
404       NRF_LOG_FLUSH();
405     }
406 }
```

A.2 Script to generate public dataset

A.2.1 Division of pressure files for each user into walking cycles

```
import csv
import pandas as pd
import numpy as np

filename=1
contar=0
a = np.array([])
h=0
o=0
ultimo_v=0
t=1
flag=0
classification = np.array([])

classification= np.append('Sub_YU', '+20') #ID PART.,SPEED

add_array = np.append('0','0')
tamanho=225
m=0

with open(
    './CSVs/Sub_YU_PWS+20/butterfly_force_curve-L.csv',
    'r') as infile:
        reader = list(csv.reader(infile, delimiter=","))
        array_reader = np.array(reader[4:], dtype=str)

array_reader = np.insert(array_reader,0,classification,0)

for i in range(len(array_reader)):

    if not (array_reader[i][1]!='' and
    array_reader[i-1][1]=='') and i!=len(array_reader)-1:

            contar+=1
    else:
        if t!=1:
            contar+=1
        a=np.append(a,contar)
        t+=1
        contar=0
i+=1

for v in range(len(array_reader)):

```

```
45      if o>=len(a):
46          o=0
47          flag=1
48
49      if flag==0 and h == int(a[o]):
50
51          final = open(
52          './New_CSVs/Sub_YU/Pressao/PWS+20/Dados_v'+
53          str(filename)+'.csv', 'w+', newline='')
54          with final:
55              write=csv.writer(final)
56              if o==0:
57                  write.writerows(
58                  array_reader[0:0+int(a[o])])
59
60              if o==len(a)-1:
61                  write.writerow(classification)
62                  write.writerows(
63                  array_reader[ultimo_v:ultimo_v+
64                  int(a[o])+1])
65
66              else:
67                  write.writerow(classification)
68                  write.writerows(
69                  array_reader[ultimo_v:ultimo_v+
70                  int(a[o])])
71                  if tamanho>int(a[o]):
72                      print('entrou')
73                      dif = tamanho-int(a[o])
74                      while m<dif:
75                          write.writerow(add_array)
76                          m+=1
77                      m=0
78
79          filename+=1
80          ultimo_v=v
81          o+=1
82          h=0
83
84      h+=1
85  v+1
```

A.2.2 Final CSV with all cycles (dataset)

```
1  ####FINAL CSV WITH ALL DATA####
2  ## CSV FORMAT:
3  ##[PARTICIPANT_ID, SPEED, PRESSURE1,...PRESSURE225]
```

```
4
5  import csv
6  import numpy as np
7  import pathlib
8  import pandas as pd
9
10
11 subjects = ['1', '2', '3', '4', '5', '6', '7', '8', '9',
12 '10', '11', '12', '13', '14', '15']
13 velocity = ['0', '1']
14
15 paths = ['./New_CSVs/Sub_AL/Pressao/PWS-20',
16         './New_CSVs/Sub_AL/Pressao/PWS+20',
17         './New_CSVs/Sub_BK/Pressao/PWS-20',
18         './New_CSVs/Sub_BK/Pressao/PWS+20',
19         './New_CSVs/Sub_CP/Pressao/PWS-20',
20         './New_CSVs/Sub_CP/Pressao/PWS+20',
21         './New_CSVs/Sub_DF/Pressao/PWS-20',
22         './New_CSVs/Sub_DF/Pressao/PWS+20',
23         './New_CSVs/Sub_EN/Pressao/PWS-20',
24         './New_CSVs/Sub_EN/Pressao/PWS+20',
25         './New_CSVs/Sub_FZ/Pressao/PWS-20',
26         './New_CSVs/Sub_FZ/Pressao/PWS+20',
27         './New_CSVs/Sub_GK/Pressao/PWS-20',
28         './New_CSVs/Sub_GK/Pressao/PWS+20',
29         './New_CSVs/Sub_HA/Pressao/PWS-20',
30         './New_CSVs/Sub_HA/Pressao/PWS+20',
31         './New_CSVs/Sub_KP/Pressao/PWS-20',
32         './New_CSVs/Sub_KP/Pressao/PWS+20',
33         './New_CSVs/Sub_LU/Pressao/PWS-20',
34         './New_CSVs/Sub_LU/Pressao/PWS+20',
35         './New_CSVs/Sub_OD/Pressao/PWS-20',
36         './New_CSVs/Sub_OD/Pressao/PWS+20',
37         './New_CSVs/Sub_PB/Pressao/PWS-20',
38         './New_CSVs/Sub_PB/Pressao/PWS+20',
39         './New_CSVs/Sub_RW/Pressao/PWS-20',
40         './New_CSVs/Sub_RW/Pressao/PWS+20',
41         './New_CSVs/Sub_SN/Pressao/PWS-20',
42         './New_CSVs/Sub_SN/Pressao/PWS+20',
43         './New_CSVs/Sub_YU/Pressao/PWS-20',
44         './New_CSVs/Sub_YU/Pressao/PWS+20',]
45
46
47 ####See how many .csv files each folder has
48 initial_count=0
49 files_in_folder = np.array([])
50 i=0
51
52 x = np.array([])
```

```
53
54 for folder in range(len(paths)):
55
56     for path in pathlib.Path(paths[folder]).iterdir():
57         if path.is_file():
58             initial_count+=1
59
60     files_in_folder=np.append(files_in_folder,
61     initial_count)
62     initial_count=0
63 i+=1
64
65 #####Open .csv files########
66 final = open('./All_data.csv', 'w+', newline='')
67 write = csv.writer(final)
68
69 vel = 0
70 i=2
71 h=0
72 s=0
73 l=1
74 for h in range(len(paths)):
75   while i < files_in_folder[h]:
76     with open(paths[h]+'/Dados_v'+str(i)+'.csv',
77     'r') as infile:
78         output = list(csv.reader(infile, delimiter=','))
79
80         array_output = np.array(output[1:], dtype=str)
81         array_output = np.delete(array_output, 0, 1)
82
83     if vel == 0:
84         write.writerow([subjects[s], velocity[0],
85             array_output[0][0], array_output[1][0],
86             array_output[2][0], array_output[3][0],
87             array_output[4][0], array_output[5][0],
88             array_output[6][0], array_output[7][0],
89             array_output[8][0], array_output[9][0],
90             array_output[10][0], array_output[11][0],
91             array_output[12][0], array_output[13][0],
92             array_output[14][0], array_output[15][0],
93             array_output[16][0], array_output[17][0],
94             array_output[18][0], array_output[19][0],
95             array_output[20][0], array_output[21][0],
96             array_output[22][0], array_output[23][0],
97             array_output[24][0], array_output[25][0],
98             array_output[26][0], array_output[27][0],
99             array_output[28][0], array_output[29][0],
100            array_output[30][0], array_output[31][0],
101            array_output[32][0], array_output[33][0],
```

```
102             array_output[34][0], array_output[35][0],
103             array_output[36][0], array_output[37][0],
104             array_output[38][0], array_output[39][0],
105             array_output[40][0], array_output[41][0],
106             array_output[42][0], array_output[43][0],
107             array_output[44][0], array_output[45][0],
108             array_output[46][0], array_output[47][0],
109             array_output[48][0], array_output[49][0],
110             array_output[50][0], array_output[51][0],
111             array_output[52][0], array_output[53][0],
112             array_output[54][0], array_output[55][0],
113             array_output[56][0], array_output[57][0],
114             array_output[58][0], array_output[59][0],
115             array_output[60][0], array_output[61][0],
116             array_output[62][0], array_output[63][0],
117             array_output[64][0], array_output[65][0],
118             array_output[66][0], array_output[67][0],
119             array_output[68][0], array_output[69][0],
120             array_output[70][0], array_output[71][0],
121             array_output[72][0], array_output[73][0],
122             array_output[74][0], array_output[75][0],
123             array_output[76][0], array_output[77][0],
124             array_output[78][0], array_output[79][0],
125             array_output[80][0], array_output[81][0],
126             array_output[82][0], array_output[83][0],
127             array_output[84][0], array_output[85][0],
128             array_output[86][0], array_output[87][0],
129             array_output[88][0], array_output[89][0],
130             array_output[90][0], array_output[91][0],
131             array_output[92][0], array_output[93][0],
132             array_output[94][0], array_output[95][0],
133             array_output[96][0], array_output[97][0],
134             array_output[98][0], array_output[99][0],
135             array_output[100][0], array_output[101][0],
136             array_output[102][0], array_output[103][0],
137             array_output[104][0], array_output[105][0],
138             array_output[106][0], array_output[107][0],
139             array_output[108][0], array_output[109][0],
140             array_output[110][0], array_output[111][0],
141             array_output[112][0], array_output[113][0],
142             array_output[114][0], array_output[115][0],
143             array_output[116][0], array_output[117][0],
144             array_output[118][0], array_output[119][0],
145             array_output[120][0], array_output[121][0],
146             array_output[122][0], array_output[123][0],
147             array_output[124][0], array_output[125][0],
148             array_output[126][0], array_output[127][0],
149             array_output[128][0], array_output[129][0],
150             array_output[130][0], array_output[131][0],
```

```
151             array_output[132][0], array_output[133][0],
152             array_output[134][0], array_output[135][0],
153             array_output[136][0], array_output[137][0],
154             array_output[138][0], array_output[139][0],
155             array_output[140][0], array_output[141][0],
156             array_output[142][0], array_output[143][0],
157             array_output[144][0], array_output[145][0],
158             array_output[146][0], array_output[147][0],
159             array_output[148][0], array_output[149][0],
160             array_output[150][0], array_output[151][0],
161             array_output[152][0], array_output[153][0],
162             array_output[154][0], array_output[155][0],
163             array_output[156][0], array_output[157][0],
164             array_output[158][0], array_output[159][0],
165             array_output[160][0], array_output[161][0],
166             array_output[162][0], array_output[163][0],
167             array_output[164][0], array_output[165][0],
168             array_output[166][0], array_output[167][0],
169             array_output[168][0], array_output[169][0],
170             array_output[170][0], array_output[171][0],
171             array_output[172][0], array_output[173][0],
172             array_output[174][0], array_output[175][0],
173             array_output[176][0], array_output[177][0],
174             array_output[178][0], array_output[179][0],
175             array_output[180][0], array_output[181][0],
176             array_output[182][0], array_output[183][0],
177             array_output[184][0], array_output[185][0],
178             array_output[186][0], array_output[187][0],
179             array_output[188][0], array_output[189][0],
180             array_output[190][0], array_output[191][0],
181             array_output[192][0], array_output[193][0],
182             array_output[194][0], array_output[195][0],
183             array_output[196][0], array_output[197][0],
184             array_output[198][0], array_output[199][0],
185             array_output[200][0], array_output[201][0],
186             array_output[202][0], array_output[203][0],
187             array_output[204][0], array_output[205][0],
188             array_output[206][0], array_output[207][0],
189             array_output[208][0], array_output[209][0],
190             array_output[210][0], array_output[211][0],
191             array_output[212][0], array_output[213][0],
192             array_output[214][0], array_output[215][0],
193             array_output[216][0], array_output[217][0],
194             array_output[218][0], array_output[219][0],
195             array_output[220][0], array_output[221][0],
196             array_output[222][0], array_output[223][0],
197             array_output[224][0]])
198
199
```

```
200     else:
201         write.writerow([ subjects[s], velocity[1] ,
202             array_output[0][0], array_output[1][0],
203             array_output[2][0], array_output[3][0],
204             array_output[4][0], array_output[5][0],
205             array_output[6][0], array_output[7][0],
206             array_output[8][0], array_output[9][0],
207             array_output[10][0], array_output[11][0],
208             array_output[12][0], array_output[13][0],
209             array_output[14][0], array_output[15][0],
210             array_output[16][0], array_output[17][0],
211             array_output[18][0], array_output[19][0],
212             array_output[20][0], array_output[21][0],
213             array_output[22][0], array_output[23][0],
214             array_output[24][0], array_output[25][0],
215             array_output[26][0], array_output[27][0],
216             array_output[28][0], array_output[29][0],
217             array_output[30][0], array_output[31][0],
218             array_output[32][0], array_output[33][0],
219             array_output[34][0], array_output[35][0],
220             array_output[36][0], array_output[37][0],
221             array_output[38][0], array_output[39][0],
222             array_output[40][0], array_output[41][0],
223             array_output[42][0], array_output[43][0],
224             array_output[44][0], array_output[45][0],
225             array_output[46][0], array_output[47][0],
226             array_output[48][0], array_output[49][0],
227             array_output[50][0], array_output[51][0],
228             array_output[52][0], array_output[53][0],
229             array_output[54][0], array_output[55][0],
230             array_output[56][0], array_output[57][0],
231             array_output[58][0], array_output[59][0],
232             array_output[60][0], array_output[61][0],
233             array_output[62][0], array_output[63][0],
234             array_output[64][0], array_output[65][0],
235             array_output[66][0], array_output[67][0],
236             array_output[68][0], array_output[69][0],
237             array_output[70][0], array_output[71][0],
238             array_output[72][0], array_output[73][0],
239             array_output[74][0], array_output[75][0],
240             array_output[76][0], array_output[77][0],
241             array_output[78][0], array_output[79][0],
242             array_output[80][0], array_output[81][0],
243             array_output[82][0], array_output[83][0],
244             array_output[84][0], array_output[85][0],
245             array_output[86][0], array_output[87][0],
246             array_output[88][0], array_output[89][0],
247             array_output[90][0], array_output[91][0],
248             array_output[92][0], array_output[93][0],
```

```
249             array_output[94][0], array_output[95][0],
250             array_output[96][0], array_output[97][0],
251             array_output[98][0], array_output[99][0],
252             array_output[100][0], array_output[101][0],
253             array_output[102][0], array_output[103][0],
254             array_output[104][0], array_output[105][0],
255             array_output[106][0], array_output[107][0],
256             array_output[108][0], array_output[109][0],
257             array_output[110][0], array_output[111][0],
258             array_output[112][0], array_output[113][0],
259             array_output[114][0], array_output[115][0],
260             array_output[116][0], array_output[117][0],
261             array_output[118][0], array_output[119][0],
262             array_output[120][0], array_output[121][0],
263             array_output[122][0], array_output[123][0],
264             array_output[124][0], array_output[125][0],
265             array_output[126][0], array_output[127][0],
266             array_output[128][0], array_output[129][0],
267             array_output[130][0], array_output[131][0],
268             array_output[132][0], array_output[133][0],
269             array_output[134][0], array_output[135][0],
270             array_output[136][0], array_output[137][0],
271             array_output[138][0], array_output[139][0],
272             array_output[140][0], array_output[141][0],
273             array_output[142][0], array_output[143][0],
274             array_output[144][0], array_output[145][0],
275             array_output[146][0], array_output[147][0],
276             array_output[148][0], array_output[149][0],
277             array_output[150][0], array_output[151][0],
278             array_output[152][0], array_output[153][0],
279             array_output[154][0], array_output[155][0],
280             array_output[156][0], array_output[157][0],
281             array_output[158][0], array_output[159][0],
282             array_output[160][0], array_output[161][0],
283             array_output[162][0], array_output[163][0],
284             array_output[164][0], array_output[165][0],
285             array_output[166][0], array_output[167][0],
286             array_output[168][0], array_output[169][0],
287             array_output[170][0], array_output[171][0],
288             array_output[172][0], array_output[173][0],
289             array_output[174][0], array_output[175][0],
290             array_output[176][0], array_output[177][0],
291             array_output[178][0], array_output[179][0],
292             array_output[180][0], array_output[181][0],
293             array_output[182][0], array_output[183][0],
294             array_output[184][0], array_output[185][0],
295             array_output[186][0], array_output[187][0],
296             array_output[188][0], array_output[189][0],
297             array_output[190][0], array_output[191][0],
```

```
298             array_output[192][0], array_output[193][0],
299             array_output[194][0], array_output[195][0],
300             array_output[196][0], array_output[197][0],
301             array_output[198][0], array_output[199][0],
302             array_output[200][0], array_output[201][0],
303             array_output[202][0], array_output[203][0],
304             array_output[204][0], array_output[205][0],
305             array_output[206][0], array_output[207][0],
306             array_output[208][0], array_output[209][0],
307             array_output[210][0], array_output[211][0],
308             array_output[212][0], array_output[213][0],
309             array_output[214][0], array_output[215][0],
310             array_output[216][0], array_output[217][0],
311             array_output[218][0], array_output[219][0],
312             array_output[220][0], array_output[221][0],
313             array_output[222][0], array_output[223][0],
314             array_output[224][0]])
315
316         i+=1
317     i=2
318     l+=1
319     if vel==0:
320         vel=1
321     else:
322         vel=0
323
324     if l==3:
325         s+=1
326         l=1
327
328     h+=1
```

A.3 Script for USB reading

```
1  import serial
2
3  port = "COM15"
4
5  ser = serial.Serial(port, 115200)
6
7  f=open('./Ensaios_Carolina/four.txt', 'w')
8  while True:
9
10     data = ser.readline()
11     teste = str(data).replace("b'", "")
12     teste = teste.replace("\r\n", "")
```

```
13      f.writelines(teste+"\n")
14      print(teste)
```

A.4 FT Filter

```
1  def fft_denoiser(x, n_components, to_real=True):
2      n=len(x)
3
4      fft = np.fft.fft(x,n)
5
6      PSD = fft*np.conj(fft) / n
7
8      _mask = PSD > n_components
9      fft = _mask * fft
10
11     clean_data = np.fft.ifft(fft)
12
13     if to_real:
14         clean_data = clean_data.real
15
16     return clean_data
```

A.5 Script to generate system dataset

A.5.1 Libraries

```
1
2  from scipy.signal import butter, filtfilt
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import csv
6  import pandas as pd
7  from scipy.signal import butter, find_peaks
8  import scipy
9  import os
```

A.5.2 Function for applying BW filter

```
1  def butter_lowpass( data ,lowcut, fs, order, nyq):
2      low = lowcut / nyq
```

```
    b, a = butter(order, low, btype='low', analog=False)
    y = filtfilt(b, a, data)
    return y
```

A.5.3 Script to obtain vector to filter

```

with open('./Ensaios_Finais/Ensaios_Catarina_Prima/
lento.csv', 'r') as f:
    data = list(csv.reader(f, delimiter=';'))

data=np.array(data)

teste=np.array(data[5:])

dados = []
tempo = []
soma =0

for row in teste:
    ded = row
    soma = int(ded[1])+int(ded[2])+int(ded[3])+int(ded[4])
    +int(ded[5])+int(ded[6])+int(ded[7])+int(ded[8])
    +int(ded[9])
    tempo.append(float(ded[0]))
    dados.append(soma)
    soma=0

plt.xlim(xmin=0, xmax=5)
plt.title(Sinal original)
plt.ylabel(Soma de pressoes (kPa))
plt.xlabel(Tempo (s))
plt.plot(tempo, dados)
```

A.5.4 Script to filter data

```

fs = 100
lowcut = 10.0
nyq = 0.5 * fs

novos_dados = butter_lowpass(dados,lowcut, fs,
order=2, nyq=nyq)

```

```
plt.xlim(xmin=0, xmax=5)
plt.title(Filtro BW de ordem 2 com fc=10 Hz)
plt.ylabel(Soma de pressoes (kPa))
plt.xlabel(Tempo (s))
plt.plot(tempo, novos_dados)
```

A.5.5 Script to identify maximum and minimum points

```

dados = np.array(novos_dados)

max_value = dados.max()
min_value = dados.min()

peaks, _ = find_peaks(dados, height=500)
peaks2, _ = find_peaks(-dados+max_value, height=1200)
plt.xlim(xmin=0, xmax=1000)
plt.plot(dados)
plt.plot(peaks,dados[peaks],"o")
plt.plot(peaks2,dados[peaks2],"o")
```

A.5.6 Script to delete first cycle

```
vetor_final = []
flag = 0

for t in range(len(dados)):

    if flag==1:
        vetor_final.append(dados[t])

    if dados[t] == dados[peaks2][2]:
        vetor_final.append(dados[t])
        posicao = t
        flag = 1

plt.xlim(xmin=0, xmax=5)
plt.title(Sinal final)
plt.ylabel(Soma de pressoes (kPa))
plt.xlabel(Tempo (s))
plt.plot(tempo[posicao:], vetor_final)
```

A.5.7 Script for final vector formation with its maximums and minimums recorded

```
dados_finais = np.array(vetor_final)

max_value = dados_finais.max()
min_value = dados_finais.min()

maximos, _ = find_peaks(dados_finais, height=500)
minimos, _ = find_peaks(-dados_finais+max_value,
height = 1200)
plt.xlim(xmin=0, xmax=1000)
plt.plot(dados_finais)
plt.plot(maximos,dados_finais[maximos],"o")
plt.plot(minimos,dados_finais[minimos],"o")
```

A.5.8 Script for counting minimums between two maximums

```
i=0
count_max=0
o=0
count_min=0
vetor_count = []

for t in range(len(dados_finais)):

  if i<len(dados_finais[maximos]) and dados_finais[t] ==
  dados_finais[maximos][i]:
    if count_min> 0:
      vetor_count.append(count_min)
      count_min=0
      count_max=1
    else:
      count_max+=1
    i+=1

  if o<len(dados_finais[minimos]) and dados_finais[t] ==
  dados_finais[minimos][o]:
    if count_max > 0:
      count_min+=1
      o+=1
```

A.5.9 Script to create vector with minimums that correspond to the end of the cycle

```
1
2 new_m = []
3
4 valor_anterior = 0
5
6 novo =0
7
8 for r in range(len(vetor_count)):
9
10  if r ==0:
11
12   novo = dados_finais[minimos][valor_anterior +
13   vetor_count[r]-1]
14
15   new_m.append(novo)
16
17   valor_anterior = valor_anterior + vetor_count[r]-1
18
19  else:
20   novo = dados_finais[minimos][valor_anterior +
21   vetor_count[r]]
22
23   new_m.append(novo)
24
25   valor_anterior = valor_anterior + vetor_count[r]
```

A.5.10 Script for splitting CSVs by cycle

```
1
2 vt_csv = []
3
4 s=0
5
6 tamanho = 0
7
8 tamanho_ciclo=0
9
10 versao=1
11
12 for t in range(len(dados_finais)):
13     if s<len(new_m) and dados_finais[t] == new_m[s]:
14         vt_csv.append(dados_finais[t])
```

```
15
16          tamanho=len(vt_csv)
17
18          if tamanho>tamanho_ciclo:
19              tamanho_ciclo=tamanho
20
21          f = open(./Ensaios_Finais/Ensaios_Catarina_Prima/
22          Ciclos_Lento/All_data+str(versao)+'.csv', 'w',
23          newline='')
24
25          w=csv.writer(f)
26          w.writerow(vt_csv)
27          vt_csv = []
28          s+=1
29          versao+=1
30
31      else:
32          vt_csv.append(dados_finais[t])
```

A.5.11 Script to create final CSV (dataset)

```
1  from csv import writer
2  from importlib.resources import path
3
4
5  r = open('./Ensaios_Finais/Final_CSV.csv', 'w',
6  newline='')
7  write = csv.writer(r)
8
9  paths = ['./Ensaios_Finais/Ensaios_Catarina_ISEP/',
10         './Ensaios_Finais/Ensaios_Catarina_Prima/',
11         './Ensaios_Finais/Ensaios_Magda/',
12         './Ensaios_Finais/Ensaios_Marco/',
13         './Ensaios_Finais/Ensaios_Maria/',
14         './Ensaios_Finais/Ensaios_Nilza/',
15          './Ensaios_Finais/Ensaios_Rui/']
16
17 paths_velocity = ['Ciclos_Lento', 'Ciclos_Rapido']
18
19 id_participante = 1
20
21 velocity = 0
22
23 counting=0
24
25 size=0
26
```

```
add_columns=0

g=0


for i in range(len(paths)):
    for t in range(len(paths_velocity)):
        for file in os.listdir(paths[i]+
            paths_velocity[t]):
            if os.path.isfile(os.path.join(str(paths[i]+
               paths_velocity[t]), file)):
                counting+=1

        for d in range(counting):
            with open(str(paths[i]+paths_velocity[t]+
                 '/All_data'+str(d+1)+'.csv'), 'r') as r:

                  read_line = list(csv.reader(r,
                  delimiter=','))

                  print(len(read_line[0]))
                  size = len(read_line[0])
                  add_columns=200-size
                  print(add_columns)
                  read_line=np.array(read_line)

                  while g < add_columns:
                       read_line=np.append(read_line,
                       '0.0')
                       g+=1

                  g=0

                  info=np.array([[id_participante,
                  velocity]])

                  write_line=np.append(info, read_line)

                  write_line=list(write_line)
                  print(write_line)
                  write.writerow(write_line)

        counting=0

        if velocity==0:
            velocity=1
        else:
            velocity=0

```

```
76      id_participante+=1
```

A.6 LSTM Model

A.6.1 Without standardization or normalization

Libraries

```
1           import csv
2           import numpy as np
3           from numpy import mean
4           from numpy import std
5
6           import tensorflow as tf
7           from tensorflow.keras.models import Sequential
8           from tensorflow.keras.layers import Dense
9           from tensorflow.keras.layers import Dropout
10          from tensorflow.keras.layers import LSTM
```

Function load_dataset3()

```
1       def load_dataset3():
2           P = np.genfromtxt(C:/Users/carol/OneDrive/Ambiente
3           de Trabalho/Ensaios_Finais/Final_CSV.csv,
4           delimiter=',', missing_values=None,
5           filling_values=0.0)
6
7           d = P.shape[0]
8
9           P = np.append(P,
10          np.zeros((d, 256 - P.shape[1] + 2)), axis=1)
11          np.random.shuffle(P)
12
13          V = P[:,1:2]
14          P = P[:,2:]
15
16          P = P.reshape((P.shape[0], P.shape[1], 1))
17
18          V = V.reshape(V.shape[0])
19          V = tf.keras.utils.to_categorical(V)
20
21          divide = round(d*0.80)
22
```

```
23          trainX = P[:divide]
24          testX = P[divide:]
25          trainY = V[:divide]
26          testY = V[divide:]
27
28          return trainX, testX, trainY, testY
```

Function evaluate_model()

```
1       def evaluate_model (trainX, trainY, testX, testY):
2           verbose, epochs, batch_size=0,25,64
3           n_timesteps = trainX.shape[1]
4           n_features = trainX.shape[2]
5           n_outputs = trainY.shape[1]
6           model = Sequential()
7           model.add(LSTM(100,
8           input_shape=(n_timesteps, n_features)))
9           model.add(Dropout(0.55))
10          model.add(Dense(100, activation='relu'))
11          model.add(Dense(n_outputs,
12          activation = 'softmax'))
13          model.compile(loss='categorical_crossentropy',
14          optimizer='adam', metrics=['accuracy'])
15
16          model.fit(trainX, trainY, epochs=epochs,
17          batch_size=batch_size, verbose=verbose)
18
19          _, accuracy = model.evaluate(testX, testY,
20          batch_size = batch_size, verbose = 1)
21
22          return accuracy
```

Function summarize_results()

```
1       def summarize_results(scores):
2           print(scores)
3           m, s = mean(scores), std(scores)
4           print('Accuracy:%.3f%%(+/-%.3f)' % (m,s))
```

Function run_experiment()

```
    # run an experiment
    def run_experiment(repeats=5):
      # load data
      trainX, testX, trainY, testY = load_dataset3()
      # repeat experiment
      scores = list()
      for r in range(repeats):
        score = evaluate_model(trainX,
        trainY, testX, testY)
        score = score * 100.0
        print('>#%d:%.3f' % (r+1, score))
        scores.append(score)
      summarize_results(scores)
```

A.6.2 Standardization

Libraries

```
import csv
import numpy as np
from numpy import mean
from numpy import std

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LSTM
from sklearn.preprocessing import  StandardScaler
import pandas as pd
```

Funtion load_dataset3()

```
def load_dataset3():
    P = np.genfromtxt(C:/Users/carol/OneDrive/Ambiente
        de Trabalho/Ensaios_Finais/Final_CSV.csv,
        delimiter=',',
        missing_values=None, filling_values=0.0)

    d = P.shape[0]

    P = np.append(P,
    np.zeros((d, 256 - P.shape[1] + 2)), axis=1)
    np.random.shuffle(P)
```

```
12
13      V = P[:,1:2]
14      P = P[:,2:]
15
16      P = P.reshape((P.shape[0], P.shape[1]))
17
18      scaler = StandardScaler()
19      P = scaler.fit_transform(P)
20      scaled_P = pd.DataFrame(data=P)
21
22      new_P = scaled_P.to_numpy()
23
24      new_P = new_P.reshape((new_P.shape[0],
25      new_P.shape[1], 1))
26
27      V = V.reshape(V.shape[0])
28      V = tf.keras.utils.to_categorical(V)
29
30      divide = round(d*0.80)
31
32      trainX = new_P[:divide]
33      testX = new_P[divide:]
34      trainY = V[:divide]
35      testY = V[divide:]
36
37      return trainX, testX, trainY, testY
```

A.6.3 Normalization

Libraries

```
1  import csv
2  import numpy as np
3  from numpy import mean
4  from numpy import std
5
6  import tensorflow as tf
7  from tensorflow.keras.models import Sequential
8  from tensorflow.keras.layers import Dense
9  from tensorflow.keras.layers import Flatten
10 from tensorflow.keras.layers import Dropout
11 from tensorflow.keras.layers import LSTM
12 from sklearn.preprocessing import  MinMaxScaler
13 import pandas as pd
14 import matplotlib.pyplot as plt
```

Function load_dataset3()

```
def load_dataset3():
    P = np.genfromtxt(C:/Users/carol/OneDrive/Ambiente
        de Trabalho/Ensaios_Finais/Final_CSV.csv,
        delimiter=',',
        missing_values=None, filling_values=0.0)

    d = P.shape[0]

    P = np.append(P,
    np.zeros((d, 256 - P.shape[1] + 2)), axis=1)
    np.random.shuffle(P)

    V = P[:,1:2]
    P = P[:,2:]

    P = P.reshape((P.shape[0], P.shape[1]))
    scaler = MinMaxScaler()
    P = scaler.fit_transform(P)

    scaled_P = pd.DataFrame(data=P)

    new_P = scaled_P.to_numpy()

    new_P = new_P.reshape((new_P.shape[0],
    new_P.shape[1], 1))

    V = V.reshape(V.shape[0])
    V = tf.keras.utils.to_categorical(V)

    divide = round(d*0.80)

    trainX = new_P[:divide]
    testX = new_P[divide:]
    trainY = V[:divide]
    testY = V[divide:]

    return trainX, testX, trainY, testY
```

A.7 CNN Model

A.7.1 Without standardization or normalization

Libraries

```
import csv
import numpy as np
from numpy import mean
from numpy import std

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import TimeDistributed
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import MaxPooling1D
```

Function load_dataset3()

```
def load_dataset3():
    P = np.genfromtxt(C:/Users/carol/OneDrive/Ambiente
        de Trabalho/Ensaios_Finais/Final_CSV.csv,
        delimiter=',',
        missing_values=None, filling_values=0.0)

    d = P.shape[0]

    P = np.append(P,
    np.zeros((d, 256 - P.shape[1] + 2)), axis=1)
    np.random.shuffle(P)

    V = P[:,1:2]
    P = P[:,2:]

    P = P.reshape((P.shape[0], P.shape[1], 1))

    V = V.reshape(V.shape[0])
    V = tf.keras.utils.to_categorical(V)

    divide = round(d*0.80)

```

```
23      trainX = P[:divide]
24      testX = P[divide:]
25      trainY = V[:divide]
26      testY = V[divide:]
27
28      return trainX, testX, trainY, testY
```

Funtion evaluate_model()

```
1  def evaluate_model (trainX, trainY, testX, testY):
2      verbose, epochs, batch_size=0,25,64
3      n_steps = 4
4      n_length = 64
5      n_timesteps = trainX.shape[1]
6      n_features = trainX.shape[2]
7      n_outputs = trainY.shape[1]
8
9      trainX = trainX.reshape((trainX.shape[0], n_steps,
10     n_length, n_features))
11     testX = testX.reshape((testX.shape[0], n_steps,
12     n_length, n_features))
13
14     model = Sequential()
15     model.add(TimeDistributed(Conv1D(filters=64,
16     kernel_size=3, activation='relu'),
17     input_shape=(None,n_length,n_features)))
18     model.add(TimeDistributed(Conv1D(filters=64,
19     kernel_size=3, activation='relu')))
20     model.add(TimeDistributed(Dropout(0.5)))
21     model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
22     model.add(TimeDistributed(Flatten()))
23     model.add(LSTM(100))
24     model.add(Dropout(0.5))
25     model.add(Dense(100, activation='relu'))
26     model.add(Dense(n_outputs, activation = 'softmax'))
27     model.compile(loss='categorical_crossentropy',
28     optimizer='adam', metrics=['accuracy'])
29
30     model.fit(trainX, trainY, epochs=epochs,
31     batch_size=batch_size, verbose=verbose)
32
33     _, accuracy = model.evaluate(testX, testY,
34     batch_size = batch_size, verbose = 1)
35
36     return accuracy
```

Function summarize_results()

```
def summarize_results(scores):
    print(scores)
    m, s = mean(scores), std(scores)
    print('Accuracy:%.3f%%(+/-%.3f)' % (m, s))
```

Function run_experiment()

```
# run an experiment
def run_experiment(repeats=5):
  # load data
  trainX, testX, trainY, testY = load_dataset3()
  # repeat experiment
  scores = list()
  for r in range(repeats):
    score = evaluate_model(trainX,
    trainY, testX, testY)
    score = score * 100.0
    print('>#%d:%.3f' % (r+1, score))
    scores.append(score)
  # summarize results
  summarize_results(scores)
```

A.7.2 Standardization

Libraries

```
import csv
import numpy as np
from numpy import mean
from numpy import std

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import TimeDistributed
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import MaxPooling1D

```

```
from sklearn.preprocessing import  StandardScaler
import pandas as pd
import matplotlib.pyplot as plt
```

Function load_dataset3()

```
def load_dataset3():
    P = np.genfromtxt(C:/Users/carol/OneDrive/Ambiente
        de Trabalho/Ensaios_Finais/Final_CSV.csv,
        delimiter=',',
        missing_values=None, filling_values=0.0)

    d = P.shape[0]

    P = np.append(P,
    np.zeros((d, 256 - P.shape[1] + 2)), axis=1)
    np.random.shuffle(P)

    V = P[:,1:2]
    P = P[:,2:]

    P = P.reshape((P.shape[0], P.shape[1]))

    scaler = StandardScaler()
    P = scaler.fit_transform(P)
    scaled_P = pd.DataFrame(data=P)

    new_P = scaled_P.to_numpy()
    new_P = new_P.reshape((new_P.shape[0],
    new_P.shape[1], 1))

    V = V.reshape(V.shape[0])
    V = tf.keras.utils.to_categorical(V)

    divide = round(d*0.80)

    trainX = new_P[:divide]
    testX = new_P[divide:]
    trainY = V[:divide]
    testY = V[divide:]

    return trainX, testX, trainY, testY
```

A.7.3 Normalization

Libraries

```
import csv
import numpy as np
from numpy import mean
from numpy import std

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import TimeDistributed
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import MaxPooling1D

from sklearn.preprocessing import  MinMaxScaler
import pandas as pd
import matplotlib.pyplot as plt
```

Function load_dataset3()

```
def load_dataset3():
    P = np.genfromtxt(C:/Users/carol/OneDrive/Ambiente
        de Trabalho/Ensaios_Finais/Final_CSV.csv,
        delimiter=',',
        missing_values=None, filling_values=0.0)

    d = P.shape[0]

    P = np.append(P,
    np.zeros((d, 256 - P.shape[1] + 2)), axis=1)
    np.random.shuffle(P)

    V = P[:,1:2]
    P = P[:,2:]

    P = P.reshape((P.shape[0], P.shape[1]))
    scaler = MinMaxScaler()
    P = scaler.fit_transform(P)

    scaled_P = pd.DataFrame(data=P)

```

```
22      new_P = scaled_P.to_numpy()
23
24      new_P = new_P.reshape((new_P.shape[0],
25      new_P.shape[1], 1))
26
27      V = V.reshape(V.shape[0])
28      V = tf.keras.utils.to_categorical(V)
29
30      divide = round(d*0.80)
31
32      trainX = new_P[:divide]
33      testX = new_P[divide:]
34      trainY = V[:divide]
35      testY = V[divide:]
36
37      return trainX, testX, trainY, testY
```

Function summarize_results()

```
1  def summarize_results(scores):
2    print(scores)
3    m, s = np.mean(scores), np.std(scores)
4    print('Accuracy:%.3f%% (+/-%.3f)' % (m, s))
```

Function run_experiment()

```
1  # run an experiment
2  def run_experiment(repeats=5):
3    # load data
4    trainX, testX, trainY, testY = load_dataset3()
5    # repeat experiment
6    scores = list()
7    for r in range(repeats):
8      score = evaluate_model(trainX,
9      trainY, testX, testY)
10     score = score * 100.0
11     print('>#%d:%.3f' % (r+1, score))
12     scores.append(score)
13   # summarize results
14   summarize_results(scores)
```

A.8 Convolutional LSTM Model

A.8.1 Without standardization or normalization

Libraries

```
import csv
import numpy as np
from numpy import mean
from numpy import std

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import ConvLSTM2D
```

Function load_dataset3()

```
def load_dataset3():
    P = np.genfromtxt(C:/Users/carol/OneDrive/Ambiente
        de Trabalho/Ensaios_Finais/Final_CSV.csv,
        delimiter=',',
        missing_values=None, filling_values=0.0)

    d = P.shape[0]

    P = np.append(P,
    np.zeros((d, 256 - P.shape[1] + 2)), axis=1)
    np.random.shuffle(P)

    V = P[:,1:2]
    P = P[:,2:]

    P = P.reshape((P.shape[0], P.shape[1], 1))

    V = V.reshape(V.shape[0])
    V = tf.keras.utils.to_categorical(V)

    divide = round(d*0.80)

    trainX = P[:divide]
    testX = P[divide:]
```

```
25      trainY = V[:divide]
26      testY = V[divide:]
27
28      return trainX, testX, trainY, testY
```

Function evaluate_model()

```
1  def evaluate_model(trainX, trainY, testX, testY):
2    verbose, epochs, batch_size = 0, 25, 64
3
4    n_timesteps = trainX.shape[1]
5    n_features = trainX.shape[2]
6          n_outputs = trainY.shape[1]
7
8    n_steps, n_length = 4, 64
9    trainX = trainX.reshape((trainX.shape[0], n_steps,
10   1, n_length, n_features))
11   testX = testX.reshape((testX.shape[0], n_steps,
12   1, n_length, n_features))
13   modelo = Sequential()
14   modelo.add(ConvLSTM2D(filters=64,
15   kernel_size=(1,3), activation='relu',
16   input_shape=(n_steps, 1, n_length, n_features)))
17   modelo.add(Dropout(0.55))
18   modelo.add(Flatten())
19   modelo.add(Dropout(0.5))
20   modelo.add(Dense(100, activation='relu'))
21   modelo.add(Dense(n_outputs, activation='softmax'))
22   modelo.compile(loss='categorical_crossentropy',
23   optimizer='adam', metrics=['accuracy'])
24   modelo.fit(trainX, trainY, epochs=epochs,
25   batch_size=batch_size, verbose=verbose)
26   _, accuracy = modelo.evaluate(testX, testY,
27   batch_size=batch_size, verbose=1)
28
29   return accuracy
```

Function summarize_results()

```
1  def summarize_results(scores):
2      print(scores)
3      m, s = mean(scores), std(scores)
4      print('Accuracy:%.3f%%(+/-%.3f)' % (m, s))
```

Function run_experiment()

```
def run_experiment(repeats=5):
  # load data
  trainX, testX, trainY, testY = load_dataset3()
  # repeat experiment
  scores = list()
  for r in range(repeats):
    score = evaluate_model(trainX, trainY,
    testX, testY)
    score = score * 100.0
    print('>#%d:%.3f' % (r+1, score))
    scores.append(score)
  # summarize results
  summarize_results(scores)
```

A.8.2 Standardization

Libraries

```
import csv
import numpy as np
from numpy import mean
from numpy import std

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import ConvLSTM2D

from sklearn.preprocessing import StandardScaler
import pandas as pd
```

Function load_dataset3()

```
def load_dataset3():
    P = np.genfromtxt(C:/Users/carol/OneDrive/Ambiente
        de Trabalho/Ensaios_Finais/Final_CSV.csv,
        delimiter=',',
        missing_values=None, filling_values=0.0)

```

```
    d = P.shape[0]

    P = np.append(P,
    np.zeros((d, 256 - P.shape[1] + 2)), axis=1)
    np.random.shuffle(P)

    V = P[:,1:2]
    P = P[:,2:]

    P = P.reshape((P.shape[0], P.shape[1]))

    scaler = StandardScaler()
    P = scaler.fit_transform(P)
    scaled_P = pd.DataFrame(data=P)

    new_P = scaled_P.to_numpy()

    new_P = new_P.reshape((new_P.shape[0],
    new_P.shape[1], 1))

    V = V.reshape(V.shape[0])
    V = tf.keras.utils.to_categorical(V)

    divide = round(d*0.80)

    trainX = new_P[:divide]
    testX = new_P[divide:]
    trainY = V[:divide]
    testY = V[divide:]

    return trainX, testX, trainY, testY
```

A.8.3 Normalization

Libraries

```
import csv
import numpy as np
from numpy import mean
from numpy import std

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LSTM
```

```
from tensorflow.keras.layers import TimeDistributed
from tensorflow.keras.layers import ConvLSTM2D

from sklearn.preprocessing import  MinMaxScaler
import pandas as pd
import matplotlib.pyplot as plt
```

Function load_dataset3()

```
def load_dataset3():
    P = np.genfromtxt(C:/Users/carol/OneDrive/Ambiente
        de Trabalho/Ensaios_Finais/Final_CSV.csv,
        delimiter=',',
        missing_values=None, filling_values=0.0)

    d = P.shape[0]

    P = np.append(P,
    np.zeros((d, 256 - P.shape[1] + 2)), axis=1)
    np.random.shuffle(P)

    V = P[:,1:2]
    P = P[:,2:]

    P = P.reshape((P.shape[0], P.shape[1]))
    scaler = MinMaxScaler()
    P = scaler.fit_transform(P)

    scaled_P = pd.DataFrame(data=P)

    new_P = scaled_P.to_numpy()

    new_P = new_P.reshape((new_P.shape[0],
    new_P.shape[1], 1))

    V = V.reshape(V.shape[0])
    V = tf.keras.utils.to_categorical(V)

    divide = round(d*0.80)

    trainX = new_P[:divide]
    testX = new_P[divide:]
    trainY = V[:divide]
    testY = V[divide:]

    return trainX, testX, trainY, testY
```

Appendix B

Protocol for testing with monitoring system

B.1 Introduction

This experimental protocol was designed to standardize the procedures regarding the measurement of forces/pressures exerted on the plantar region of the foot. Additionally a record of the data coming from the acquisition system will be carried out for later use in an AI model for classifying the type of human activity. In Figure B.1, the Ortomedical insole is represented with the Nanopaint insole integrated and in Figure B.2 the final shoe and insole set used in the tests to be carried out is represented.

Figure B.1: Nanopaint insole integrated under the Ortomedical insole

Figure B.2: End insole integrated into the Ortomedical shoe

Initially, the registration of information relating to each participant must be carried out. Then the four tests defined in chapter B.3 are carried out for each participant.

B.2 Registration of data related to the participant

Before carrying out the tests, fill in the Excel file "MetaInfo.xlsx" with the following data about each participant:

- Name
- Age
- Sex
- Weight
- Height
- Velocity in normal gait
- Velocity in slow speed
- Velocity in fast gait
- Comments (Optional)

Procedures before starting the tests:

1. To determine each user's reference speeds (normal, slow and fast), perform a trial where the speed is increased until the participant detect that he is at your normal/slow and fast speed;
2. To familiarize the participant with the treadmill, perform a 6 minutes test before the defined tests, at a speed chosen by them;

B.3 Procedure with the Nanopaint insole integrated into the insole of the Ortomedical

B.3.1 Measurement at rest

1. Put on the Ortomedical shoe;
2. Record the values measured by the sensors for approximately 40 seconds;
3. Turn on the system and run the python script to save the data in a .txt file.

B.3.2 Measurement at normal gait speed

1. Put on the Ortomedical shoe;
2. Position the participant on the treadmill;
3. Program the treadmill for the participant's normal speed;
4. Turn on the system and run the python script to save the data in a .txt file for 2 minutes;
5. At the end of the 2 minutes, stop the data recording and decrease the speed of the treadmill gradually.

B.3.3 Measurement at slow gait speed

1. Put on the Ortomedical shoe;
2. Position the participant on the treadmill;
3. Program the treadmill for the participant's slow speed;
4. Turn on the system and run the python script to save the data in a .txt file for 2 minutes;
5. At the end of the 2 minutes, stop the data recording and decrease the speed of the treadmill gradually.

B.3.4 Measurement at fast gait speed

1. Put on the Ortomedical shoe;
2. Position the participant on the treadmill;
3. Program the treadmill for the participant's fast speed;
4. Turn on the system and run the python script to save the data in a .txt file for 2 minutes;
5. At the end of the 2 minutes, stop the data recording and decrease the speed of the treadmill gradually.