Detection of sleep position by means of a wrist-worn sensor technology

Ricardo Marques Trindade

Thesis to obtain the Master of Science Degree in

Biomedical Engineering

Supervisor(s):  Prof. Miguel Pedro Tavares da Silva
               Dr. Carla Cristina Paulo Gabriel Bentes

Examination Committee
Chairperson:  Prof. João Pedro Estrela Rodrigues Conde
Supervisor:  Prof. Miguel Pedro Tavares da Silva
Member of the Committee:  Prof. João Miguel Raposo Sanches
Member of the Committee:  Prof. Maria Teresa Aguiar Santos Paiva

November 2016
Acknowledgments

The author would like to thank:

To Duarte Mendes de Almeida for giving me the opportunity to develop this work, his constant help, suggestions, and for all he taught me that proved essential to this work.

To Professor Miguel Tavares da Silva for his constant motivation and readiness to discuss and criticize my ideas.

To Professor Carla Bentes, for her help in the medical component of this work and availability in conducting the study in Hospital de Santa Maria.

To my parents, grandparents and brother, without them, none of this journey would have been possible.

To my godmother and my aunt Mécia for all they’ve done that somehow led me to where I am today.

To Joana Faria and Beatriz Almeida for putting up with me during these 5 years.

To Jorge Martins, Nuno Matias, and Ricardo Marques for countless hours of good disposition which made the journey much easier.

To the neurology department technicians Rosa and Lígia for always keeping me up to date with the issues with the devices being used.

To Hole19, for what I’ve learned from them, really improved the code quality in this work.

To Susana, whose help and support goes beyond words.
Resumo

A posição do sono é um parâmetro clinicamente relevante, está diretamente relacionado com a qualidade do mesmo e associado a patologias como a apneia do sono. O objectivo deste trabalho é responder à questão: É possível determinar a posição do sono, recorrendo apenas a um acelerómetro de três eixos incorporado num smartwatch? Para tal construímos uma base de dados de movimentos do sono. Para a aquisição dos dados desenvolvemos uma aplicação Android que permite a leitura dos valores do acelerómetro do smartwatch. Ao mesmo tempo pacientes do Hospital de Santa Maria também usaram um acelerômetro no punho enquanto dormiam. Os dados obtidos destas noites foram usados para analisar o desempenho do algoritmo. O algoritmo desenvolvido usa classificadores como Redes Neuronais, Máquina de Suporte Vectorial, Modelos de Markov e classificadores de Bayes. A tarefa principal do algoritmo é determinar qual o movimento executado e a partir daí inferir a posição. Definindo que apenas 3 posições possíveis (Decúbito Dorsal, Decúbito Lateral e Decúbito Ventral) a precisão média do algoritmo foi 53.2±26.9% para redes neuronais, 51±20.9% para máquinas de suporte vectorial, 44.5±15.9% para o classificador de Bayes e 35.6±21.1% para os modelos de Markov. Concluímos que abrimos caminho para o desenvolvimento de um algoritmo que classifica a posição do sono usando apenas um acelerômetro no pulso, embora ainda haja muito espaço para melhorias.

Palavras-chave: posição do sono, apneia do sono, redes neuronais, máquina de suporte vectorial
Abstract

Sleep position is a clinically relevant parameter, as it affects the quality of sleep and is associated with several pathologies such as sleep apnea. The goal of this work is to answer the question: Is it possible to develop an algorithm that classifies the sleeping position, solely based on a three-axis accelerometer embedded in a smartwatch? For this we built a sleep movement database. To acquire sleep movements we developed an Android application that allowed to record the accelerometer values from a smartwatch. A wrist accelerometer was worn by patients from Hospital de Santa Maria, and volunteers while they slept. The data obtained from these nights was used to assess the algorithm’s performance. The developed algorithm uses classifiers such as Neural Networks, Support Vector Machines, and Naive Bayes classifiers. The core task is to identify the sleep movement performed and from there infer the position. Defining that there can only be 3 possible sleep positions (supine, lateral and prone) the average algorithm accuracy was 53.2±26.9% for neural networks, 51±20.9% for support vector machines, 44.5±15.9% for the Naive Bayes classifier and 35.6±21.1% with Hidden Markov models. We conclude that we have paved the way for the development of an algorithm that classifies sleep position using only an accelerometer placed on the wrist, as there is a lot of room for improvement.

Keywords: sleep position, sleep apnea, neural networks, accelerometer, support vector machine,
## Contents

Acknowledgments ................................................................. iii  
Resumo ................................................................. v  
Abstract ................................................................. vii  
List of Tables ............................................................... xiii  
List of Figures ............................................................... xv  
Glossary ................................................................. xvii  

1 Introduction ................................................................. 1  
1.1 Motivation and Objectives ................................................. 1  
1.2 State of the art ............................................................... 2  
1.3 Approach and Organization ............................................... 5  

2 Sleep Pathologies and Investigation Technologies .................. 7  
2.1 Sleep Apnea ................................................................. 7  
2.1.1 Anatomy ................................................................. 8  
2.1.2 Respiration and Sleep ................................................ 8  
2.1.3 Age ................................................................. 9  
2.1.4 Gender ................................................................. 9  
2.1.5 Posture and gravity .................................................. 10  
2.1.6 Anatomical Factors ................................................ 10  
2.1.7 Body fat ............................................................... 10  
2.1.8 Genetic Factors ............................................................... 11  
2.2 Restless Leg Syndrome .................................................. 12  
2.2.1 Genetic Factors ............................................................... 12  
2.2.2 Iron Deficiency ............................................................... 12  
2.3 MEMS Devices ............................................................... 12  
2.3.1 Accelerometer ............................................................... 12  
2.3.2 Gyroscopes ............................................................... 14  
2.3.3 Magnetometer ............................................................... 17  
2.3.4 Rotation Vector ............................................................... 18
List of Tables

4.1 Number of examples for each movement, of a total of 460 different examples. . . . . . . . 34
4.2 Number of examples for each movement considering the position on which they ended, using
3 classes, of a total of 460 different examples. . . . . . . . . . . . . . . . . . . . . . . . . . 34
4.3 Number of examples for each movement, considering the position on which they ended, using
2 classes, of a total of 460 different examples. . . . . . . . . . . . . . . . . . . . . . . . . . 34

6.1 8 Classes, with a simple input-hidden-output neural network structure similar to figure 5.4.
The number of units correspond to the hidden layer. Using the same number of examples
for all classes $n=41$. Without gyroscope features . . . . . . . . . . . . . . . . . . . . . . . . . 56
6.2 8 Classes, with a simple input-hidden-output neural network structure similar to figure 5.4.
The number of units correspond to the hidden layer. Using the same number of examples
for all classes $n=41$. Using gyroscope features. $p = 0.1354$ when compared to the results
in table 6.1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
6.3 3 Classes (Supine, Lateral, Prone), with a simple input-hidden-output neural network
structure similar to figure 5.4. The number of units correspond to the hidden layer and
without gyroscope features, using the same number of examples on all classes $n=89$ . . 57
6.4 3 Classes (Supine, Lateral, Prone), with a simple input-hidden-output neural network
structure similar to figure 5.4. The number of units correspond to the hidden layer with
gyroscope features, using the same number of examples on all classes $n=89$. $p = 0.2718$
when compared to the results in table 6.3 . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
6.5 2 Classes (Supine, NonSupine), with a simple input-hidden-output neural network struc-
ture similar to figure 5.4. The number of units correspond to the hidden layer without
gyroscope features, trained with the same number of examples in both classes $n=139$ . 58
6.6 2 Classes (Supine, NonSupine), with a simple input-hidden-output neural network struc-
ture similar to figure 5.4. The number of units correspond to the hidden layer with gy-
roscope features, trained with the same number of examples in both classes $n=139$.
$p = 0.9120$ when compared to the results in table 6.5 . . . . . . . . . . . . . . . . . . . . . . . . . 58
6.7 SVM with a Gaussian Kernel, Naive Bayes classifier with a Normal Distribution . . . . . 58
6.8 ANN with a simple input-hidden-output structure, 5 units, 8 classes as described in chap-
ter 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
6.9 ANN with a simple input-hidden-output structure, 5 units, 3 classes(Supine,Lateral,Prone) 61
6.10 ANN with a simple input-hidden-output structure, 5 units, 2 classes (Supine vs Not Supine) 62
6.11 Markov Model results ........................................................................................................ 62

A.1 Results from the procedure described chapter 5.4.5, using 8 classes, with a simple input-
hidden-output architecture, similar to figure 5.4 .................................................................. 79
A.2 Results from the procedure described chapter 5.4.5, using 3 classes, with a simple input-
hidden-output architecture, similar to figure 5.4 .................................................................. 80
A.3 Results from the procedure described chapter 5.4.5, using 2 classes, with a simple input-
hidden-output architecture, similar to figure 5.4 .................................................................. 80
# List of Figures

1.1 A common piezoelectric sensor used to measure sleeping position taken from [6].  

1.2 Flowchart for estimating sleep posture epoch [7].  

1.3 Experimental setup taken from [1].  

1.4 Accelerometer Reading variation for different lying positions taken from [1].  

1.5 Typical 6 sleep postures as defined in [9].  

1.6 Joint model detected by the Kinect as in [9].  

2.1 Different grades of the Mallampati score taken from [23].  

2.2 Diagram of the comb structure present in MEMS devices taken from [31].  

2.3 Illustration of the Hall effect principle, taken from [35].  

2.4 Example output from the rotation vector on a movement going from a lateral to supine position, from top to bottom we have the x, y and z components of the rotation vector over time.  

3.1 A simplified illustration of the Activity lifecycle, expressed as a step pyramid. This shows how, for every callback used to take the activity a step toward the Resumed state at the top, there’s a callback method that takes the activity a step down. The activity can also return to the resumed state from the Paused and Stopped state.  

3.2 Application flow.  

3.3 Doze provides a recurring maintenance window for apps to use the network and handle pending activities, taken from [39].  

3.4 Android’sCoordinate System taken from [36].  

4.1 Body positions while lying on bed adapted from [1]. From left to right, we have left lateral position, right lateral position, prone position and supine position.  

4.2 Empatica E4 wristband (left) and the Sony Smartwatch 3 (right).  

4.3 Application used for the Kinect recordings.  

4.4 Application used by the Empatica wristband for recording data.  

4.5 Application used for the recording of sleep movements.
5.1 Obtained p-values from pairwise comparison between features, using 3 classes. On the vertical axis we have each feature in the same order as they are present in section 5.3. The numbers on the list of features are the direct correspondence to each row on the image. First column is the comparison between supine and prone classes, second column is supine and lateral classes, and third column is the comparison between lateral and prone. The scale on the right represents p-value.

5.2 Acceleration signal on the z-axis on a change from supine to right lateral position. Vertical axis is in $m.s^{-2}$. 

5.3 Multi layer perceptron, taken from [42].

5.4 Neural network architecture similar to the one used in this work, where we altered the number of hidden units and the size of input and output vectors. Taken from [43].

5.5 Example of a multilayer perceptron and of the corresponding backpropagation network.
   a) Multilayer perceptron. b) Backpropagation network. Taken from [42].

5.6 Screenshot of the neural network toolbox training process [44].

5.7 Plot of each network's performance over 100 iterations, using 3 classes. Vertical axis represents the average accuracy.

5.8 Hyperplane through two linearly separable classes from [56].

5.9 Dichotomous data re-maped used Radial Basis Kernel, taken from [56].

5.10 Illustration of the model's training process.

5.11 Markov model classification process.

6.1 Example of the algorithm output(blue) vs groundtruth(red). Using neural networks with three classes.

6.2 Output of the transition detector, spikes represent the places where a transition was detected, the red line represents the position vector from the PSG.

6.3 Output of the algorithm, using neural networks with no transition detector, with three classes.

6.4 Confusion matrix resulting from training the network with 8 classes. Classes 1 to 8 are prone-right lateral, prone-left lateral, supine-right lateral, supine-left lateral, left lateral-prone, right lateral-prone, left lateral-supine, right lateral-supine respectively.

6.5 Confusion matrix resulting from training the network with 3 classes. Classes 1 to 3 are movements ending in supine, lateral, and prone position respectively.
Abbreviations

**ANN** Artificial Neural Network. 35, 38, 64

**API** Application Programming Interface. 25

**APK** Android Application Package. 23

**BMI** Body-Mass Index. 11

**COP** Center of Pressure. 2, 3

**CSV** Comma-separated values. 30, 33

**ECG** Electrocardiogram. 5

**GMI** Giant Magneto-Impedance. 17

**HMM** Hidden Markov Model. 49, 50, 52

**IDE** Integrated Developer Environment. 23

**JVM** Java Virtual Machine. 23

**MEMS** Microelectromechanical systems. 13, 14

**MLP** Multilayer Perceptron. 38

**OSA** Obstructive Sleep Apnea. 7–11

**PLM** Periodic Limb Movements. 12

**PPG** Photoplestimogram. 1, 31, 68, 69

**PSG** Polysomnography. 1, 2, 4, 29, 31, 58, 64, 65

**RDI** Respiratory-Disturbance Index. 7

**REM** Rapid-Eye Movement. 9
RERA  Respiratory Effort Related Arousals. 8
RLS  Restless Leg Syndrome. 12
SVM  Support vector machine. 35, 43, 47, 48, 55, 59, 60, 69
WISP  Wireless Identification Sensing Platforms. 3
XML  Extendend Markup Language. 23
Chapter 1

Introduction

1.1 Motivation and Objectives

The world of smartphones and their applications has already taken hold of our lives, and more recently the arrival of Smartwatches has allowed the development of a wide variety of health and well-being related applications. Some of these applications are fitness related as they allow to monitor physical activity by making use of the embedded watch’s sensors. Take a look at Google Fit, an application by Google that uses the accelerometer and GPS of the smartphone to monitor the distance and pace of running or walking sessions.

Regarding sleep, the position during the night can be a clinically relevant parameter as it is associated with several sleep disturbances, even further, analyzing sleep movements helps to determine the quality of sleep and sleep patterns [1]. Some of these pathologies are the Gastroesophageal Reflux Disease where it has already been demonstrated that an optimal sleep position for individuals with this disturbance is by lying on their left side [2]. Another disturbance is sleep apnea which is associated with several cardiovascular diseases affecting mostly middle-aged men and has been shown to be aggravated by certain sleep positions [3]. Monitoring sleep position is also necessary to prevent immobile patients and the elderly from pressure ulcers. It is necessary to change patients’ position regularly to prevent sores because it is less harmful and expensive treating them at an early stage than when they have progressed far enough [4, 5].

There are already smartphone applications on the market as well as devices that monitor sleeping patterns like the "Sleep Cycle Alarm Clock" by Northcube AB for iPhone.

Up until now, no application or device on the wrist can detect the sleeping position. By having an application or device that could measure this it would make the life of the patient easier, and keeping up with the advances in technology with the incorporation of sensors like the photoplestimogram (PPG), accelerometer, and gyroscope in a smartwatch would reduce the burden of cables that patients are subjected to when doing a polysomnography (PSG).

The main goal of this work is to answer the question: Is it possible to develop an algorithm that classifies the sleeping position, solely based on a three-axis accelerometer embedded in a smartwatch?
To do so, a smartwatch application will be developed to extract data from the watch's sensors. We will also build a sleep movement database with the data retrieved from the sensors.

During the realization of the work, a study will be running at Hospital de Santa Maria with patients with sleeping disorders being subjected to PSGs. These patients will also wear a device during the procedure with the purpose of retrieving the data from it. This data will be used for assessing the algorithm’s performance.

1.2 State of the art

For the classification of the sleeping position several approaches have already been made, some of them are presented here. For patients doing a PSG the most common way to measure their sleeping position is by placing a piezoelectric sensor in the area of the chest (see figure 1.1).

This sensor, however, can be uncomfortable to wear, specially if the patient tries to sleep on its stomach. A study using this type of sensors by Yoon et al. [7], small patch containing an accelerometer was attached to the left side of the chest. In this paper, after analyzing the values outputted from the device, 5 postures were defined, supine, prone, left and right lateral sleep, and unknown postures such as sitting and standing, which were not classified as sleep postures. As for the data processing, a simple low-pass 0.1 Hz filter was applied to the 3 accelerometer signals. The data were then averaged on 30s epochs, resulting in a 3-dimensional vector $F_{x,y,z}$. Their algorithm would classify the position according to predefined thresholds (see Figure 1.2).

A non-invasive approach to measure sleeping position was made by Zachary Beatle et al. in [8], load cells were placed under the patient’s bed. Each load cell was placed under each bed support, giving a total of 4 cells. The experiment conducted had patients lying in bed during 32 minutes where they were instructed to lie in 4 different positions. Any movement of legs or arms were left to the patient’s own will. The signal retrieved from the load cells allowed to compute the bed’s center of pressure (COP) on the $x$ and $y$ directions. For the classification of the sleeping position, a K-Means classifier was implemented.
with 4 centroids, using the COP in the $y$ direction as a feature. An accuracy of 0.68, 0.57, 0.69 and 0.33 was obtained for the back, right, left and stomach positions respectively [8]. The biggest problem with this approach is that the $COP_y$ values for the stomach and back position are similar and the classifier fails to distinguish between them sometimes.

Another non-invasive approach to measure sleep position, and similar to the one by [8] was made by Hoque et al in [1]. Their work used devices called Wireless Identification Sensing Platforms (WISPs), with embedded accelerometers and are capable of transmitting information wirelessly to a computer. These devices were placed under a mattress as seen in figure 1.3. Similar to our work, in [1] were also defined 4 possible sleeping positions (Left Lateral, Right Lateral, Prone, and Supine). For their experiment, they had 10 subjects sleeping on these mattresses. The values recorded from the accelerometers were then analyzed, which allowed distinguishing perfectly between the four defined positions. Figure 1.4 shows the accelerometer data along the $y$-axis taken from one of the sensors, where it is clearly noticeable the four different readings for each position.

Jaehoon et al in [9] developed a system that used a Kinect sensor to monitor the sleeping position. In this paper, they took advantage of the Kinect infrared camera, and the Kinect's own human body joints algorithm that detects human body movement and builds a model based on that information. The Kinect was placed above the bed and would stream data to a nearby computer which would later process the data. The joint model outputted by the Kinect had 25 points, and they recorded the $x$, $y$ and $z$ position of all those points. The sleep movement was calculated based on the Euclidean distance between
those points through time. This data was also compared with 6 sleeping positions that are illustrated in figure 1.5.

Another approach was taken by Yao et al. [10], where 3-axis accelerometers were used to monitor the sleeping position and head position. They developed a low-cost low-power system that can be integrated with PSGs to perform real-time diagnosis with an accurate body and head position. They used two accelerometers, one which placed on the patient’s forehead for head positioning, and the other attached to the chest for body positioning. They claim that transient spontaneous remissions occur during the diagnostic of obstructive sleep apnea, could be due to changes in body or head position. As the existing PSG systems are unable to measure head position with high accuracy, this system was developed. Their system computed two angles, an angle $\alpha$ between the gravity vector and the $yz$ plane, and angle $\beta$ between the projection of the gravity vector on the $yz$ plane and the $z$-axis. They were able to obtain an accuracy of $\pm 0.466$ degree which is high, considering the broad head movements during sleep.

Shinar et al. [11] introduced the algorithm which detects sleep posture characterized by morpholog-
ical differences in QRS complex of the Lead I, II, and III electrocardiogram ECG. The limitation of this study is that this method is applicable only when the ECG is clearly measured and the QRS complex is accurately detected.

Despite not being available an algorithm for detecting the sleep position, using a wrist accelerometer, these sensors are being used for sleep detection. In [12] Borazio et al. developed an algorithm that was able to classify whether the patient was sleeping or not based on the data of a single wrist-worn accelerometer. They analyzed the signal with a fixed size sliding window. Their method was based on thresholding the standard deviation of the accelerometer signal based on the following rule

$$S_{\delta} = \begin{cases} 1, & \text{if } \sqrt{\frac{1}{99} \sum_{i=1}^{100} (z_i - \bar{z})^2} > \delta \\ 0, & \text{otherwise} \end{cases}$$

(1.1)

The value of $\delta$ would vary as well the length of the window. Their findings outperformed previous methods like the one by [13] which used an actigraph and was also threshold based.

### 1.3 Approach and Organization

This work is divided into 7 chapters. 1. Introduction, 2. Sleep pathologies and investigation technologies, 3. Android Application Development, 4. Experimental Acquisition of Sleep Movements, 5. Implemented Classifiers and Methodology, 6. Results and Discussion, 7. Final Remarks. In chapter 1 we present a brief overview of what has been done so far to detect the sleeping position, while in chapter 2 we expose sleep apnea as it is the pathology on which sleep position has a great impact and finally we cover the working principle of the sensors present on the devices used. In chapter 3 we take a look at
the whole process of developing an Android application for the smartphone and smartwatch, that allows accessing the watch’s sensors. Chapter 4 focuses the data acquisition protocols for each of the different setups as the approach taken was to build a sleep movement database. The acquisition was performed in three different ways. Recording the sleep of volunteers with a Kinect V2 video camera along with the Smartwatch, and the polysomnographies (PSGs) in sleep disorder patients realized at Hospital de Santa Maria and recording different sleep movements on awake volunteers. Chapter 5 goes into the theoretical details for each classifier used in this work. Chapter 6 covers the data analysis and performance of the developed algorithm. Finally, Chapter 7 draws conclusions and the future research steps in this field of study are explored.
Chapter 2

Sleep Pathologies and Investigation Technologies

This chapter includes the literature review on sleep apnea, periodic limb movements, and restless leg syndrome. Then we also include the functioning process of the sensors present on the devices used, and the derivations of the equations that rule these sensors.

2.1 Sleep Apnea

Individuals with Obstructive Sleep Apnea (OSA) present recurrent episodes of partial or complete upper airway obstruction during sleep. Despite the inspiratory effort, airflow is decreased (hypopnea) or completely interrupted (apnea). The lack of adequate alveolar ventilation generally results in oxyhemoglobin desaturation and, in cases of prolonged events, in a progressive increase of arterial carbon dioxide tension ($PaCO_2$). Despite this fact, patients with OSA might not notice anything at all during their nights. During the day excessive sleepiness starts to show, and difficulties in concentrating also tend to manifest. A way to classify the severity of this syndrome is the Apnea-Hypopnea Index. It is represented by the number of apnea and hypopneas events per hour of sleep. These apneas must last for more than 10 seconds and also be associated with the decrease in blood oxygenation. This index is divided into four categories:

- Normal : 0-4
- Mild sleep apnea : 5-14
- Moderate sleep apnea : 15-29
- Severe sleep apnea : 30 or more

Besides the Apnea-Hypopnea Index, the respiratory disturbance index RDI is also used in the medi-
cal community to classify the severity of the syndrome. This index has the formula:

$$RDI = (RERA + Hypopneas + Apneas) \times 60/TST$$

(2.1)

where TST stands for Total Sleep Time in minutes, and RERAs are respiratory effort related arousals. This index takes into account the occurrence of RERAs unlike the Apnea-Hypopnea Index. The RERAs occur when the respiratory effort is increased and leads to an arousal from sleep but does not fit the criteria for a hypopnea or apnea. The American Academy of Sleep Medicine recommends that RERAs are measured by esophageal manometry.

This index is divided into the same four categories as the Apnea-Hypopnea Index. OSA is thought to be a multifactorial pathology. Some of these factors are described in [14] and are presented here.

### 2.1.1 Anatomy

The upper airways are divided into four anatomical subsegments: the nasopharynx, located between the nostrils and hard palate; the retropalatal pharynx, located between the hard palate and soft palate; the oropharynx, which extends from the soft palate to the epiglottis; and the hypopharynx, which extends from the base of the tongue to the larynx. The last three subsegments compose the collapsible portion of the pharynx. The absence of bone and cartilage in these segments enables their lumen to remain permeable due to the action of muscles, which actively constrain and dilate the lumen of the upper airways [14]. Soft tissue structures, including the tonsils, soft palate, uvula, tongue and lateral wall of the pharynx, from the walls of the upper airways. The principal craniofacial bone structures that determine the dimensions of the upper airways are the mandible and hyoid. The transverse area of the various upper airway segments can be taken in different ways like an acoustic reflection of the conventional high-resolution tomography. Studies have shown a great diversity in these values. There is a consensus that the smallest upper airway diameter during wakefulness is found in the retropalatal oropharynx, which makes this point a potential locale for its collapse during sleep. However it is known that the obstruction can occur, either simultaneously or sequentially, in any of the four subsegments [15].

### 2.1.2 Respiration and Sleep

Pulmonary ventilation is controlled by two systems: an automatic one located in the brainstem; and a voluntary one in the cerebral cortex. The central chemoreceptors are sensitive to variations in pH; the increase of carbonic gas reduces pH, thereby stimulating those receptors. The peripheral chemoreceptors are sensitive to a decrease in arterial oxygen tension and in pH. Those chemoreceptors stimulate the respiratory centers located in the brainstem and control ventilation in an automatic or metabolic manner. Voluntary control is capable of dominating the function of the brainstem, within certain limits. For example, we can reduce $PaCO_2$ through voluntary hyperventilation. However, hypoventilation is more difficult since the interruption of ventilation is limited by metabolic factors [16]. The increase in $PaCO_2$ stimulates the respiratory center and increases ventilation. Active hypercapnia activates the upper air-
way dilator muscles and decreases their collapsibility by increasing the tension of the upper airway walls [17]. During sleep, the only respiratory control system that is active is the automatic one. Therefore, respiration during sleep depends on metabolic factors as well as on the activation of central and peripheral chemoreceptors. Sleep reduces the sensitivity of the chemoreceptors, de-activates the bulbar neurons and decreases overall motor activity. Some patients with OSA present chronic hypoxemia, which can affect the synthesis and activity of various neurotransmitters, thereby altering the function of central and peripheral chemoreceptors responsible for ventilatory control. It is believed that alterations resulting from sleep disorders perpetuate themselves during wakefulness. Patients with OSA and hypercapnia present a lower ventilatory response to hypercapnia, even during wakefulness. The alterations that occur in ventilation during sleep depend on the phases of sleep. In a normal individual In stages 3 and 4, ventilation is extremely regular with no alteration of the flow volume and respiratory frequency. In rapid-eye movement REM sleep, ventilation becomes irregular with variable flow volume and respiratory frequency, there is the loss of upper airway muscle tonus in relation to non-REM (NREM) sleep and, at certain moments, the arousal threshold can be high. Therefore, REM sleep is the moment of greatest risk for patients presenting severe, prolonged episodes of sleep apnea. These episodes occur at maximum frequency during REM sleep, stage 1 NREM sleep and stage 2 NREM sleep [15].

In calm respiration, the lungs are actively inflated through the contraction of the diaphragm, the external intercostal muscles and the accessory muscles involved in inspiration. There are other muscles involved like the nasal alae, responsible for opening the nostrils, and the small muscles of the neck and head. The coordinated activation of these muscles maintains the permeability of the airways that are prone to occlusions. In patients with OSA there is an accentuated reduction in upper airway diameter at the end of expiration, and so this period is considered critical and prone to occlusion. It is also believed that the upper airways are more complacent and collapse more easily in OSA patients, which is why the dilator muscles and more active during wakefulness and why there is a decrease in activity during sleep.

2.1.3 Age

The effect of age on OSA prevalence in the general population still needs to be clarified a bit more. Studies involving men and women between 20 and 100 years of age have been carried out in order to identify age-specific prevalences. Bixler et. al found that among women, those over 65 years presented the highest prevalence of OSA, whereas the highest prevalence in men was seen among those in the 45 to 64 age bracket [18]. Regarding the severity of OSA, the study by Bixler et. al [19], suggests that it is less severe in older individuals. It is also known that age correlates positively with obesity and neck circumference, and these are both risk factors for sleep apnea.

2.1.4 Gender

The prevalence of sleep apnea is greater in men than in women. In [18], Bixler et. al used clinical criteria and polysomnography finding the prevalence to be 1.2 % and 3.9 % in men. The prevalence of sleep
apnea is higher in menopausal women than in premenopausal women [18]. Hormone replacement therapy for postmenopausal women seems to be associated with a lower prevalence of sleep apnea. A difference between genders is the genioglossus muscle tone, which is believed to be greater in women, suggesting a defense mechanism for the maintenance of the upper airway permeability. Therefore, in the dorsal decubitus position, men present a greater reduction in the dimensions of the upper airway lumen than do women. Martin corroborates this hypothesis stating that when there is an increase in neck circumference due to fat accumulation, the transverse diameter of the upper airway remains greater in women than in men, suggesting one again that women present a better defense mechanism [20].

2.1.5 Posture and gravity

The shape and dimension of the upper airways depend on the position of the soft tissue structures (palate, uvula and pharyngeal wall), which can be influenced by gravity. In the dorsal decubitus position, the tongue and soft palate project themselves posteriorly, thereby reducing the area of the oropharynx [15]. A study was conducted in order to evaluate the upper airways area and volume during the dorsal decubitus position, which found no statistically significant difference between genders. However, men presented more significant changes in the dorsal decubitus position than did women [20]. It was defined by Cartwright and Lloyd [3] that patients whose RDI was at least twice as high in the dorsal decubitus position than in the lateral position were classified as positional apnea patients, whereas the other patients were considered nonpositional patients. Some patients have reduced their total RDI by sleeping on their sides [3, 21].

2.1.6 Anatomical Factors

Alterations in the external anatomy of the head and neck can be risk factors for developing OSA, irrespective of obesity. Craniofacial anomalies can involve a delay in the development of the mandible, producing mandibular retroposition. This causes the tongue to be positioned posteriorly and thus reducing the upper airways. In [15] they found out that in normal individuals, the upper airways present a lateral diameter larger than the anteroposterior diameter, whereas the inverse occurs in OSA patients. This corresponds to a narrowing at a critical point, the retropalatal airway. The structures that cause this effect are the lateral pharyngeal walls and the lateral pharyngeal fat pad. Another considered factor is the Mallampati score. This is a classification used in anesthesia, that measures the distance from the base of the tongue to the roof of the mouth, and tells how easy it is to perform endotracheal intubation. The classification is divided into four grades (I to IV), being grade IV the one corresponding to a smaller space between the tongue and the palate, consequently to a more difficult intubation. A high Mallampati score was shown in [22] by Nuckton et al. to be a predictor of the presence and severity of OSA.

2.1.7 Body fat

Obesity is an important pathogenic factor in sleep apnea. Approximately 70 % of OSA patients are obese, and obesity is the only significant risk factor that is reversible. Obesity of type 1 is defined as
having a Body-Mass Index (BMI) greater than $30\text{kg/m}^2$. Central or visceral obesity has been more frequently associated with apnea than have other forms of obesity. Some authors analyzed the distribution of body fat using magnetic resonance imaging of the neck and abdomen in patients under clinical suspicion of having sleep apnea. The results showed a significant correlation between sleep apnea and the quantity of intra-abdominal fat, as well as a borderline correlation between sleep apnea and subcutaneous abdominal fat and no correlation between sleep apnea and parapharyngeal or subcutaneous fat in the neck region [24]. Oksenberg [25], found that losing weight causes a drastic reduction of 91.1% of a patient’s RDI. This study also showed that as the body mass index increases, the positional dependency decreases, thus showing how critical the weight of a person is related to the probability of having OSA.

### 2.1.8 Genetic Factors

Familial aggregation of OSA has been shown in [26], with various affected members. The genetic factors related to craniofacial structure, distribution of body fat, neural control of the upper airways are all involved in OSA patients. Therefore, members of the same family that share genetic characteristics might present OSA. This prevalence can vary from 22 to 84% in first-degree relatives, hence the importance of investigating family history for evaluating patients [26].
2.2 Restless Leg Syndrome

Restless leg syndrome RLS is a common neurological sensory-motor disorder that is characterized by intense restlessness and unpleasant creeping sensations deep inside the lower legs. They force patients to keep moving their legs, and often to get out of bed and wander about. People suffering from this condition often have Periodic Limb Movements PLM during sleep, which reduces greatly their sleep efficiency. These uncomfortable sensations are experienced only when the limbs are at rest for a long time, and are relieved by movement [27].

2.2.1 Genetic Factors

Several authors have reported familial RLS, where at least 50% of the patients report a positive family history and a substantial number of investigations have been performed in the past years in order to identify genes associated with RLS. These studies have already found positive associations with sequence variants in or around specific genes or chromosomes. Moreover, the gene MEIS1 has been considered as the leading common genetic risk factor identified so far. This gene may have a function in the motor part of RLS and PLM [28, 29].

2.2.2 Iron Deficiency

There is an increasing bulk of data evidencing that RLS is common in individuals with low iron levels. It was reported by Nordlander [30], that the movement urges decreased after a few injections of an iron solution. Since then other studies have reported the same association of low levels of iron and RLS.

2.3 MEMS Devices

MEMS devices are a powerful, low power and low-cost technology used in several applications for sensing with precision and robustness. The following derivation of equations for the accelerometer can be seen [31]

2.3.1 Accelerometer

A MEMS accelerometer is an electromechanical device that measures acceleration forces. The forces can be static like gravity or dynamic caused by the movement of the accelerometer. Some accelerometers take advantage of the piezoelectric effect, that is, they have crystals that produce a voltage when stressed by accelerative forces. Others may use the changes in capacitance due to motion. Capacitors have excellent sensitivity and a good transduction mechanism, which is insensitive to temperature. Capacitive sensing is independent of the base material and relies on the variation of capacitance when the geometry of the capacitor is changing. The parallel-plate capacitance is given by:

\[ C = \varepsilon_0 \frac{A}{d} = \varepsilon \frac{1}{d} \]  \hspace{1cm} (2.2)
Where $\epsilon_0$ is the vacuum permittivity, $\epsilon$ is the medium permittivity, $A$ the area of the electrodes, and $d$ the distance between them. Any change in these parameters will be measured as a change in capacitance.

A typical MEMS accelerometer is composed of movable proof mass with plates attached to a mechanical suspension system as shown in figure 2.2. The movable plates and fixed outer plates represent capacitors. The deflection of the proof mass is measured using the capacitance difference. The free-space capacitances between the movable plates and two stationary outplates $C_1$ and $C_2$ are functions of the corresponding displacements $x_1$ and $x_2$.

\[
C_1 = \epsilon A \frac{1}{x_1} = \epsilon A \left( \frac{1}{d + x} \right) = C_0 - \Delta C, \\
C_2 = \epsilon A \frac{1}{x_2} = \epsilon A \left( \frac{1}{d - x} \right) = C_0 + \Delta C
\]

(2.3)

For a zero acceleration, the capacitances are equal because $x_1 = x_2$. The proof of mass displacement results due to acceleration and so for $x \neq 0$ the capacitance difference is found to be.

\[
C_2 - C_1 = 2\Delta C = 2\epsilon A \frac{x}{d^2 - x^2}
\]

(2.4)

Measuring $\Delta C$, one finds the displacement $x$ by solving the non-linear algebraic equation:

\[
\Delta C x^2 + \epsilon A x - \Delta C d^2 = 0
\]

(2.5)

We can omit the term $\Delta C x^2$ for small displacements.

\[
x \approx \frac{d^2}{\epsilon A} \Delta C = d \frac{\Delta C}{C_0}
\]

(2.6)

So we can conclude that the displacement is approximately proportional to the capacitance difference $\Delta C$.

As can be seen from the figure 2.2, we have several capacitors wired in parallel for an overall capacitance $C_1$ and likewise for $C_2$. 
The sensors fixed plates are driven by a given frequency and voltage $V_0$ that comes out of an oscillator. The output voltage $V_x$ and the proof mass can be obtained using

$$(V_x + V_0)C_1 + (V_x - V_0)C_2 = 0 \quad (2.7)$$

By making use of equations 2.6 and 2.7

$$V_x = V_0 \frac{C_2 - C_1}{C_2 + C_1} = \frac{x}{d} V_0 \quad (2.8)$$

Finally to relate this with acceleration, the plates can be compared with springs. According to Hook’s law for an ideal spring, it exhibits a restoring force, proportional to the displacement given by

$$F = -kx \quad (2.9)$$

Where $k$ is the spring constant. From Newton’s second law, assuming that only the restoring force acts on the spring we can say $F = ma = kx$, from which we can obtain

$$a = \frac{k}{m} x \quad (2.10)$$

Making use of the equation 2.8 we finally obtain

$$a = \frac{kd}{mV_0} V_x \quad (2.11)$$

Showing that the acceleration is proportional to the voltage output.

### 2.3.2 Gyroscopes

These derivations are done according to [32, 33]. All MEMS gyroscopes take advantage of the Coriolis effect. In the Smartwatch 3 we have a tuning fork gyroscope. In order to make use of the Coriolis effect, the device must be in constant rotation, which is achieved by vibrating the structure at one of its natural frequencies. This is referred as Drive Mode [32]. When the structure begins to rotate, the Coriolis force acting on the moving proof masses changes the direction of the vibration from horizontal to vertical. This vibration corresponds to a higher frequency than the Drive Mode frequency and it is referred as Sensing Mode. Metal plates are placed above the proof masses, and the whole set forms a capacitor. As the proof mass vibrates in Drive Mode, the distance between the proof mass and the plates is kept constant and so the capacitance is also constant. However, in Sensing Mode, the distance between plates changes and so does the capacitance. This change in capacitance can be detected and converted to indicate the corresponding rotation [32].
The Coriolis effect is the name given to the acceleration experienced by a moving point in a rotating reference frame. The Coriolis acceleration is defined as:

$$\vec{a}_{\text{Coriolis}} = 2\vec{\Omega} \times \vec{V}$$ \hspace{1cm} (2.12)

where $\vec{\Omega}$ is the angular velocity of the reference frame, and $\vec{V}$ is the velocity of the particle within this reference frame. The force necessary to keep the proof masses in motion in drive mode is provided by a comb drive transducer placed next to one of the proof masses, forming a capacitor. The plates on the comb drive and the side of the proof mass increase the surface area of the capacitor and allow changes in the system to be detected more easily. The capacitance $C$ for one finger of the proof mass is given by:

$$C = 2\epsilon_0 \frac{l_0 h}{g}$$ \hspace{1cm} (2.13)

Where $l_0$ is the length of the initial overlap between the finger on the proof mass and the finger on the comb drive, $h$ is the height of the gyroscope, $g$ is the gap between the fingers on the comb drive and the finger on the proof mass, and $\epsilon_0$ the permittivity of free space. To find the total capacitance of the system, multiply the capacity of one finger by the total number of fingers $n$.

The energy stored in a capacitor, $E$ is given by:

$$E = \frac{1}{2} CV^2$$ \hspace{1cm} (2.14)

Where $V$ is the voltage applied to the capacitor. Substituting for total capacitance we have

$$E = n\epsilon_0 \frac{l_0 h}{g} V^2$$ \hspace{1cm} (2.15)

For a small movement of the proof mass $x$ in drive mode the equation 2.15 becomes

$$E = n\epsilon_0 \frac{(l_0 + x) h}{g} V^2$$ \hspace{1cm} (2.16)

To find the force acting on the proof mass, take the derivative of the energy with respect to the position $x$ resulting in

$$F = \frac{dE}{dx} = n\frac{\epsilon_0 h}{g} V^2$$ \hspace{1cm} (2.17)

To prevent the capacitors to reach steady state it is applied an AC voltage to the comb driver and a DC voltage to use in Sensing Mode. With the two voltages applied the equation becomes

$$F = n\frac{\epsilon_0 h}{g} (V_{DC} + v_{AC})^2$$ \hspace{1cm} (2.18)
Expanding the square we obtain

\[ F = n\frac{\epsilon_0 h}{g} V_{DC}^2 + n\frac{\epsilon_0 h}{g} 2V_{DC}v_{ac} + n\frac{\epsilon_0 h}{g} v_{AC}^2 \]  

(2.19)

The first term is an offset and will be disregarded. The AC voltage is time dependent so it can be described by:

\[ v_{AC} = v_0 \sin(wt) \]  

(2.20)

with \( w \) the drive mode natural frequency of the gyroscope. By noting that the \( v_{AC} \) term is squared in equation 2.19 and that \( \sin(wt) = \frac{1 - \cos(2wt)}{2} \), it can be disregarded, leaving only,

\[ F = 2n\frac{\epsilon_0 h}{g} V_{DC}v_0 \sin(wt) \]  

(2.21)

Applying Newton’s Second Law to the system we have

\[ F = m\frac{d^2x}{dt^2} + D\frac{dx}{dt} + kx \]  

(2.22)

Where \( k \) is the spring constant, \( D \) is the damping coefficient, given by \( \sqrt{\frac{km}{Q}} \), where \( Q \) is a large number. As the displacement of the proof mass is also time dependent, it can also be written as

\[ x = x_0 \sin(wt) \]  

(2.23)

where \( x_0 \) is the displacement’s amplitude. The differential equation 2.22 can be solved for \( x_0 \), yielding

\[ x_0 = \frac{F}{k Q} = 2nQ\frac{\epsilon_0 h V_{DC}v_0}{gk} \]  

(2.24)

The current in AC state in a capacitor is given by

\[ i = \frac{dC}{dt} V_{DC} \]  

(2.25)

Using the capacity \( C = \frac{2n\epsilon_0 h}{g} (l_0 + x) \) from before and taking the derivative we obtain

\[ \frac{dC}{dt} = \frac{2n\epsilon_0}{g} \frac{dx}{dt} \]  

(2.26)

The sinusoidal movement from \( x = x_0 \sin(wt) \) can be written as \( x = x_0 e^{jwt} \) and so the derivative \( \frac{dx}{dt} \) becomes \( (jw)x \). Replacing this expression in the current we have the value of the current necessary for producing the vibration in Drive Mode.

\[ i = V_{DC} \frac{2n\epsilon_0 h}{g} (jwx) \]  

(2.27)

For the sensing mode, a pair of metal plates is used forming a parallel plate capacitor with the proof mass. Both plates and proof mass are connected to a detector that helps amplifying the signal to help
detection. As shown before the capacitance is given by \( C = \epsilon_0 \frac{A}{d} \). The sensing mode current can be expressed as

\[
i_s = C_0 \frac{dy}{dt} V_{DC} \quad (2.28)
\]

and note that \( \frac{dy}{dt} = w_s y_0 \) where \( w_s \) is the sense mode angular frequency, \( y_0 \) the amplitude of vibration. As the amplitude of vibration is function of the force applied to the proof mass we have

\[
y_0 = \frac{F Q_s}{k - k_{elec}} \quad (2.29)
\]

Where \( Q_s \) is the damping coefficient, \( k \) is the stiffness constant of the system and \( k_s \) is the Electrostatic Stiffness, defined as \( k_{elec} = \frac{C V_{DC}^2}{d_0} \).

The force due to the Coriolis effect can be expressed as a scalar by

\[
F = 2m \Omega x_0 \omega \quad (2.30)
\]

Replacing this in equation 2.29 we obtain

\[
y_0 = \frac{2m \Omega x_0 \omega Q_s}{k - k_{elec}} \quad (2.31)
\]

The amplitude \( y_0 \) can be replaced in the current equation yielding

\[
i = \frac{CV_{DC} \omega 2m \Omega x_0 \omega d Q_s}{d_0 (k - k_{elec})} \quad (2.32)
\]

### 2.3.3 Magnetometer

The Earth’s magnetic field is a vector quantity that has both magnitude and direction. Its magnitude ranges from 25 to 65 \( \mu \)T, and its direction has a component parallel to the Earth’s surface that always points towards the magnetic North. This is the basis for all magnetometer measurements. Measuring this vectorial quantity can be quite a challenge as the Earth’s magnetic field is relatively weak when compared to surrounding electromagnetic interferences. However there are a few methods for measuring this field, the first one being the use of an Anisotropic Magneto-Resistance (AMR) sensor. This sensors are made of a permalloy film deposited on a silicon wafer and patterned as a resistive strip. The properties of this film cause it to change resistance in the order of 3% when in the presence of a magnetic field. Typically, a Wheatstone bridge configuration with 4 resistors is used to measure both magnitude and direction along a single axis. A disadvantage of this method is its sensitivity to temperature.[34] Another method is the Giant Magneto-Impedance GMI sensor, which is a highly sensitive micromagnetic sensor based on the magneto-impedance effect. When a soft ferromagnetic conductor - an amorphous wire - is subjected to a small alternating current, a large change appears in the AC complex impedance of the conductor and can be detected by applying a magnetic field change. Based on Faraday’s law this invokes a voltage output given by:

\[
V = -nA \frac{dB}{dt} \quad (2.33)
\]
Yet another method widely used in mobile devices is based on the Hall effect. It works on the principle that voltage $V_{\text{hall}}$ can be developed in a direction transverse to the current flow in a system of charged particles in a magnetic field, owing to the Lorentz force

$$F = q(\vec{v} \times \vec{B})$$

(2.34)

Measuring the Hall voltage output across a metallic surface, the proportional magnetic field can be derived. This method is very low cost and used predominantly in smartphones today. The main drawback for this method is the temperature compensation, because increasing temperature increases the electron’s motility [34].

### 2.3.4 Rotation Vector

A rotation vector is a virtual sensor present in some Android devices that represents the device’s orientation. We refer to this sensor as virtual, as it is purely software based, there is no physical rotation sensor. The readings from this sensor result from computations performed by the operating system internally. In figure 2.4 there’s an example of how the device orientation changes when a subject moved from a lateral to supine position. This is done by using values from the accelerometer and gyroscope. We need to use both sensors as accelerometers are generally very noisy and gyroscopes tend to drift over time. This drift results from the integration of the gyroscope values to obtain the rotation angle. Using the accelerometer solely also leads to incorrect values of position, as when integrating the acceleration signal to obtain velocity, we are also integrating the inherent noise from the accelerometer. This noise increases, even more, when the integration is done to obtain the displacement from the velocity leading to larger errors in the position signal. By integrating the value obtained from the gyroscope we can determine the angle which the device turned and by that compute its orientation, an example of this computation is found in [36].
Figure 2.4: Example output from the rotation vector on a movement going from a lateral to supine position, from top to bottom we have the $x$, $y$ and $z$ components of the rotation vector over time.
To understand how the rotation sensor work, we will cover Kalman filters first.

A Kalman filter operates by producing a statistically optimal estimate of the system state based upon the measurements. Suppose that the state of a discrete-time system at time $k$ is given by

$$x_k = Fx_{k-1} + Bu_k + w_k$$  \hspace{1cm} (2.35)

with $x_k$ given by

$$x = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}$$

Where $\theta$ is an angle of orientation of the device and $\dot{\theta}_b$ is a bias based upon the measurements from the accelerometer and gyroscope. This bias is the amount the gyro has drifted, $u_k$ is the gyroscope measurement, and $w_k$ we consider it to be Gaussian noise with 0 mean and a covariance matrix $Q_k$, defined as

$$Q_k = \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

$F$ and $B$ are defined as

$$F = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} \Delta t \\ 0 \end{bmatrix}$$

Next step is to take a look at the measurement of the state $x_k$, given by $z_k$

$$z_k = Hx_k + v_K$$  \hspace{1cm} (2.36)

$H$ is called the observation model and is used to map the true state space into the observed space. The true state cannot be observed. Since the measurement is just the measurement from the accelerometer, then $H$ is given by

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$v_k$ is just gaussian noise with zero mean, and covariance matrix $R$, we assume that this variance does not change over time.

**Kalman Filter**

The first equations of the Kalman filter are to estimate the current state of the system based on previous states, that is given by

$$\hat{x}_{k|k-1} = F\hat{x}_{k|k-1} + B\dot{\theta}_k$$  \hspace{1cm} (2.37)
Next we need to estimate the a prior covariance matrix $P_{k|k-1}$ based on the previous covariance matrix $P_{k-1|k-1}$, defined as

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q_k \tag{2.38}$$

The first thing we do is to compute the difference between the measurement $z_k$ and the a priori state $x_{k|k-1}$, this is called innovation

$$\tilde{y}_k = z_k - H\hat{x}_{k|k-1} \tag{2.39}$$

The next step is to calculate the innovation covariance $S$, given by

$$S_k = HP_{k|k-1}H^T + R \tag{2.40}$$

Up next we compute the Kalman gain defined as

$$K_k = P_{k|k-1}H^T S_k^{-1} \tag{2.41}$$

If one does not know the state of the system at startup, the matrix $P$ can be initialised as a diagonal matrix with a large number on both entries. Finally let’s move to the computation of $\hat{x}_{k|k}$, given by

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\tilde{y}_k \tag{2.42}$$

And then we must update posteriori error covariance matrix

$$P_{k|k} = (I - K_kH)P_{k|k-1} \tag{2.43}$$

So with these computations we have estimations for the orientation over time.
Chapter 3

Android Application Development

In this work, we used a Sony Smartwatch 3®, which runs on Android Wear OS (6.0 Marshmallow) and a Smartphone LG G4c running on Android OS 6.0. Android Smartwatches need to be paired with a smartphone in order to function properly. The pairing is mediated by the Android Wear application which can be downloaded from Google’s Play Store. For the pairing to occur Bluetooth must be turned on the smartphone and then the app will search for a response on the watch side and pair with it. Pairing allows communication on both sides (sending files, images, music, messages), and it is by means of this communication that the smartphone will send a message to the watch to start registering the sensors’ values. The development of the application for the phone and watch was done on Android’s official IDE, Android Studio. The applications are developed in Java, the compilation and packaging of the Android Application Package APK is performed by Gradle a build system for the Java Virtual Machine JVM, Gradle is also responsible for managing all the dependencies between code modules, i.e. in case one has to import an external library for the project.

3.1 Application Lifecycle

Android applications follow a paradigm different from most common applications, whereas common Java apps are launched with a main() method, Android apps are initiated in what is called an Activity. An Activity is a Java class that has certain callback functions, responsible for controlling the flow of the application, and there can be several activities on a single application. A good practice is to create a separate activity for each functionality that the application has. Imagine that one develops an application that takes pictures using the phone’s camera and that allows sending those pictures by email. Here one would have an activity responsible for taking the pictures (call it CameraActivity), and an Activity for handling sending the pictures by email (call it EmailActivity). All the activities must be declared on an XML file called “AndroidManifest.xml”, this is the file that the operating system reads to know what part of the code to execute when the application is started, which is the first activity to launch, what permissions are required to run the app (Bluetooth, Internet, Sensors, access to device’s storage). Every activity has six callback functions that are associated with the application lifecycle. The functions are as follows, and
Figure 3.1: A simplified illustration of the Activity lifecycle, expressed as a step pyramid. This shows how, for every callback used to take the activity a step toward the Resumed state at the top, there’s a callback method that takes the activity a step down. The activity can also return to the resumed state from the Paused and Stopped state.

are displayed on figure 3.1

- **onCreate** - This function is called when the activity starts for the first time ever (e.g. when the app launches)

- **onStart** - This function is called when the activity comes to the front screen, i.e. when the phone screen is turned with the application open, then the onStart function is called

- **onResume** - This function is called when the activity is resumed, i.e. by being another activity (can be the EmailActivity) and press the back button on Android to go back to the CameraActivity, then the onResume function is called as this activity was already started in the past.

- **onPause** - This is the complementary function of onResume, as it is called when another activity is started (going from Email to Camera activity will call the onPause on the Email Activity).

- **onStop** - This is the complementary function of onStart, whenever the display is turned off or another application is started the onStop function is called.

- **onDestroy** - This is the complementary function of onCreate, as it is called when the application is terminated by the operating system or by the user.

Knowing when these functions are called is important for the application development. With respect to this work, the setup for GoogleApiClient, PowerManager and Sensor Manager related instances must be done in the onCreate method to have them all ready for use as soon as the application finishes loading, the destruction of these objects is only called on the onDestroy method to avoid memory leaks or unnecessary object recreation. As Activities are meant for UI interaction, all the background operations should run in another class, in Android these classes are called Services. A Service is a class that runs in background and performs operations like retrieving locations, listening for messages, handling sensor
values updates. Services are declared to handle all the heavy work while the user interacts with the application on the UI level. Services must also be declared on the AndroidManifest file. The whole flow of the application is illustrated in the figure 3.2.

Android has made publicly available their Application Program Interface API, on which the most important ones used for the development of the applications are the GoogleApiClient, NodeApi and Wearable.MessageApi, PowerManager and the SensorManager.

### 3.2 GoogleApiClient

*GoogleApiClient* is the main entrypoint for GooglePlay services integration. It is through this API that the communication is established between the smartwatch and the smartphone. An instance of this class needs to be called on the smartphone side.

```java
    googleApiClient = new GoogleApiClient.Builder(this)
        .addApi(Wearable.API)
        .build();
    googleApiClient.connect();
```

The method `connect()` returns a callback function whether the connection was successful `onConnected`, or if the connection failed `onConnectionFailedListener`. These callbacks are not used in this work, but can be useful for debugging possible failures in connecting the watch to the smartphone. When the connection is established, the googleApiClient makes available other API that allow the exchange of information between devices, in the current case the API used were the NodeApi and Wearable.MessageApi. [37] Although the application for the smartphone was meant to be used by technicians at Hospital de Santa Maria, it was designed to be as simple to use as possible to avoid errors when recording data. On the background when the button is pressed and message is sent to the watch. The messages sent...
Figure 3.3: Doze provides a recurring maintenance window for apps to use the network and handle pending activities, taken from [39].

were either "START" or "STOP".

### 3.3 Node Api and Wearable.MessageApi

The NodeApi is responsible for handling the connected devices, whereas the Wearable.MessageApi is the one to send and receive messages between the devices.[38] For receiving messages on the watch a Service must be implemented in order for the messages to be delivered at any time, without the need of having an application running and consuming battery.

### 3.4 PowerManager

PowerManager is an Android class that allows to control the power state of the device. With the introduction of Android 6.0, Google introduced Doze Mode which goal is to preserve battery life. Doze mode is entered when the smartphone or the smartwatch is not either moving or charging. When Doze mode is active the device defers operations that it is performing to a queue to be executed later on. [39] All these operations are then performed periodically (see figure 3.3). Doze mode can be exited by picking up the phone or turning the screen on. This class highly affects the device's battery life and it is not recommended to use unless it is really needed. In this case this was necessary to keep the sensor values being registered with a regular sampling frequency, as when people fall asleep the variations in registered sensor values are perceived as it is idle and so Doze mode activates. As in Doze mode tasks are queued up and executed periodically, the sensors sampling frequency is highly affected a drops from 32Hz to 0.02Hz. To avoid, this we acquire an instance of a WakeLock object from the PowerManager and program it to keep Watch out of Doze mode while the sensors are registering values. This is called a Partial Wake Lock and allows the device's screen to go off in order to save battery but keeps the CPU active. After the sensors stop recording we must release the WakeLock object to preserve battery life.
3.5 SensorManager

Sensor Manager is the name of the class used to handle all sensor values [36]. This class must initiated in a SensorEventListener Service class in Android. For every sensor that will be used, we need to call the registerListener method with the following signature:

\[
\text{registerListener}(\text{SensorEventListener} \ \text{listener}, \text{Sensor} \ \text{sensor}, \text{int} \ \text{samplingPeriodUs})
\]

Here, listener refers to the class where the SensorManager is implemented, sensor is the name of the sensor that we need to work with, and the sampling period is the desired delay between two consecutive events in microseconds. Below is a snippet on how to start using the accelerometer sensor at 32Hz.

\[
mSensorManager.registerListener(this, accelerometerSensor,31250);
\]

To stop working with a sensor, the method unregisterListener must be called. The SensorEventListener Service has a method that is executed each time a new value is obtained from the sensors, this new value comes in a form of an event object that has the following properties:

- **Type** which refers to the type of sensor that produced this event, being it the accelerometer, gyroscope, magnetometer or rotation vector;
- **Accuracy**, which refers to accuracy of the measurement, it can be HIGH, MEDIUM or LOW. The accuracy depends on the level of workload that Android is going through at the moment, being high when there is only the listener running and low with a lot of background processes;
- **Timestamp**, which is the time in nanoseconds at which the event was generated;
- **Values**, Which are the sensor’s measurements and come in an array.

Below is the image of the device’s coordinate system. It is important to know how to interpret the values registered from the sensors.

![Android's Coordinate System](image)

Figure 3.4: Android's Coordinate System taken from [36].

**Accelerometer**

The outputs of the Accelerometer are in SI units \( m/s^2 \), and they come as an array of size 3.

- **values[0]** Acceleration measured on the \( x \)-axis;
- **values[1]** Acceleration measured on the \( y \)-axis;
• values[2] Acceleration measured on the z-axis.

As an example, when the device is placed on a table with the screen facing up, the accelerometer reads 9.81 m/s².

**Gyroscope**

All values are measured in radians/second and measure the rate of rotation around the device’s local X, Y, and Z axis. The coordinate system is the same as the one used for the acceleration sensor. Rotation is positive in the counter-clockwise direction.

• values[0] Angular speed around the x-axis;
• values[1] Angular speed around the y-axis;
• values[2] Angular speed around the z-axis.

**Magnetometer**

All values are measured in micro Tesla and measure the ambient magnetic field in the X, Y, and Z axis. The coordinate system is once again the same used for the acceleration sensor.

• values[0] Magnetic field measured the x-axis;
• values[1] Magnetic field measured the y-axis;
• values[2] Magnetic field measured the z-axis.

**Rotation Vector**

All values are unitless and represent the orientation of the device as a combination of an angle and an axis, in which the device has rotated through an angle θ around an axis jx, y, zj. The x, y, and z axis are the same as the one defined for the other sensors.

• values[0] \( x \times \sin(\theta/2) \);
• values[1] \( y \times \sin(\theta/2) \);
• values[2] \( z \times \sin(\theta/2) \);
• values[3] \( \cos(\theta/2) \).
Chapter 4

Experimental Acquisition of Sleep Movements

For our work we defined 4 possible positions in which the patient can be. This was defined according with the position nomenclature used by the PSG system used in Hospital de Santa Maria, also it is the same found in [1], and is listed below.

- Supine Position, where the person is lying on its back;
- Right Lateral Position, where the person is lying on its right side;
- Left Lateral Position, where the person is lying on its left side;
- Prone Position, where the person is lying on its stomach

![Figure 4.1: Body positions while lying on bed adapted from [1]. From left to right, we have left lateral position, right lateral position, prone position and supine position.](image)

The approach taken to develop the algorithm was to train a sleep movement classifier, because by knowing which movement was performed, we can derive the position from it. We defined 8 possible sleep movements as follows:

- From supine to left lateral position;
- From supine to right lateral position;
- From left lateral to supine position;
• From right lateral to supine position;
• From left lateral to prone position;
• From right lateral to prone position;
• From prone to left lateral position;
• From prone to right lateral position

There can be other movements considered like left lateral to right lateral position but these movements are a composition of the ones stated above. A classifier needs a great amount of data to be trained, so we acquired several examples of each movement. The acquisition was done in three different ways, that are described in the following sections

4.1 Acquisition of Sleep with a Kinect

A way to acquire the sleeping movements in their most natural way is to record a volunteer’s sleeping night while using the Sony Smartwatch 3 and recording it with a Kinect V2 from Microsoft, with the purpose of having some way to confirm the actual position. The sampling frequency of the smartwatch was kept at 32Hz. As was also done in the recordings from Hospital de Santa Maria, the device was placed on the patient’s left hand.

The Kinect V2 has an infrared camera with a high resolution that allows discriminating perfectly the body position while sleeping. The camera would be set next to the volunteer’s bed and just before the volunteer went to sleep he/she had to run a computer script that prompted for his/her name and after that, the Kinect would start recording. The watch would start recording as well after the volunteer pressed the button on the smartphone application, as is illustrated in figure 4.3. If the watch started recording successfully a warning would display on the smartphone screen. The same applies if the user was able to stop the recording, which is also shown in figure 4.3.

The video files from each night occupy a high amount of space in the computer (around 90Gb/hour) so the script would only allow to record for 8 hours and then shutdown the computer. This was done to prevent volunteers from forgetting to stop Kinect’s recording and the files from growing into astronomical sizes. The data processing would occur after when we looked at the video recording and compare it with the accelerometer signal to identify each movement and extract it to the movement’s database, or add it to the algorithm test set. When the recording on the watch stops, the next operation is to write all the recorded values to the CSV files. As the whole night recording is per se an expensive battery process the watch needed to be plugged to a computer before stopping the recording and writing the CSV files, or all the information would be lost in case the battery would run out. The reason why we did not implement the two threads at the same time (recording and writing) was because these are heavy operations to perform on a smartwatch/smartphone, that lead to the loss of sensor data, also when performing two operations at the same time the sampling frequency from the sensors tends to decrease over time with the amount of data being written.
4.2 Recordings in Hospital de Santa Maria

During the realization of this work, patients from the Neurology department in Hospital de Santa Maria were asked to take part in this work. These patients went to the Neurology consultation to realize a PSG due to suspicions of sleep disorders such as sleep apnea. This procedure started by them entering the Neurology consultation to be prepared for the PSG. That preparation would consist in the technician placing sensors on the scalp and temples (for EEG), a blood oxygen sensor on the finger, and a device in the chest area where all the cables would be connected. It is also in this device where all the sensor readings are stored. After the preparation — and if the patient would agree to participate — a consent would be signed and then the device was placed on the left hand. The PSG started recording by pressing a button on the device attached to the chest, and in order to synchronize the recording with the device, the pressing of the PSG button occurred at the same time as the button pressing on the smartphone to trigger the recording in the device. After the preparation, the patient would go home and sleep the whole night with the watch recording, or stay in the hospital in case the PSG required a video recording as well. The total time from start until the stop of the PSG would take around 13-15 hours and the Sony Smartwatch 3 battery did not last that time while recording, so we used an Empatica E4 wristband paired with an iPhone. The disadvantage of this wristband is that it only has an accelerometer when compared to the accelerometer, gyroscope, magnetometer and rotation vector from the Smartwatch. The accelerometer from the wristband has a fixed sampling rate of 32Hz, unlike the smartwatch which can be set manually by code. The wristband also has a PPG sensor which readings are displayed on the iPhone as shown in figure 4.4.

When the patient returned to the hospital or woke up the recording would stop (in case the patient slept in the Hospital). The Empatica E4 streams all data through Bluetooth to the iPhone. For each recording the files written would have a different id associated to each patient (i.e. 0001_acceleration.csv, 0002_acceleration.csv). The same id would be present on the PSG signal. The PSG would be later analyzed by medical doctors at Hospital de Santa Maria. The position signal from PSG is an integer-valued signal where each position corresponds to a single integer and has the following notation:

- 2, For upright position, when the patient is standing or sitting;
- 3, For left lateral position;
• 4, For right lateral position;

• 6, For supine position;

• 8, For prone position.
4.3 Acquisition of Sleep Movements in Awake Volunteers

For the recording of sleep movements, we developed an Android Wear application that allowed to record the values from the smartwatch’s accelerometer, gyroscope, magnetometer and rotation vector and store them in CSV files on the watch. The application usage required two people, one for controlling which movements to perform and another, a volunteer that would put on the watch and perform the desired movements. The person controlling the phone would select the movement to execute within a list (see figure 4.5). After selecting the movement, to start the recording one just needed to press the Record button, and tell the volunteer to execute the movement, after that one would press the Stop button on the smartphone, and the data would be stored in a folder corresponding to the movement performed. For each volunteer, all movements would be performed several times in no particular order and never doing the same movement twice in a row to allow the data to be as varied as possible. After recording the movements all the CSV files would be extracted from the watch by plugging it to the computer and with the help of Android's ADB utility tool and executing the command `adb pull /sdcard/sleep_data`. This command retrieves all the stored files that are on watch’s folder `sleep_data`. The acquisition was performed in $n = 20$ different individuals, with ages between 18-51 years. None of the individuals participated in the Kinect recordings analyzed in chapter 6. The sampling frequency for the accelerometer was kept at 32Hz, for the gyroscope was 100Hz, for the magnetometer 10Hz, and rotation vector 5Hz (although it can change with time). The number of examples for each movement is detailed in table 4.1. There is not the same number of examples for each class due to communication issues with the watch that failed to send the start or stop information, which lead for the files to be corrupted.

As was done later on, only 3 classes were considered. These classes are an aggregation of the 8 mentioned before. The aggregation was done in a way that classes ending in supine position belong to

![Figure 4.5: Application used for the recording of sleep movements](image)
Table 4.1: Number of examples for each movement, of a total of 460 different examples.

<table>
<thead>
<tr>
<th>Movement Type</th>
<th># Examples</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supine - Left Lateral</td>
<td>64</td>
<td>13.9%</td>
</tr>
<tr>
<td>Supine - Right Lateral</td>
<td>79</td>
<td>17.1%</td>
</tr>
<tr>
<td>Left Lateral - Supine</td>
<td>69</td>
<td>15.0%</td>
</tr>
<tr>
<td>Right Lateral - Supine</td>
<td>64</td>
<td>13.9%</td>
</tr>
<tr>
<td>Left Lateral - Prone</td>
<td>41</td>
<td>8.9%</td>
</tr>
<tr>
<td>Right Lateral - Prone</td>
<td>48</td>
<td>10.4%</td>
</tr>
<tr>
<td>Prone - Left Lateral</td>
<td>45</td>
<td>9.8%</td>
</tr>
<tr>
<td>Prone - Right Lateral</td>
<td>50</td>
<td>10.9%</td>
</tr>
<tr>
<td>Total</td>
<td>460</td>
<td>100%</td>
</tr>
</tbody>
</table>

the first class (Supine Class), movements ending in a lateral position belong to the second class (Lateral Class), and finally movements ending in a prone position belong to the third class (Prone Class). This means merging the classes Supine-Left Lateral, Supine-Right Lateral, Prone-Left Lateral, Prone Right Lateral into one class, Left Lateral - Supine, Right Lateral Supine into another and finally Left Lateral - Prone and Right Lateral - Prone into another. The distribution was as follows:

Table 4.2: Number of examples for each movement considering the position on which they ended, using 3 classes, of a total of 460 different examples.

<table>
<thead>
<tr>
<th>End Position</th>
<th># Examples</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supine</td>
<td>133</td>
<td>28.9%</td>
</tr>
<tr>
<td>Lateral</td>
<td>238</td>
<td>51.7%</td>
</tr>
<tr>
<td>Prone</td>
<td>89</td>
<td>19.3%</td>
</tr>
<tr>
<td>Total</td>
<td>460</td>
<td>100%</td>
</tr>
</tbody>
</table>

We also tried to merge the Lateral and Prone classes into a single one, in order to build a binary classification problem (Supine vs Non Supine). The distribution was as follows:

Table 4.3: Number of examples for each movement, considering the position on which they ended, using 2 classes, of a total of 460 different examples.

<table>
<thead>
<tr>
<th>End Position</th>
<th># Examples</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supine</td>
<td>133</td>
<td>28.9%</td>
</tr>
<tr>
<td>Non Supine</td>
<td>327</td>
<td>71.1%</td>
</tr>
<tr>
<td>Total</td>
<td>460</td>
<td>100%</td>
</tr>
</tbody>
</table>
Chapter 5

Implemented Classifiers and
Methodology

This chapter aims to describe the mathematical details behind the implemented classifiers present in this work. First, we go over the derivations for the classifier equations and then present the way the classifiers themselves are used in the algorithm.

5.1 Algorithm Flow

A first procedure, before analyzing the signals from Hospital de Santa Maria was to synchronize them with the Position signal from the PSG. The accelerometer has a sampling frequency of 32Hz, whereas the position signal only has 4Hz, so it had to be upsampled by a factor of 8. The signals from the Kinect video recordings did not need any resampling as the position signal was built manually. The accelerometer signal is then analyzed using a sliding window approach of constant duration of $t = 6s$. On each windowing a feature vector is extracted from the accelerometer signal and is passed as input for the classifier (Artificial Neural Networks (ANN), Support Vector Machines (SVM) or Naive Bayes classifier).

5.2 Transition Detection

The first step before classifying the position is detecting changes in position done by the patient during the night. This is done by analyzing the accelerometer data and comparing it with either the PSG position signal or with the video recording from the Kinect (depending on the origin of the data). A challenge here is discriminating between spurious movements that occur during the night and that are captured by the accelerometer and between real position transitions. These changes in position correspond to fast variations on the accelerometer signal, hence to detect them we will resort to the derivative of the accelerometer signal. The derivative of the accelerometer signal will be represented by $S$. The method used to detect these movements used two signal features that were as follows.
Here, \( m \) can be seen as the absolute value of the derivative of the accelerometer signal. For sleeping movement detection this value had to be between 3 and 10. These values were computed for the whole sleeping movement dataset. Plotting them in MATLAB allowed the establishment of these empirical values.

\[ m = S^T S \]  

(5.1)

5.3 Feature Selection

The selection of features was done manually, by running statistical tests between each pair of classes and selecting features where the p-value for each comparison was lower than 0.05. Figure 5.1 shows the results obtained from the pairwise comparison between each feature. This process helps reducing the number of features in the model, and accelerating the training process. Representing the accelerometer signal by \( a \), as a matrix, where each row is the acceleration on the \( x, y, \) and \( z \) directions respectively, the initially selected features are listed below. The numbers next to each item on the list are the correspondence to the numbers on image 5.1.

- Mean of the acceleration signal on each direction at the start of the transition (1-3);
- Mean of the acceleration signal on each direction at the end of the transition (4-6);
- Acceleration standard deviation on each direction (7-9);
- Correlation between \( xy, yz \) and \( xz \) signals (10-12);
- Correlation between \( xy, yz \) and \( xz \) signals from the gyroscope;
- Median on each direction (13-15);
- Interquartile range on each direction (16-18);
- Number of samples above zero (positive acceleration) on each direction (19-21);
- Number of samples below zero (negative acceleration) on each direction (22-24);
- Determinant of \( a^T a \) (25).

Using the correlation between different directions of the signals was used by Ravi et. al [40], where accelerometer data was used to identify the type of activity performed. As in our work, the accelerometer was also placed on the wrist. This proved to be extremely useful in distinguishing between activities that involved translation in just one direction. Our idea was that these features would allow distinguishing between movements that occur in opposite direction (i.e. from supine to right lateral and prone to left lateral position). The signal mean and the standard deviation are also present in [40], and are commonly used features for pattern recognition. The number of samples above and below zero is the novelty in here as to our knowledge it was never used before in gesture recognition. The purpose of using the
Figure 5.1: Obtained p-values from pairwise comparison between features, using 3 classes. On the vertical axis we have each feature in the same order as they are present in section 5.3. The numbers on the list of features are the direct correspondence to each row on the image. First column is the comparison between supine and prone classes, second column is supine and lateral classes, and third column is the comparison between lateral and prone. The scale on the right represents p-value.

Figure 5.2: Acceleration signal on the z-axis on a change from supine to right lateral position. Vertical axis is in $m/s^2$. 
mean at the start and end of the transition is illustrated in figure 5.2, as clearly there is a change in values before and after the transition. Other works like [41], also placed an accelerometer on the wrist and used the Wavelet transform and its decompositions to extract features from signal. This work focused on detecting the type of activity performed (walking, running, sitting, standing). In here we chose not to use this feature, as sleep movements consist of alterations in position and these changes occur almost always in the same time interval, thereby extracting frequency-related features was not judged to be useful. We did not use any of features for the magnetometer signal, as the values from this signal are influenced by the surrounding environment, i.e. if the volunteer was close to any electrical device the signal would have that influence. We also did not include features from the Rotation Vector as this sensor would have a varying sampling frequency, which would change for example if an e-mail notification arrived at the watch.

After comparing the p-values obtained from the pairwise comparison between features, we opted to use only 5 features as our dataset is not that large and also to simplify the model. The selected features were as follows. Further reasons for using a small number of features are presented in chapter 6.4.

- Acceleration mean at the end of the transition on the \( x \) axis;
- Acceleration mean at the end of the transition on the \( y \) axis;
- Correlation between acceleration signal on direction \( x \) and \( y \);
- Correlation between acceleration signal on direction \( x \) and \( z \);
- Number of samples with acceleration greater than 0 on the \( y \) direction.

5.4 Artificial Neural Networks

In machine learning, ANNs, are a family of models inspired by biological neural networks, used for pattern recognition, estimation or approximation of functions, computer vision, speech recognition. The networks are represented as a system of connected neurons, where each neuron has connections to other neurons or to the output of the network and these connections have numeric weights that are tuned by training the network. By tuning these weights the networks is able to learn the desired model. A commonly used kind of neural network is the multilayer perceptron MLP. A MLP is formed by a set of units as shown in figure 5.3, where each of the inputs is weighted and summed at the node. This sum is then passed through a unit which contains an activation function (usually a non-linear activation function like the hyperbolic tangent).

Overall the networks can be divided into three sections, an input layer, a hidden layer, that can contain more than one layer of neurons and an output layer. In a typical network, a neuron in each layer is connected to all other neurons in the following layer as shown in figure 5.4. As for the activation functions, one of their objectives is to set the output from a neuron between 0 and 1, or between -1 and 1. Some activation functions are:

\[
f(x) = \tanh(x)\]

(5.2)
$f(x) = \arctan(x)$ \hspace{1cm} (5.3)

Other than the weights from each neuron, we also add a fixed input to each neuron which is called the bias, represented in 5.3 by $w_0$.

In order to train these networks, we use the so-called backpropagation algorithm, which is described below. In this work, we used MATLAB’s Neural Network toolbox [44] that already has implemented the algorithms described below.

### 5.4.1 Backpropagation Algorithm

The presentation of this algorithm is done according to the one in [42] by Luís B. Almeida.

The input pattern of a network is represented by an $m$-dimensional vector $\vec{x}$, and the output units of that unit by an $N$-dimensional vector $\vec{y}$. The input units of the network copy the components of the input pattern such as $\vec{y}_i = \vec{x}_i$, $i = 1, ..., m$. We will also assume that there is a unit number 0, whose output is fixed at 1. The weights from this unit to other units of the network represent the bias terms of those units. Denoting by $w_{ij}$ the weight in the branch that links unit $j$ to $i$, the weighted sum performed by unit
i is written as
\[ s_i = \sum_{j=0}^{N} w_{ji} y_j. \] (5.4)

The outputs \( y_j \) are computed as follows
\[ y_i = S(s_i) \quad i = m + 1, \ldots, N \] (5.5)

Where \( S \) represents the activation function. The output of the network is represented by the vector \( \vec{o} \).

Denoting by \( \vec{x}^k \) the kth training pattern of the network, and its desired output represented by \( \vec{d}^k \), we can define an error vector \( e^k \) in the following way
\[ e^k = o^k - d^k \] (5.6)

We use the squared norm of the error vector \( E^k = \|e^k\|^2 \) as a scalar measure of the deviation of the network from its ideal behavior for each pattern. \( E^k \) is zero if the output of the network matches exactly the desired output. For the whole training set we have
\[ E = \sum_{k=1}^{K} E^k \] (5.7)

By fixing the training set and the network’s architecture, the error \( E \) is only a function of the weights, so to train the network we need to find the weights that minimize \( E \). To minimize this error function we can use the gradient method, that consists in taking steps iteratively proportional to the negative gradient of the function to be minimized, this can be written as:
\[ \vec{w}_{n+1} = \vec{w}_n - \eta \nabla E \] (5.8)

where \( \nabla E \) represents the gradient of \( E \) with respect to \( w \). This procedure is repeated until some criterion is met. \( \eta \) is normally designated as the learning rate parameter or step size parameter. The key here is how to compute the gradient components. A first step would be to invert the network, where we reverse the direction of all the branches, replace summing nodes with divergence nodes and change outputs to inputs and vice-versa (see 5.5 for further clarification). The nonlinear elements are replaced by linear branches with gains defined as \( g'_i = S'(s_i) \), being \( S' \) the derivative of the activation function.

The derivation of the following expression can be consulted in [45], but it comes that
\[ \frac{\partial E^k}{\partial w_{ji}} = y_j \bar{s}_i \] (5.9)

Where \( \bar{s}_i = g'(s_i) \bar{y}_i \).

This algorithm is guaranteed to converge if minima are not situated at infinity, provided that the activation functions are differentiable. The value of \( \eta \) must not be large enough as well. The speed of convergence depends a lot on \( \eta \), so the choice of this value is a critical step in training the network. As
for the stopping criterion, we usually fix a number of iterations or set a minimum value for the $E$ function to reach.

**Accelerating the Gradient Descent**

The algorithm often takes a long time to converge, and as we are somewhat limited in choice for the value of $\eta$, smaller values make the convergence slower, and larger values lead to divergence. This aggravates with big datasets, so some techniques have been developed to accelerate this process. One of them inserts a "momentum term", $\alpha$. The equation for the weight update is written as follows

$$\vec{w}^{n+1} = \vec{w}^n - \eta \nabla E + \alpha \Delta \vec{w}^n$$ \hspace{1cm} (5.10)

Where $0 \leq \alpha \leq 1$. This term accelerates the convergence by making successive weight updates similar to one another. This is particularly useful when the error function has ravines (although most of the times we do not know the behaviour of the error function). The most used values for $\alpha$ are between 0.5 a 0.95, using $\alpha > 0.95$ tends to cause divergence.

**5.4.2 Levenberg-Marquard Algorithm**

Another used algorithm for training the networks is the Levenberg-Marquardt algorithm. This is an approach to second order training methods that do not require the computation of the Hessian matrix. This is accomplished if the error function has the form of the sum of squares, and so the Hessian matrix

---

Figure 5.5: Example of a multilayer perceptron and of the corresponding backpropagation network. a) Multilayer perceptron. b) Backpropagation network. Taken from [42].
can be approximated by \( H = J^T J \), and the gradient is given by \( g = J^T e \), being \( e \) the error vector and \( J \) the Jacobian matrix containing the derivatives of the network errors with respect to weights and biases as was already seen in section 5.4.1. The update formula for the Levenberg-Marquardt algorithm is given by

\[
\vec{w}^{n+1} = \vec{w}^n - (J^T J + \mu J)^{-1} J^T e
\]  

(5.11)

where \( \mu \) is a positive coefficient that guarantees the inversion of the matrix \( (J^T J + \mu J) \). This method outperforms the gradient descent in speed for smaller networks (less than 10000 parameters), although requiring more memory than the gradient descent. For larger networks, with more than 10000 parameters, the matrix inversion becomes computationally expensive and so gradient descent is a preferred method [46, 47].

5.4.3 Bayesian Regularization of Neural Network

The bayesian regularization framework for neural networks states that instead of looking for a set of weights that minimize an objective function, we make use of the probability distribution of the weights, hence making the output of the network also a probability distribution. One of the objective functions used for this case is

\[
F = E_D(D|w, M) = \frac{1}{N} \sum_{i=1}^{n} (o_i - d_i)^2
\]  

(5.12)

Where \( M \) is a neural network architecture, \( D \) the set, and \( E_D \) the sum of squares of the network error. A common approach to avoid overfitting is adding a regularization term like

\[
F = \beta E_D(D|w, M) + \alpha E_W(w|M)
\]  

(5.13)

Where \( E_W = \frac{1}{n} \sum_{i=1}^{n} w_i^2 \), that is the sum of the networks’ weights. The values of \( \alpha \) and \( \beta \) are estimated in the training process.

This framework uses the assumption that our data has Gaussian noise and so it can be written

\[
P(w|D, \alpha, \beta, M) = \frac{P(D|w, \beta, M)P(w|\alpha, M)}{P(D|\alpha, \beta, M)}
\]  

(5.14)

where \( P(w|\alpha, M) = \left( \frac{1}{2\alpha} \right)^{m/2} \exp\left( -\frac{\beta}{\alpha} w^T w \right) \) Now consider this joint posterior density

\[
P(\alpha, \beta|D, M) = \frac{P(D|\alpha, \beta, M)P(\alpha, \beta|M)}{P(D|M)}
\]  

(5.15)

The term \( P(D|\alpha, \beta, M) \) can be written as:

\[
\frac{P(D|w, \beta, M)P(w|\alpha, M)}{P(w|D, \alpha, \beta, M)} = \frac{Z_F(\alpha, \beta)}{\frac{n}{\beta^{n/2}} \frac{m}{\alpha^{m/2}}}
\]  

(5.16)

Being \( n \) and \( m \) the training set size and the total number of parameters, respectively The equation 5.16,
with a Laplacian approximation (see [48] for mathematical details) produces the following:

\[ Z_F(\alpha, \beta) \propto |H^{MAP}|^{-1/2} \exp(-F(w^{MAP})) \]  

(5.17)

Here, \( MAP \) stands for \textit{maximum a posteriori}. For further clarification on what a posterior probability is, take a look at section 5.5. \( H \) is the Hessian, which we already saw its structure. Regarding the weight updates, this framework follows the exact same method as the Levenberg Marquardt algorithm. An implementation of Neural Networks, along with the described functions and many other functionalities is available as well in MATLAB, which we used during the development of this work.

### 5.4.4 Training the network

To build the training set for the neural network, we created feature vectors (using features described in 5.3 from each example movement taken from the sleep movement database, and label it accordingly to which movement it represents. This same approach is done for SVMs and the Naive Bayes classifier. The implementation/training/testing of the neural network was done with MATLAB’s Neural Network Toolbox [44]. This is a very flexible toolbox as it allows to control several parameters of the network. For all the tests we made, we designed a simple 3 layer (input-hidden-output) network (as in figure 5.4), while varying the number of hidden units and training function. The hyperbolic tangent was used as activation function for all units. First, we removed 20% of the dataset, that would be used for testing. From the remaining dataset, we split it on an 80/20 proportion. 80% was used to train the network, while the remaining 20 was used for validation. The partition of the dataset was done randomly. The use of a validation set is an important method, as it allows to stop training the network early, moreover, it prevents the network from overfitting. To do this, one needs to keep track of both training set and validation set error. As the network’s weights are adjust on every iteration, the training set error decreases. This should happen as well for the validation set, if it does not, then the training process is reset as continuing to train, will overfit the parameters to the training set. The toolbox has a parameter that stops the training of the network after the reset has occurred a specific number of times - which can be set manually - in the current case it was set to 20. The validation set should be completely independent and non-related with the test set, as the objective of training the network is to generalize it as better as possible. By generalizing we mean that if the network is trained with a set \( S \), then it should classify correctly any other training example that is completely independent of \( S \). After training we would test the network, using the first 20% that were removed from the dataset. This was done to ensure the test set was never seen by the network during the training process. An example of how the performance varied with different partitions of the dataset is in figure 5.7

### 5.4.5 Preventing overfit

To the process described in section 5.4.4, we will call it the training process. The network training was done 100 iterations for each configuration (with the same network structure). For each iteration of the
training the process, the dataset was split differently.

Figure 5.6: Screenshot of the neural network toolbox training process [44].

![Neural Network Toolbox](image)

**Algorithms**
- Data Division: Random (divided and)
- Training: Bayesian Regularization (trainbr)
- Performance: Mean Squared Error (mse)
- Calculations: MATLAB

**Progress**
- Epochs: 0, 27, 1000
- Time: 00:00:01
- Performance: 0.0134
- Gradient: 0.0103
- Mar: 0.00050
- Effective F Param: 189
- Sum Squared Param: 85.6
- Validation Checks: 0

**Plots**
- Performance (plotperform)
- Training State (plottrainstate)
- Error Histogram (ploterrhist)
- Confusion (plotconfusion)
- Receiver Operating Characteristic (plotroc)

Training neural network...

Figure 5.7: Plot of each network's performance over 100 iterations, using 3 classes. Vertical axis represents the average accuracy.

We implemented a procedure to assess if the classifier was indeed overfitting. This procedure consisted in permuting randomly the labels assigned to each feature vector and apply the training process.
with no further alterations. The labels’ permutation kept the prevalence of each class. The assumption of this procedure is as follows. Permuting the labels randomly and training will lead to a classifier whose results are meaningless in a real scenario. Hence the probability for classifying correctly a feature vector should be \(1/K\), where \(K\) is the number of classes of the trained model, if the classifier is not overfitting. The same process was applied for the Naive Bayes and Support Vector Machine classifiers. This process is similar to the one illustrated in figure 5.10, except that before splitting the dataset we first permute the labels associated to each feature vector on the dataset, and on the last step we are not choosing the best performing model, but just computing the average classification accuracy. These results are reported in annex A.

5.4.6 Classification Using a Neural Network

Neural networks were used in the past for pattern recognition, namely for gesture recognition in [49, 50, 51, 52]. For the classification with a neural network, first, the network had to be trained using the process already described above. As there were multiple iterations during the training process, and from each iteration, the output was a neural network, the chosen network was the one that performed better amongst all iterations. Then the accelerometer signal retrieved from the wristband or the smartwatch would be analyzed using a sliding time window. For each window, the transition detector would classify it as movement or not, and if the classification was positive the feature vector of the corresponding window was computed and would serve as input to the neural network. Here the neural network would output the corresponding position or movement. The output of the network was used to build a classification vector that was compared to the original position signal retrieved from the PSG or the video analysis. The training process is illustrated in figure 5.10

5.5 Naive Bayes Classifier

This is a probabilistic classifier based on Bayes’s theorem with strong independence assumptions between features. It yields the lowest possible expected error for a given situation. As a brief introduction, the reader can assume that we have a classification problem in which the joint probability distribution of the data and the classes is known. More specifically, assume that we have patterns \(x\) which belong with certain probabilities to the classes \(C_1, C_2...C_n\), and that the joint density \(p(C, x)\) is known. From this distribution, we can compute the probability that an observed pattern \(x\) belongs to a given class \(C\), this probability is given by \(P(C|x)\). This is called the posterior class probability because it is the probability we assign to the class \(C\) after observing \(x\). The prior probability \(P(C)\) is the probability of assigning \(x\) to class \(C\) before observing it. Given a pattern \(x\) to be classified, it is natural to think of assigning it to the class to which is has the highest probability of belonging i.e to the class \(C\) for which \(P(C|x)\) is highest [53]. Let the reader now look at the mathematical details behind this classifier. Assume that an instance to be classified is represented by its feature vector \(x = (x_1, x_2, ...x_n)\), and the
probability of this vector belonging to a class $C_k$ represented by

$$p(C_k|x_1, x_2...x_n)$$ (5.18)

with $K$ possible outcome classes. Using Bayes theorem, this conditional probability can be decomposed as

$$p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)}$$ (5.19)

As $p(x)$ does not depend on $x$, we can write the classification rule

$$f(x) \in \arg \max_C P(x|C)P(C)$$ (5.20)

As we consider all features independent this can be written as

$$f(x) \in \arg \max_C P(C) \prod_i P(x_i|C)$$ (5.21)

This assumption of the features being independent is far from true in most practical situations, however, good classifiers have been developed so far. We assume that $P(C_k)$ is the same for all classes, hence this term can be dropped. The estimation of $P(x_k|C)$ is performed in the training phase of the algorithm [53, 54].

### 5.5.1 Classification Using a Naive Bayes Classifier

The classification using a Naive Bayes classifier is similar to one with Neural Networks. The training phase would occur in multiple iterations and we would pick classifier that performed better amongst all iterations. This is illustrated in figure 5.10. The implementation of the model was done with MATLAB Statistics and Machine Learning Toolbox [55].

### 5.6 Support Vector Machines

The presentation made to Support Vector Machines is based on [56] but it is very superficial as it was not heavily used in the development of the algorithm. Support vector machines are a classifier, that in its simplest form assume that the data points are linearly separable, that is, if we plot the data we can draw a line separating the classes, in higher dimensional cases we draw a hyperplane. Take a look at the figure 5.8 for more clarification. The hyperplane can be described by

$$\vec{w}.\vec{x} + b = 0$$ (5.22)

where $\vec{w}$ is a vector normal to the hyperplane and $\frac{b}{||w||}$ is the perpendicular distance from the hyper-
plane to the origin. If we have two classes the classification rule is as follows

\[ x.w + b \geq 1 \quad \text{for class 1} \]  \( (5.23) \)

\[ x.w + b \leq -1 \quad \text{for class 2} \]  \( (5.24) \)

According to Figure, H1 and H2 are the points closest to the separating plane. Defining \( d_1 \) and \( d_2 \) as the distances from each points to the plane respectively. This distance is called margin. The goal of an SVM is to place the plane in a way that it maximizes the margin, therefore being called a large margin classifier. Finding the highest margin is the equivalent to

\[
\min ||w|| \quad \text{such that} \quad y_i(x_i.w + b) - 1 \geq 0 \quad \forall i \]  \( (5.25) \)

This is the same as minimizing \( \frac{1}{2}||w||^2 \) which is achieved with the use of quadratic programming, which will not be addressed in the work.

For data that is not linearly separable, the trick is to map it to a higher-order dimension according to a kernel function. As an example, the reader can consider the following scenario in figure 5.9. In the two-dimensional space, it is impossible to linearly separate the data, using a plane, as was shown in 5.8. Using a Gaussian Kernel, as shown in equation 5.26, maps the data points into a three-dimensional space, henceforth the data can now be separated by means of a plane. The higher dimensional space onto which the data points were mapped is often called the feature space. For further clarification on using the kernel trick to map the data into a higher dimensional space, or on the computation of the separating margin, we suggest reading [56]. Some of the most used kernel functions are

\[
k(x_i, x_j) = e^{-\frac{||x_i-x_j||^2}{\sigma^2}} \]  \( (5.26) \)

\[
k(x_i, x_j) = (x_i . x_j + a)^b \]  \( (5.27) \)
Figure 5.9: Dichotomous data re-maped used Radial Basis Kernel, taken from [56].

\[ k(x_i, x_j) = \tanh(ax_i x_j - b) \]  \hspace{1cm} (5.28)

In this work, the function used was the Gaussian Kernel or Radial Basis Function (RBF), available on MATLAB’s SVM toolbox [55]

5.6.1 Classification Using a Support Vector Machine

The classification using a SVM is similar to one with Neural Networks. The training phase would occur in multiple iterations and we would pick the SVM that performed better amongst all iterations. This is illustrated in figure 5.10.

5.7 Markov Models

Markov models are stochastic models used in the modelization of randomly changing systems where it is assumed that the future states depend only on the current state, and not on past events. The following presentation of the Markov Model problems and their solution is done according to Rabiner et. al in [57]. Consider a discrete-time system that can be in \( N \) possible states, where the state at time \( t \) is denoted by \( q_t \). The system undergoes a change of state, according to a set of probabilities (it can also go back to the same state). In a first-order Markov chain, the probabilistic description of our system depends on the current state and its predecessor. We define the state transition probabilities as the coefficients \( t_{ij} \), which form a matrix that we will denote by \( T \).
Because the state transition coefficients obey standard stochastic constraints have the following properties

\[ t_{ij} \geq 0 \quad (5.30) \]

\[ \sum_{j=1}^{N} t_{ij} = 1 \quad (5.31) \]

An HMM is characterized by the following:

1) \( N \), the number of states in the model. The states are connected in a way that any state can be reached from any other state (e.g. an ergodic model). The individual states are denoted by \( S = \{ S_1, S_2, \ldots, S_N \} \).

2) \( M \), the number of distinct observation symbols per state, in our case the possible values that the accelerometer can output. The symbols are denoted by \( V = \{ v_1, v_2, \ldots, v_M \} \).

3) The state transition probability distribution \( T = \{ t_{ij} \} \).

4) The observation symbol probability distribution in state \( j \), \( E = \{ e_j(k) \} \) where

\[ e_j(k) = P[v_{k|t} = S_j], \quad 1 \leq j \leq N, 1 \leq k \leq M \quad (5.32) \]
5) The initial state distribution (i.e. the probability for the system to start in each state) $\pi = \pi \{ \pi_i \}$ where

$$\pi_i = P[q_1 = S_i], \quad 1 \leq i \leq N$$  \hspace{1cm} (5.33)

Given appropriate values of $N, M, T, E$ and $\pi$, the HMM can be used as a generator to give an observation sequence $O$

$$O = \text{O}_1 \text{O}_2 \ldots \text{O}_T$$ \hspace{1cm} (5.34)

And so our model can be defined in a compact notation by

$$\lambda = (T, E, \pi)$$ \hspace{1cm} (5.35)

### 5.7.1 The three basic problems for Hidden Markov Models

Given how we defined an HMM in the previous section, there are three basic problems that must be solved for the model to be useful in real-world applications. These problems are the following.

- **Problem 1**: Given the observation sequence $O = \text{O}_1 \text{O}_2 \ldots \text{O}_T$ and a model $\lambda = (T, E, \pi)$, how do we efficiently compute $P(O|\lambda)$.

- **Problem 2**: Given the observation sequence $O = \text{O}_1, \text{O}_2, \ldots, \text{O}_T$, and the model $\lambda$, how do we choose a corresponding state sequence $Q = q_1 q_2 \ldots q_T$ which is optimal in some meaningful sense (i.e., best "explains the observations")?

- **Problem 3**: How do we adjust the model parameters $\lambda = (T, E, \pi)$ to maximize $P(O|\lambda)$?

We will not cover the solution to problem 1, as it has no interest in the implemented solution. For a detailed solution of this problem, take a look at [57].

#### Solution to Problem 2

This problem can be formulated as: Given the observation sequence $O = \text{O}_1, \text{O}_2, \ldots$, and the model $\lambda$, how do we choose a corresponding state sequence $Q = q_1, q_2, \ldots$ which is optimal in some meaningful sense. In the context of our work, this could be seen as the following problem. Consider that your observations are the values outputted from the accelerometer, and the state sequence is the corresponding position, so what is the best state sequence that according to the accelerometer values (e.g. some values outputted by the accelerometer are more likely to be seen in one position instead of another)?

The solution to this problem relies on defining an optimality criterion. A possible criterion is as follows

$$\gamma_t(i) = P(q_t = S_i|O, \lambda)$$ \hspace{1cm} (5.36)

This is the probability of being in state $S_i$ at time $t$, given the observation sequence $O$, and the model $\lambda$. We also introduce the definition of forward and backward variable in the context of HMMs, as is
necessary for writing the upcoming equations. A forward variable can be seen as the probability of a partial observation sequence, until time $t$, and state $S_i$, given the model $\lambda$. Therefore it is defined as:

$$\alpha_T(i) = P(O_1, O_2, ... O_T, q_T = S_i | \lambda)$$  \hspace{1cm} (5.37)$$

For the backward variable, it can be thought of as the probability of the partial observation sequence from $t + 1$ to the end, given state $S_i$ at time $t$ and model $\lambda$, it is defined as:

$$\beta_t(i) = P(O_{t+1}, O_{t+2} ... O_T | q_t = S_i | \lambda)$$  \hspace{1cm} (5.38)$$

Equation 5.36 can be expressed in terms of backwards and forwards variables.

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i)$$  \hspace{1cm} (5.39)$$

$\alpha_t(i)$ accounts for the observations up until time $t$, whereas $\beta_t(i)$ accounts for the remainder of the sequence (from time $t + 1$ until $N$). This can be seen as a probability due to the summation on the denominator, as it is a normalization factor. To find the most likely state we can solve for

$$q_t = \arg\max [\gamma_t(i)] \hspace{0.5cm} 1 \leq t \leq T \hspace{0.5cm} 1 \leq i \leq N$$  \hspace{1cm} (5.40)$$

This is achieved by using the Viterbi algorithm [58, 59]. First we define the quantity

$$\delta_t(i) = \max P[q_1q_2...q_t = i, O_1, O_2, ... O_t | \lambda], \hspace{0.5cm} q_1, q_2, ... qt - 1$$  \hspace{1cm} (5.41)$$

i.e $\delta_t(i)$ is the best score (highest probability) along a single path, at time $t$, which accounts for the first $t$ observations and ends in state $S_i$. By induction we have

$$\delta_{t+1}(j) = \max[\delta_t(i)t_{ij}]c_j(O_{t+1})$$  \hspace{1cm} (5.42)$$

To retrieve the best state sequence, we keep track of the argument that maximized the previous equation and store it in an array $\psi_t(i)$ The complete procedure is described below

1. Initialization

$$\delta_1(i) = \pi_i b_i O_1 \hspace{0.5cm} 1 \leq i \leq N$$  \hspace{1cm} (5.43)$$

$$\psi_1(i) = 0$$  \hspace{1cm} (5.44)$$

2. Recursion

$$\delta_t(j) = \max[\delta_{t-1}(i)t_{ij}]c_j(O_t) \hspace{0.5cm} 2 \leq t \leq T, 1 \leq j \leq N$$  \hspace{1cm} (5.45)$$

$$\psi_t(j) = \arg\max[\delta_{t-1}t_{ij}] \hspace{0.5cm} 2 \leq t \leq T, 1 \leq j \leq N$$  \hspace{1cm} (5.46)$$

3. Termination

$$P^* = \max[\delta_T(i)] \hspace{0.5cm} 1 \leq i \leq N$$  \hspace{1cm} (5.47)$$
4. State sequence backtracking

\[ q^*_t = \psi_{t+1}(q^*_{t+1}) \quad t = T - 1, T - 2, \ldots 1 \]  

(5.49)

Although these formulae may seem complicated, we suggest taking a look at a trellis structure implementation of this algorithm and see how simple the computations are.

Solution to Problem 3

This problem has no known analytical solution, to find the model which maximizes the probability of the observation sequence, so some iterative methods have been developed. The one used in this work is the Baum-Welch algorithm. In order to explain how this algorithm works we define a quantity \( \xi_t \), the probability of being in state \( S_i \), at time \( t \), and in state \( S_j \) at time \( t+1 \), given the model and observation sequence i.e.

\[ \xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \]  

(5.50)

Taking into account the definition of forward and backward variables from the previous problem, \( \xi_t \) can be rewritten as

\[ \xi_t(i,j) = \frac{\alpha_t(i,j) \beta_{t+1}(j)}{P(O | \lambda)} \]  

(5.51)

It is also defined \( \gamma_t(i) \)

\[ \gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \]  

(5.52)

The summation of \( \gamma \) over time can be interpreted as the expected number of times that the state \( S_i \) is visited, or equivalently, the expected number of transitions made from state \( S_i \). Similarly, summing \( \xi_t \) from time \( t = 1 \) to time \( t = T - 1 \), can be interpreted as the expected number of transitions from state \( S_i \) to state \( S_j \). Hence we can define the following re-estimation formulae.

\[ \pi_i = \gamma_1(i) \]  

(5.53)

\[ \pi_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \]  

(5.54)

\[ c_j(k) = \frac{\sum_{t=1}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}, \text{ s.t } O_t = v_k \]  

(5.55)

If the current model is defined as \( \lambda = (T, E, \pi) \), and the re-estimated model as \( \bar{\lambda} = (\bar{T}, \bar{E}, \bar{\pi}) \), then it can be shown that either we are at a critical point of the likelihood function, in which case \( \bar{\lambda} = \lambda \), or that our new model \( \bar{\lambda} \) is more likely than \( \lambda \), in the sense that \( P(O | \bar{\lambda}) > P(O | \lambda) \). The final result of this procedure is called a maximum likelihood estimate of the HMM [60, 61]. The re-estimation formulas can be derived
directly by maximizing (using standard constrained optimization techniques), this auxiliary function

\[ Q(\lambda, \bar{\lambda}) = \sum_Q P(Q|O,\lambda) \log[P(O,Q|\bar{\lambda})] \]  

(5.56)

Maximizing this function over \( \lambda \) leads to increased likelihood, and eventually it converges to a critical point. A quick note on the re-estimation procedure is that the sum each parameter \( \pi_i, t_{ij}, e_j \), must equal to 1.

This re-estimation is done using only a single training pattern, but it cannot be done as we will capture just a small number of observations for any state, hence we need to have multiple training patterns. The modification of the re-estimation procedure is straightforward and goes as follows. We denote the set of \( K \) observation sequences as

\[ O = [O^{(1)}, O^{(2)} \ldots O^{(K)}] \]  

(5.57)

Where \( O^k = [O^{(k)}_1, O^{(k)}_2 \ldots O^{(k)}_{T_k}] \), is the \( k \)-th training pattern. Again our goal is to maximize

\[ P(O|\lambda) = \prod_{k=1}^{K} P(O^{(k)}|\lambda) = \prod_{k=1}^{K} P_k \]  

(5.58)

Finally our modified re-estimation formulas are

\[ \hat{t}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_{t+1}^k t_{ij} e_j O^{(k)}_{t+1} \beta_t^k(i)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k \beta_t^k(i)} \]  

(5.59)

\[ \hat{\tau}_{ij}(l) = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k \beta_t^k(i)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k \beta_t^k(i)} \]  

(5.60)

With the summation over \( t \) in the numerator being such that \( O_t = v_l \). The solution to each of the described problems is already implemented in MATLAB’s Hidden Markov Model toolbox [62], which was used during the development of this work.

### 5.7.2 Classification Using a Markov Model

The classification using a Markov follows a different procedure, from the ones described in the previous chapters. In this case the computation of a feature vector is not done, as the whole accelerometer signal is analyzed using Viterbi’s algorithm. The first step is training a Markov Model using MATLAB’s own function `hmmtrain`, that uses the algorithm described in section 5.7.1. Each signal had to be normalized between 1 and 10, and rounded to the closest integer, as MATLAB’s toolbox only works with integer sequences, we also had to pass as input the corresponding position signal. This was done for each of directions of the accelerometer signal. After training we would obtain the transition and emission \( (t_x, t_y, t_z, e_x, e_y, e_z) \) matrices that define our model. For the analysis of the accelerometer signal, we used MATLAB’s own function `hmmviterbi` that receives as input a model and a vector, and returns the most probable sequence of states, using the algorithm described for the solution to problem 2 in section 5.7.1. Here the sequence of states corresponds the person’s position. As each direction of the accelerometer
was analyzed, we end up with 3 different position signals, which we average to obtain a final position signal, according to figure 5.11.

Figure 5.11: Markov model classification process.
Chapter 6

Results and Discussion

6.1 Problem Description

The main goal of this work is to develop an algorithm that allows detecting the sleeping position. The approach taken was to build a sleep movement database. These movements were recorded in volunteers and patients from Hospital de Santa Maria. The movements were recorded with a Sony Smartwatch 3, where we recorded the values from 4 different sensors, the accelerometer, gyroscope, magnetometer, and rotation vector, and also with an Empatica E4 wristband that only has an accelerometer. The Empatica, however, has better battery life. The original idea was to detect which movement was performed by means of a classifier and from there infer the sleeping position, thus this is a pattern recognition problem.

6.2 Sleep Movement Dataset Results

Our dataset had a total of \( n = 460 \) movements, distributed over the 8 different movement classes. The full distribution of movements can be consulted in section 4.3. Here we used several variations on the structure of the neural network, the training function, number of hidden layers. The same approach was also taken when training a SVM and a Naive Bayes classifier, which results are also reported in this chapter. The results shown in tables 6.1, 6.3, 6.5 are the average accuracy and standard deviation obtained from the training process described in chapter 5.4.4. The features used in these can be consulted in section 5.3 and do not include the gyroscope sensor features. The results in tables 6.2, 6.4, and 6.6, are also resulting from the training process described in chapter 5.4.4 and include the original features plus the correlations between the different axes from the gyroscope.
Table 6.1: 8 Classes, with a simple input-hidden-output neural network structure similar to figure 5.4. The number of units correspond to the hidden layer. Using the same number of examples for all classes \( n=41 \). Without gyroscope features

<table>
<thead>
<tr>
<th>Training Function</th>
<th># Units</th>
<th>Accuracy</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Descent</td>
<td>5</td>
<td>42.8%</td>
<td>6.3%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>47.4%</td>
<td>7.4%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>49.0%</td>
<td>7.4%</td>
</tr>
<tr>
<td>Levenberg-Marquardt</td>
<td>5</td>
<td>48.7%</td>
<td>7.1%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>57.1%</td>
<td>12.6%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>63.4%</td>
<td>14.3%</td>
</tr>
<tr>
<td>Bayesian Regularization</td>
<td>5</td>
<td>54%</td>
<td>6.6%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>59%</td>
<td>7.9%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>60%</td>
<td>8.3%</td>
</tr>
</tbody>
</table>

Table 6.2: 8 Classes, with a simple input-hidden-output neural network structure similar to figure 5.4. The number of units correspond to the hidden layer. Using the same number of examples for all classes \( n=41 \). Using gyroscope features. \( p = 0.1354 \) when compared to the results in table 6.1

<table>
<thead>
<tr>
<th>Training Function</th>
<th># Units</th>
<th>Accuracy</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Descent</td>
<td>5</td>
<td>50.4%</td>
<td>5.6%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>54.7%</td>
<td>6.7%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>57.5%</td>
<td>4.8%</td>
</tr>
<tr>
<td>Levenberg-Marquardt</td>
<td>5</td>
<td>45.3%</td>
<td>6.2%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>59.3%</td>
<td>5.5%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>76.5%</td>
<td>5.4%</td>
</tr>
<tr>
<td>Bayesian Regularization</td>
<td>5</td>
<td>62.7%</td>
<td>7.2%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>64.2%</td>
<td>6.2%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>64.7%</td>
<td>8.6%</td>
</tr>
</tbody>
</table>
Table 6.3: 3 Classes (Supine, Lateral, Prone), with a simple input-hidden-output neural network structure similar to figure 5.4. The number of units correspond to the hidden layer and without gyroscope features, using the same number of examples on all classes n=89

<table>
<thead>
<tr>
<th>Training Function</th>
<th># Units</th>
<th>Average Accuracy</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Descent</td>
<td>5</td>
<td>64.5%</td>
<td>7.1%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>67.4%</td>
<td>7.5%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>69.0%</td>
<td>7.5%</td>
</tr>
<tr>
<td>Levenberg-Marquardt</td>
<td>5</td>
<td>74.7%</td>
<td>8.8%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>75.5%</td>
<td>8.4%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>78.4%</td>
<td>9.7%</td>
</tr>
<tr>
<td>Bayesian Regularization</td>
<td>5</td>
<td>77.4%</td>
<td>6.3%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>75.5%</td>
<td>6.5%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>74.8%</td>
<td>6.4%</td>
</tr>
</tbody>
</table>

Table 6.4: 3 Classes (Supine, Lateral, Prone), with a simple input-hidden-output neural network structure similar to figure 5.4. The number of units correspond to the hidden layer with gyroscope features, using the same number of examples on all classes n=89. $p = 0.2718$ when compared to the results in table 6.3

<table>
<thead>
<tr>
<th>Training Function</th>
<th># Units</th>
<th>Average Accuracy</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Descent</td>
<td>5</td>
<td>68.9%</td>
<td>6.5%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>73.2%</td>
<td>6.2%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>68.3%</td>
<td>6.8%</td>
</tr>
<tr>
<td>Levenberg-Marquardt</td>
<td>5</td>
<td>73.7%</td>
<td>7.9%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>74.6%</td>
<td>10.7%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>85.5%</td>
<td>5.1%</td>
</tr>
<tr>
<td>Bayesian Regularization</td>
<td>5</td>
<td>72.3%</td>
<td>5.7%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>88.0%</td>
<td>7.3%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>82.2%</td>
<td>7.6%</td>
</tr>
</tbody>
</table>
Table 6.5: 2 Classes (Supine, NonSupine), with a simple input-hidden-output neural network structure similar to figure 5.4. The number of units correspond to the hidden layer without gyroscope features, trained with the same number of examples in both classes n=139.

<table>
<thead>
<tr>
<th>Training Function</th>
<th># Units</th>
<th>Average Accuracy</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Descent</td>
<td>5</td>
<td>88.8%</td>
<td>6.5%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>90.6%</td>
<td>6.2%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>90.5%</td>
<td>6.7%</td>
</tr>
<tr>
<td>Levenberg-Marquardt</td>
<td>5</td>
<td>92.4%</td>
<td>5.5%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>93.4%</td>
<td>5.1%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>93.9%</td>
<td>5.1%</td>
</tr>
<tr>
<td>Bayesian Regularization</td>
<td>5</td>
<td>88.3%</td>
<td>5.4%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>87.3%</td>
<td>8.0%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>88.0%</td>
<td>7.7%</td>
</tr>
</tbody>
</table>

Table 6.6: 2 Classes (Supine, NonSupine), with a simple input-hidden-output neural network structure similar to figure 5.4. The number of units correspond to the hidden layer with gyroscope features, trained with the same number of examples in both classes n=139. \( p = 0.9120 \) when compared to the results in table 6.5.

<table>
<thead>
<tr>
<th>Training Function</th>
<th># Units</th>
<th>Average Accuracy</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Descent</td>
<td>5</td>
<td>86.8%</td>
<td>7.4%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>89.0%</td>
<td>7.8%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>88.9%</td>
<td>6.9%</td>
</tr>
<tr>
<td>Levenberg-Marquardt</td>
<td>5</td>
<td>92.5%</td>
<td>8.5%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>96.4%</td>
<td>7.4%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>98.3%</td>
<td>3.6%</td>
</tr>
<tr>
<td>Bayesian Regularization</td>
<td>5</td>
<td>83.6%</td>
<td>10.9%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>89.0%</td>
<td>6.5%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>50.1%</td>
<td>6.1%</td>
</tr>
</tbody>
</table>

Table 6.7: SVM with a Gaussian Kernel, Naive Bayes classifier with a Normal Distribution

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>55.7%</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>53.6%</td>
</tr>
</tbody>
</table>
6.3 Sleep Recording Results

This section reports the results of the algorithm when applied to real night recordings. The real position that the patient was sleeping is reported in its PSG position signal, which we design by ground truth and is represented by a vector. As for the recordings from the Kinect, the position vector was built manually but follows the same structure as the ones from the PSG. The output of the algorithm is also a vector with the same nomenclature as the position signal from the PSG. The method to assess the performance of the algorithm consisted in computing the percentage of time which the position signal and the algorithm output matched each other. Regarding the recordings in Hospital de Santa Maria, we had a total of \( n = 15 \) recordings. Of those 15 only 8 were usable. This happened as the patient had to carry the Empatica wristband and the iPhone all the time to keep the communication going on. If at any time the communication was stopped the wristband would not be able to restart the recording. However, this is not a problem for the Sony Smartwatch, as it stores all the information on the watch’s memory, even if the Bluetooth connection is broken, and it will keep recording as long as its battery does not run out. As there are few recordings we present the results obtained for all of them, using all classifiers mentioned previously. None of the data from these recordings was used on training, validating or testing the classifiers, as well the persons involved in the recordings never participated in the construction of the sleep movement database.
Figure 6.2: Output of the transition detector, spikes represent the places where a transition was detected, the red line represents the position vector from the PSG.

Figure 6.3: Output of the algorithm, using neural networks with no transition detector, with three classes
Table 6.8: ANN with a simple input-hidden-output structure, 5 units, 8 classes as described in chapter 4

<table>
<thead>
<tr>
<th>Recording name</th>
<th>ANN</th>
<th>ANN (No transition detector)</th>
<th>SVM</th>
<th>Naive Bayes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSM1</td>
<td>65.3%</td>
<td>62.2%</td>
<td>66.3%</td>
<td>64.3%</td>
</tr>
<tr>
<td>HSM2</td>
<td>61.7%</td>
<td>60.4%</td>
<td>87.1%</td>
<td>87.8%</td>
</tr>
<tr>
<td>HSM3</td>
<td>63.0%</td>
<td>83.8%</td>
<td>48.8%</td>
<td>64.1%</td>
</tr>
<tr>
<td>HSM4</td>
<td>48.4%</td>
<td>55.8%</td>
<td>29.8%</td>
<td>32.2%</td>
</tr>
<tr>
<td>HSM5</td>
<td>33.4%</td>
<td>8.4%</td>
<td>33.7%</td>
<td>41.5%</td>
</tr>
<tr>
<td>HSM6</td>
<td>38.3%</td>
<td>27.8%</td>
<td>38.5%</td>
<td>37.0%</td>
</tr>
<tr>
<td>HSM7</td>
<td>97.1%</td>
<td>39.1%</td>
<td>99.8%</td>
<td>52.7%</td>
</tr>
<tr>
<td>HSM8</td>
<td>48.9%</td>
<td>55.8%</td>
<td>51.4%</td>
<td>49.9%</td>
</tr>
<tr>
<td>Kinect1</td>
<td>30.5%</td>
<td>46.1%</td>
<td>21.3%</td>
<td>45.2%</td>
</tr>
<tr>
<td>Kinect2</td>
<td>46.1%</td>
<td>5.6%</td>
<td>27.2%</td>
<td>79.1%</td>
</tr>
<tr>
<td>Kinect3</td>
<td>64.4%</td>
<td>35.2%</td>
<td>16.5%</td>
<td>16.3%</td>
</tr>
<tr>
<td>Kinect4</td>
<td>55.3%</td>
<td>42.4%</td>
<td>55.9%</td>
<td>48.1%</td>
</tr>
<tr>
<td>Average</td>
<td>46.3±18.8%</td>
<td>43.5±22.5%</td>
<td>39.7±23.7%</td>
<td>51.5±19.9%</td>
</tr>
</tbody>
</table>

Table 6.9: ANN with a simple input-hidden-output structure, 5 units, 3 classes (Supine, Lateral, Prone)

<table>
<thead>
<tr>
<th>Recording name</th>
<th>ANN</th>
<th>ANN (No transition detector)</th>
<th>SVM</th>
<th>Naive Bayes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSM1</td>
<td>12.3%</td>
<td>19.2%</td>
<td>47.1%</td>
<td>64.0%</td>
</tr>
<tr>
<td>HSM2</td>
<td>23.7%</td>
<td>30.7%</td>
<td>66.2%</td>
<td>67.1%</td>
</tr>
<tr>
<td>HSM3</td>
<td>51.4%</td>
<td>58.9%</td>
<td>53.5%</td>
<td>32.7%</td>
</tr>
<tr>
<td>HSM4</td>
<td>30.3%</td>
<td>11.0%</td>
<td>21.2%</td>
<td>28.2%</td>
</tr>
<tr>
<td>HSM5</td>
<td>44.5%</td>
<td>66.0%</td>
<td>47.3%</td>
<td>29.4%</td>
</tr>
<tr>
<td>HSM6</td>
<td>38.9%</td>
<td>34.3%</td>
<td>38.4%</td>
<td>37.1%</td>
</tr>
<tr>
<td>HSM7</td>
<td>52.6%</td>
<td>53.9%</td>
<td>52.7%</td>
<td>52.7%</td>
</tr>
<tr>
<td>HSM8</td>
<td>50.4%</td>
<td>36.1%</td>
<td>37.7%</td>
<td>47.1%</td>
</tr>
<tr>
<td>Kinect1</td>
<td>57.6%</td>
<td>54.0%</td>
<td>17.7%</td>
<td>59.0%</td>
</tr>
<tr>
<td>Kinect2</td>
<td>92.5%</td>
<td>94.3%</td>
<td>73.8%</td>
<td>25.3%</td>
</tr>
<tr>
<td>Kinect3</td>
<td>91.9%</td>
<td>64.8%</td>
<td>67.2%</td>
<td>29.2%</td>
</tr>
<tr>
<td>Kinect4</td>
<td>92.4%</td>
<td>55.9%</td>
<td>89.2%</td>
<td>62.5%</td>
</tr>
<tr>
<td>Average</td>
<td>53.2%±26.9%</td>
<td>48.2±23.0%</td>
<td>51±20.9%</td>
<td>44.5±15.9%</td>
</tr>
</tbody>
</table>
Table 6.10: ANN with a simple input-hidden-output structure, 5 units, 2 classes (Supine vs Not Supine)

<table>
<thead>
<tr>
<th>Recording name</th>
<th>ANN</th>
<th>ANN (No transition detector)</th>
<th>SVM</th>
<th>Naive Bayes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSM1</td>
<td>90.0%</td>
<td>90.1%</td>
<td>5.4%</td>
<td>3.6%</td>
</tr>
<tr>
<td>HSM2</td>
<td>71.4%</td>
<td>71.5%</td>
<td>26.6%</td>
<td>26.5%</td>
</tr>
<tr>
<td>HSM3</td>
<td>63.3%</td>
<td>82.2%</td>
<td>17.5%</td>
<td>17.3%</td>
</tr>
<tr>
<td>HSM4</td>
<td>42.9%</td>
<td>79.1%</td>
<td>17.3%</td>
<td>19.5%</td>
</tr>
<tr>
<td>HSM5</td>
<td>1.4%</td>
<td>1.4%</td>
<td>69.4%</td>
<td>46.0%</td>
</tr>
<tr>
<td>HSM6</td>
<td>35.8%</td>
<td>35.8%</td>
<td>53.1%</td>
<td>46.9%</td>
</tr>
<tr>
<td>HSM7</td>
<td>97.9%</td>
<td>99.8%</td>
<td>78.2%</td>
<td>52.6%</td>
</tr>
<tr>
<td>HSM8</td>
<td>67.9%</td>
<td>67.9%</td>
<td>11.3%</td>
<td>11.7%</td>
</tr>
<tr>
<td>Kinect1</td>
<td>53.9%</td>
<td>53.9%</td>
<td>13.5%</td>
<td>41.1%</td>
</tr>
<tr>
<td>Kinect2</td>
<td>94.3%</td>
<td>94.3%</td>
<td>26.5%</td>
<td>5.6%</td>
</tr>
<tr>
<td>Kinect3</td>
<td>64.7%</td>
<td>64.7%</td>
<td>15.7%</td>
<td>28.9%</td>
</tr>
<tr>
<td>Kinect4</td>
<td>55.8%</td>
<td>55.8%</td>
<td>41.5%</td>
<td>41.8%</td>
</tr>
<tr>
<td>Average</td>
<td>53.4±29.9%</td>
<td>66.3±27.5%</td>
<td>29.6±25.0%</td>
<td>28.4±17.0</td>
</tr>
</tbody>
</table>

Table 6.11: Markov Model results

<table>
<thead>
<tr>
<th>Recording name</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSM1</td>
<td>54.0%</td>
</tr>
<tr>
<td>HSM2</td>
<td>17.4%</td>
</tr>
<tr>
<td>HSM3</td>
<td>47.8%</td>
</tr>
<tr>
<td>HSM4</td>
<td>52.7%</td>
</tr>
<tr>
<td>HSM5</td>
<td>11.0%</td>
</tr>
<tr>
<td>HSM6</td>
<td>16.8%</td>
</tr>
<tr>
<td>HSM7</td>
<td>2.5%</td>
</tr>
<tr>
<td>HSM8</td>
<td>40.4%</td>
</tr>
<tr>
<td>Kinect1</td>
<td>45.4%</td>
</tr>
<tr>
<td>Kinect2</td>
<td>75.5%</td>
</tr>
<tr>
<td>Kinect3</td>
<td>35.3%</td>
</tr>
<tr>
<td>Kinect4</td>
<td>29.2%</td>
</tr>
<tr>
<td>Average</td>
<td>35.6±21.1%</td>
</tr>
</tbody>
</table>

6.4 Discussion

The initial idea for this work was to train a sleep movement classifier, and from the movement classification it would be possible to infer the position. This implied building an eight class classifier, which may require more data than the one used to train it in order to generalize correctly. To simplify this process
we decided to merge the 8 classes into 3, aggregating the classes based on the person’s final position. A more detailed explanation was done in chapter 4.

As we have not found any other work that uses an accelerometer on the wrist to detect the sleeping position, no comparison with existing can be done at this time.

Regarding the sleep movement database, acquired with the Sony Smartwatch 3, the values from the magnetometer were not used as they can be biased if the patient is near any electronic device, which we are not able to control in a real life situation. The values from the rotation vector were not used as well because its sampling frequency is not constant during the whole recording.

For all the training procedures we kept the number of examples in each class equal, to avoid biases towards any class. This was done by picking randomly examples from the classes that had more movements on each training iteration. As an example, when training a network with 3 classes, each class had 89 examples (as in table 4.2 we have 89 movements ending in prone position), so we picked randomly 89 examples from the Supine class and 89 random examples from the Lateral class. As we have more examples of some classes than others (e.g more movements ending in a lateral position than ending in a prone position), when training the model the examples from the classes which have more examples were picked randomly on all iterations. This also takes into account what is said at the end of section 5.5, where we assumed that $P(C_k)$ is equal for all classes.

As can be seen by looking at the tables 6.1 and 6.3, average accuracy results increased when moving from 8 classes to 3 classes using Neural Networks. When using only 3 classes, regarding the lateral position, we stop knowing whether the person is lying on its left or right side. If the algorithm is to be applied, with 3 classes only, for investigating patients with Gastroesophageal Reflux Disease where the side of which the person is lying is important, then this is a disadvantage, as it is known that sleeping on the left side eases the symptoms of this disorder. Regarding the case of sleep apnea, the side on which the person is lying - whether is left or right - is not important, the apnea’s episodes decrease in frequency just by the person sleeping on its side, so the approach does not pose any problem for this case.

When training the network and testing it, we noticed that its worst performance was with movements that ended in a prone position. When recording these movements we noticed that this kind were the most complex, involving many complex arm movements, whereas movements that ended in a lateral or supine position were simpler. Therefore, capturing features that can describe these movements is a tough problem. Perhaps the features we are using for the describing movements are not the best ones. A possible improvement could be the use of the rotation vector, present on the Smartwatch, based on the idea that it can capture the rotation movement performed when the person lies on its stomach. Figure 6.4 represents the confusion matrix obtained from training the network with 8 classes, and illustrates what we have said before about the misclassification of movements ending in a prone position (see row 6 of figure 6.4). An interesting value is the misclassification of movements prone-left lateral and left-lateral prone on row 5, column 2. Once more, this is according to what has been said so far on the difficulties for the algorithm to classify correctly movements associated with the prone position.
Figure 6.4: Confusion matrix resulting from training the network with 8 classes. Classes 1 to 8 are prone-right lateral, prone-left lateral, supine-right lateral, supine-left lateral, left lateral-prone, right lateral-prone, left lateral-supine, right lateral-supine respectively.

When using 3 classes the same happened for the classes ending in a prone position, as was expected and it is illustrated in figure 6.5.
In this work, we also merged the classes of movements ending in a lateral position and prone position into a single class. This was meant as a workaround to improve the classification for movements ending in a prone position. These results show a significant improvement \((p < 10^{-5})\) when compared with the three classes scenario, however having only two classes once more leads to loss of information.

The increase in the number of units, using the same training function, does not result in a large increase of the testing performance as the network error function is already at local minima. Increasing the number of units further will only increase the number of parameters that the network has to train. The number of parameters for the simple input-hidden-output feedforward architecture is given by:

\[
\#NetworkParameters = \#I \times \#H + \#H \times \#O + \#I + \#H + \#O
\]  

(6.1)

where \(#I, \#H, \#O\) are the size of the input vector, the number of hidden units, and the size of the output vector respectively. The last term in the expression is associated with the bias terms, if we consider one for each unit. One can see that for a simple network with an input of size 10, 5 units in the hidden layer and an output vector of size 8, we need to train 113 parameters. As a rule of thumb, we would say that the number of weights should be around or below one hundreth of the product of the number of training patterns by the number of outputs [42]. This does not happen in our case, as in the example of 3 classes, using a length 5 feature vector, 5 neurons on the hidden layer we get a network with 53 parameters, that is trained with 267 different training examples. This number of training patterns, for
3 classes would require a network with only 8 parameters, according to the rule of thumb. Hence the obtained models may not generalize well. As a possible solution, we strongly suggest increasing the size of the movement database.

The training function that produced the best results during the tests was the Bayesian Regularization. This happens as the training process of ANN with Bayesian Regularization excels in preventing overfitting, by calculating effective network parameters and discarding all those that are not relevant [63]. Moreover, regarding the values in tables A.1, A.2, and A.3 for Bayesian Regularization, they are according to what we have said in section 5.4.5, the classification accuracy should be $1/K$ when $K$ is the number of classes. As our scenarios were for 8, 3, and 2 classes the expected accuracies should be around 12.5%, 33.3% and 50%, which we can say that are.

Including the gyroscope features when using 8 classes did not result in a significant improvement on the classification ($p > 0.05$). The same happened for 3 classes, where no significant changes were detected ($p > 0.05$). As for the classifier with 2 classes, we did not consider the last value corresponding to 15 units in the hidden layer and using Bayesian Regularization as the training function, as we suspect that overfitting might have occurred in that scenario. However, there are still no significant changes in the classification accuracy ($p > 0.05$). Perhaps the correlation between the three axis of the gyroscope is not the most indicated feature in our scenario, so we suggest investigating further in other features to include that might add more information to the model.

Regarding the full night recordings, some of them present very low accuracies, while others are around 100%. For the recordings with extremely high accuracies, these are recordings where the patient did not move very much during the night, namely in the recording HSM7 the patient stayed in the same position during the whole night. Should the classifier fail to detect the correct position in the beginning of the night for cases likes these, and the accuracy will drop to values close to zero, as no transitions are picked up by the transition detector and so our algorithm is not able to classify correctly. This is what happened for example in the recording HSM1 or HSM5 for Support Vector Machines and neural networks respectively. This is a reflection from our chosen method for performance assessment, as we are measuring in percentage the number of samples that the algorithm matches the signal from the PSG. The large values of standard deviation present in all tables are due to this phenomenon.

During all sleep recordings, the prone position was the least common to been seen. This is comprehensible in the Hospital de Santa Maria recordings as the patients have a large device stuck to their chest area, and sleeping on their stomach would be quite uncomfortable. Regarding the recordings in volunteers, this may have happened due to the small size of the dataset. A possible improvement in described in chapter 7.2

As can be seen in figure 6.2, the transition detector detected some transitions where there was no change in position. One of the reasons was the presence of spikes on the accelerometer signal. These spikes often coincide with episodes of periodic limb movements. The spasms show up on the accelerometer signal misleading the algorithm, and detecting a transition where there is none. After this event, the algorithm fails to detect the actual position and stays on an incorrect one until the next transition occurs. As a solution, we propose improving the transition detector, by including a spike
detector that denies the detection of the transition if it finds an isolated spike on the accelerometer signal. A possible solution would be for the algorithm to prevent itself from evaluating periods of great movement, as happens when the patient gets up or sits in bed. This may require the inclusion of another class on the classifier that represents the patient being standing or sitting. As we already mentioned before, the inclusion of the rotation vector sensor may prove useful in this case, as it may capture the orientation of the device which may be unique for a standing or sitting position. Also here could be useful including the frequency related features used in [41]. In chapter 7.2 we propose some more improvements to the transition detector. The patients from Hospital de Santa Maria recordings, would also often wake up and sit in bed, which can be seen on the annotations from the PSG signal. Our algorithm fails to identify these periods and from there onwards the position classification is incorrect until a new transition occurs, leading to low accuracies (this is actually the case of figure 6.2).

As the transition detector was often failing to detect movements, we decided to experiment running the algorithm without it. For this, we introduced an extra class in the network that contained spurious movements, or samples taken from when the person was sleeping and doing no movement. However, it did not lead to better results when we used 2, 3 or 8 classes ($p < 10^{-5}$ on all cases). This happens as the network output failed to distinguish between a spurious movement and a transition and kept changing the position output through time as can be seen in figure 6.3. This could be enhanced by adding more examples to the class of spurious movements. Another possible improvement for the transition detector would be to include a variable that has memory about the time when the last transition was detected. If a transition was detected one or two seconds ago then it should not allow the algorithm to proceed to the classification step and keep analyzing the signal.

Regarding the Bayes classifier it is important to remark that the it is based on the strong assumption that all features are independent. This is certainly not true in our case, and can explain why its classification accuracy is so low in some cases. In what concerns SVMs, the use of a Gaussian kernel may have lead to overfitting the data to the training set, resulting in poor results on the sleep recordings set. A possible solution would be to reduce the number of features used or even try another kernel function like the polynomial or hyperbolic tangent.

As for Markov Models, the average accuracy is lower when compared with the other classifiers. This may have been due to bad combination of weights on each accelerometer signal (we are using 0.33 for each direction). A possible solution would be study around the values that will output better results. Also the normalization procedure, of setting the acceleration values as integers between 1 and 10, leads to a great loss of information on the content of the acceleration signal. This is a disadvantage of MATLAB’s toolbox [62]. In this case a possible solution would be to implement the Markov Model algorithm that do not require the emitted symbols to be represented by integers.

Finally, another drawback from this work and that can be improved in the future, is placing the watch on the right hand. In the beginning of this work, the device’s were placed on the left hand and we do not know if the same results can be generalized if the device is placed on the right hand.
Chapter 7

Final Remarks

7.1 Achievements

In this work, we implemented a sleep position classifier using different classifiers namely Artificial Neural Networks, Support Vector Machines, Naive Bayes Classifiers and Markov Models. The novelty of this work is that it classifies the position according to data from an accelerometer placed on the wrist, which to our knowledge has not been done before. Another novelty in this work is that we developed a classifier for sleep movements, instead of classifying only the position. Two Android applications were developed, one for recording the sleep of volunteers and patients during the night, and the other to record sleep movements in volunteers. The latter allowed building a sleep movement database with over four hundred different examples. We insist on the need to increase the size of this database to broaden more variations of the same movement.

The Android application that was meant to be used during recordings in Hospital de Santa Maria was not used as patients would arrive at the hospital around 6pm and return the next day at 9am. The device’s battery life does not last more than eight hours recording so we opted for using the Empatica E4 wristband instead. In section 7.2 we suggest some other devices that may have better battery life. Smartwatches are still in an early stage of development and were not designed for recording sensor values during such a long period of time.

The process of developing the application took a long time until a production version (by production we mean, ready to use by the technicians at Hospital de Santa Maria) was ready. Until we realized that Doze mode was taking place and lowering the sampling frequency from the sensors, several approaches were tried in order to overcome this problem. Fortunately, an update from Google arrived in April which allowed to use the PowerManager to overcome this hindrance. Unfortunately, however, this was not enough as the device never got to be used by the technicians in Hospital de Santa Maria more than once, due to battery considerations as was already mentioned.

We were not able to compare our results with the literature, as to our knowledge, this is a pioneer work. Other works have developed classifiers for sleep position, using accelerometers, placed on the head, hip or chest, but none of them used a device on the wrist. Although we believe that our method
will never be able to surpass the conventional piezoelectric sensor attached to the chest.

A study in Hospital de Santa Maria was conducted in which patients wore a wristband with an embedded accelerometer during their sleep. These patients often suffered from disorders like sleep apnea or restless leg syndrome. The latter often produces periodic limb movements. These limb movements would be captured by the accelerometer present in the wristband and deteriorate the classifier results.

There is still a problem with the incapability of the used wearable devices to function as independent devices. The patients would often forget the iPhone in one room and due to this the recording would stop even before they went to sleep. This is one the reasons why the size of the Hospital de Santa Maria dataset is so small, and not all of the acquisitions are usable.

Due to reasons beyond our control the recordings at Hospital de Santa Maria only started mid-July while this work start being developed since February.

The results obtained in the training process look very promising, as the classifier achieved accuracies with values above 70% and sometimes close to 100% (always using the same prevalence in all classes). However, this does not mean that our classifier is generalizing well, as the results from the full night recordings do not present, on average, the same accuracy as the training process results. Despite the results being far from perfect, we feel that they are quite encouraging and that have paved the way for the development a better algorithm for classifying sleep position as there is a lot of room for improvement. On the next section, we suggest some improvements that can be made to the algorithm.

A major difficulty of this work was handling all the large video files recorded by the Kinect, which slowed down the acquisition process of data. Despite having a high video quality the Kinect was not designed to record several hours of video, and we were not aware of the necessity of recording the depth camera stream in order to record the infrared one.

Regarding the use of the gyroscope, none of the scenarios we tested (8 classes, 3 classes and 2 classes) proved to be a significant improvement in the classification accuracy. We were not able to use the rotation vector sensor as its sampling frequency is not constant and is dependent on the workload that the device is subjected to. The magnetometer was not used as well, as it output is influenced by the presence of electromagnetic devices.

### 7.2 Future Work

As a first recommendation for future work, we suggest the usage of another smartwatch, which we present hereafter. Samsung’s Gear S2 is a very powerful device with a price similar to Sony Smartwatch 3. This device has the advantage of having a PPG sensor, along with an accelerometer and other sensors. On the other hand to develop any applications for this device, one needs to learn how to program in Tizen, the operating system chosen by Samsung for their devices. The fact the Samsung’s device run on a different operating system was the deciding factor for us using the device by Sony. Another smartwatch will be available by the end of 2016 by Samsung, the Gear S3. This device can also be an interesting alternative to the Sony, as it also features a PPG sensor and accelerometer among other sensors. This device also runs on Tizen operating system. The launch of Android Wear 2.0 will
occur in January 2017. This new version for Android Wear improves continues to run use Android as the operating system and will have improvements relative to the current version that is present on the Sony Smartwatch 3. New smartwatches will be available in the market, although only a few have been announced yet. These devices will have embedded the same sensors as the Sony Smartwatch plus the PPG sensor as well as better battery lifetime. Another strong point for these devices is that they are built to adapt to one’s lifestyle, hence we strongly suggest to develop an application that can classify sleep position recurring only to the sensors present in the smartwatch. An interesting approach for improving this work would be to use the PPG in detecting sleep position. However for this, we recommend reading first the paper by Shinar[11].

During the development of the work we only mentioned smartwatches running on Android, however Apple has its own device as well, the iWatch. This device was not used for the acquisitions due to software constraints. The watch’s operating system stops all the running applications once its screen goes off, which is unfeasible for us as the application needs to keep recording during the entire night and not only when the device’s screen is on. We also felt more comfortable at developing in Java for Android, instead of Objective-C for Apple.

The increase of the movement database might as well improve the results, we suggest developing a similar movement acquisition protocol, acquiring the movements defined in chapter 4, but using a larger sample and with a wider range of ages. The analysis of these movements can provide insights on the next steps to take on improving the classifier, namely on feature selection. A possible improvement in the acquisition of sleep movements in volunteers could be the automatization of the process. By this we mean to generate a random sequence of movements for the volunteer to perform, where the start and of end of each movement would be triggered by a sound on the smartphone.

Another suggestion for future work is the use of another infrared camera, instead of the Kinect. The hardware restrictions imposed by the Kinect, that if one wishes to record only the infrared stream, it must also record the depth stream are unacceptable. The size of the 8-hour recordings is completely unfeasible nowadays as most external hard drives can only store up to a few terabytes, which would be filled with 2 or 3 recordings.

As for improvements in the classifier, we suggest trying out recurrent neural networks, as Google already did for speech recognition. The idea here would be to train the network with the accelerometer signals of night recordings. To train these networks it is also necessary a larger dataset. This may require better hardware as these networks are harder to train. They were not used in this work as they require powerful hardware to handle the heavy computations inherent to them. If one is not obligated to use MATLAB we suggest using Google’s own platform for machine learning, TensorFlow. This is a Python framework specialized in deep learning, that is used in Google’s speech to text application [64] that may prove useful in developing a classifier of this kind. We suggest using another toolbox for Hidden Markov Models, as the one used in this work only allows using integer-valued sequences, and this may lead to great loss information on the content of the signals. Regarding SVM, using another kernel, instead of the Gaussian one, may prove useful and give in to better results. As for the Naive Bayes classifier, the suggestion would be to check for feature dependency first before using them in the
models as to avoid violating the strong assumption on which it lies on. Another workaround would be to train a Bayesian network.

As was mentioned before, the use of the gyroscope did not bring any significant improvement on the average accuracy on the test set. This can be improved by including features from the gyrosopes other than the correlation between each direction. We do not recommend investing time on using the magnetometer for acquiring sleep movements, these values can be biased by the presence of electronic devices. The use of the rotation vector present in the smartwatch could bring an improvement to the classifier. Deep down this sensor represents the orientation of the device, and some of the sleep positions may have unique orientations that can only be captured by the rotation vector. In figure 2.4 we presented an example on how this sensor captured the change in position on the z-direction.

As was done in this work, we merged the eight movement classes according to the final position on which the person would stay after the movement. However it would be interesting to take the opposite approach, in a way that we would be classifying the person’s initial position based on the movement, and from here infer the sleeping position that the person was before the movement.

As a final suggestion, could be interesting to place a device on the patient’s ankle instead of the wrist. This would be done, based on the assumption that there are less degrees of freedom on the ankle than are on the hand, and could be a effective way to detect sleep position.
Bibliography


Appendix A

Chance Level Distribution - Preventing Overfit

The results in tables A.1, A.2, and A.3 are the average accuracies and standard deviations obtained after performing the procedure described in section 5.4.5.

Table A.1: Results from the procedure described chapter 5.4.5, using 8 classes, with a simple input-hidden-output architecture, similar to figure 5.4

<table>
<thead>
<tr>
<th>Training Function</th>
<th>#Units</th>
<th>Accuracy</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Descent</td>
<td>5</td>
<td>19.7%</td>
<td>4.4%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>15.4%</td>
<td>3.9%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>20.7%</td>
<td>4.4%</td>
</tr>
<tr>
<td>Levenberg Marquardt</td>
<td>5</td>
<td>22.8%</td>
<td>5.4%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>19.9%</td>
<td>4.5%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>27.7%</td>
<td>4.9%</td>
</tr>
<tr>
<td>Bayesian Regularization</td>
<td>5</td>
<td>12.5%</td>
<td>3.6%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>12.6%</td>
<td>3.4%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>12.8%</td>
<td>3.3%</td>
</tr>
</tbody>
</table>
Table A.2: Results from the procedure described chapter 5.4.5, using 3 classes, with a simple input-hidden-output architecture, similar to figure 5.4

<table>
<thead>
<tr>
<th>Training Function</th>
<th>#Units</th>
<th>Accuracy</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Descent</td>
<td>5</td>
<td>31.0%</td>
<td>6.9%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>31.2%</td>
<td>6.3%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>31.4%</td>
<td>6.6%</td>
</tr>
<tr>
<td>Levenberg Marquardt</td>
<td>5</td>
<td>31.8%</td>
<td>6.4%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>32.7%</td>
<td>6.9%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>32.8%</td>
<td>6.3%</td>
</tr>
<tr>
<td>Bayesian Regularization</td>
<td>5</td>
<td>32.1%</td>
<td>6.1%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>32.4%</td>
<td>5.8%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>32.6%</td>
<td>6.0%</td>
</tr>
</tbody>
</table>

Table A.3: Results from the procedure described chapter 5.4.5, using 2 classes, with a simple input-hidden-output architecture, similar to figure 5.4

<table>
<thead>
<tr>
<th>Training Function</th>
<th>#Units</th>
<th>Accuracy</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Descent</td>
<td>5</td>
<td>67.4%</td>
<td>4.4%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>68.8%</td>
<td>3.9%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>69.5%</td>
<td>4.4%</td>
</tr>
<tr>
<td>Levenberg Marquardt</td>
<td>5</td>
<td>70.8%</td>
<td>8.0%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>72.4%</td>
<td>8.4%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>76.7%</td>
<td>10.4%</td>
</tr>
<tr>
<td>Bayesian Regularization</td>
<td>5</td>
<td>55.4%</td>
<td>5.5%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>56.1%</td>
<td>5.9%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>55.7%</td>
<td>6.3%</td>
</tr>
</tbody>
</table>