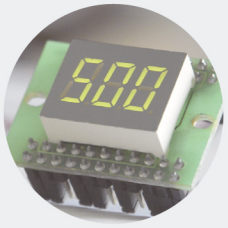
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Reconhecimento de Exercícios Físicos em Tempo-Real em Dispositivos Wearable

TIAGO MIGUEL DA SILVA FERREIRA
Outubro de 2016

POLITÉCNICO
DO PORTO

Real-Time Physical Exercises Recognition on Wearable Devices

Tiago Miguel da Silva Ferreira

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

2016

A dissertation submitted in partial fulfillment of the requirements for the Master degree of Electrical and Computer Engineering

Candidate: Tiago Miguel da Silva Ferreira, Nº 1100498, 1100498@isep.ipp.pt

Scientific Orientation: Paula Maria Marques de Moura Gomes Viana, pmv@isep.ipp.pt

Company: Fraunhofer Portugal Research Association

Supervision: Lourenço Barbosa de Castro, lourenco.castro@fraunhofer.pt

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

21 de outubro de 2016

Acknowledgements

Aos meus pais, minha irmã e minha namorada por todo o apoio e compreensão!

Resumo

Os grandes avanços tecnológicos têm permitido o desenvolvimento de novos equipamentos móveis com elevadas capacidades e que permitem uma utilização não intrusiva e ubíqua com os utilizadores. Os dispositivos *wearable*, têm sido apresentados por diversos fabricantes e tornam-se cada vez mais omnipresentes.

A utilização de sensores permite a monitorização de dados externos e o desenvolvimento de aplicações inteligentes que têm como objetivo facilitar a vida dos utilizadores ou fornecer apoio adicional em áreas tão diversas como a saúde, desporto, ambientes de vida assistida, etc.

No âmbito desta dissertação, desenvolveu-se uma aplicação a ser utilizada num dispositivo *wearable* e que pode ser vista como um personal trainer que valida um conjunto de exercícios propostos numa sessão de desporto.

A solução desenvolvida, usa os sensores inerciais de um *smartwatch* Android Wear, para com base num conjunto de algoritmos de reconhecimento de padrões, detetar a taxa de sucesso na execução de um treino planeado. O facto de todo o processamento ser realizado no próprio dispositivo, é um fator diferenciador para outras soluções existentes.

Palavras-Chave

Android, Smartwatch, Wearable, Reconhecimento de Padrões, Sensores, Exercício Físico.

Abstract

Technological advances have allowed the development of new mobile devices with high capacity and allow non-intrusive and ubiquitous use. Wearable devices have been introduced in the market by several manufacturers and are becoming increasingly ubiquitous.

Sensors allows monitoring external data and developing intelligent applications that aim to make life easier for users and may provide additional support in areas as diverse as health, sports, ambient assisted living, etc.

As part of this work, an application to be used in a wearable device and that can be seen as a personal trainer which validates a set of exercises proposed in a sport session, was developed.

The developed solution uses inertial sensors of an Android Wear smartwatch, and based on a set of pattern recognition algorithms, detects the rate of success in the execution of a planned workout. The fact that all processing can be performed on the device itself, is a differentiator factor to other existing solutions.

Keywords

Android, Smartwatch, Wearable, Pattern Recognition, Sensors, Physical Exercise.

Table of Contents

Index of Figures

Index of Tables

Acronyms

ADB – Android Debug Bridge

API – Application Programming Interface

ART – Android Runtime

BLE – Bluetooth Low-Energy

CPU – Communications Processor Unit

DTW – Dynamic Time Warping

GPS – Global Positioning System

HMM – Hidden Markov Models

IDE – Integrated Development Environment

MEMS – Microelectromechanical Systems

OTA – Over-the-Air

SDK – Software Development Kit

1. INTRODUCTION

The work presented in this document was developed as the Thesis for the Master Course in Electrical and Computer Engineering. It was developed as an internship in Fraunhofer Portugal and the main objective was to create an application for an Android wearable device able to recognized some pre-defined gestures.

The proposal was driven by the fact that, in the last few years, smartphones and the entire mobile platforms have suffered major performance improvements, and technology diversification made them reachable to everyone. The development of applications that make use of the abilities of these devices has been growing and used in the very different tasks, boosted by the integration of sensors that enable making the solutions smarter and more reactive.

The applications of this sensorial ability, integrated in recent wearable devices, made gesture recognition possible in an almost natural way maintaining for example a smartwatch tight to his wrist. The sensors used in this work belong to the inertial group and are used to obtain acceleration, rotation and magnetic field values.

1.1. MOTIVATION

This project was setup from the objective of doing a deeper investigation in an innovative field like the wearable technology is, and in that way, getting knowledge in the field that, surely, will be one of the most explored in the future.

Due to the recent nature of this technology, the use of wearable devices is still emerging in the market, with the creation of the smartwatches, smart glasses and smart bands. The integration of this technology within the surrounding world is being also supported by the concept of the Internet of Things that is expected to contribute to a ubiquitous integration and interaction of people and things.

1.2. OBJECTIVES

The main goal of this dissertation is the research and development of a solution capable of understanding and recognizing sensor data provided from a smartwatch and, using that same data, to classify gestures using algorithms with pattern recognition abilities. The final solution of such an application could be a device used for observing the quality of some sequence of executed exercises. Different application areas can be identified, including sports and healthcare.

Due to the inherent complexity of this objective, smaller sequential tasks were defined:

- Analyze the state of the art in Pattern Recognition Algorithms;
- Analyze the state of the art in the supporting technology: Inertial Sensors and Sensor Fusion approaches, Android OS in wearable platforms;
- Development of a gesture recognizer system under an Android platform;
- Development of real-time functionalities and other optimizations;

1.3. REPORT ORGANIZATION

Chapter 1 presents an introduction to the theme of this dissertation, including a contextualization, as well as its main objectives and goals. The next chapter provides and

overview on similar products and related technology, listing existing devices in the market, as well as presenting aspects related to sensors usually available in mobile devices.

Chapter 3 presents the Android Wear operating system used in this thesis project. Given that it is mainly based on Android, an overall description is first introduced and specific features are then identified.

On Chapter 4, some approaches used for pattern recognition are discussed. Given the complexity of this area, we selected two well-known and accepted solutions that were analyzed in more detail during the internship.

Chapter 5 describes all the decisions and steps in the project implementation. Validation approach and results are presented in Chapter 6.

The final chapter presents the main conclusions and put in perspective future developments for a more stable and intuitive solution.

2. SIMILAR PRODUCTS AND RELATED TECHNOLOGIES

Technologic development in the area of mobile and wearable devices contributed to the growing interest of the scientific community in the development of gesture recognition solutions.

Applications that can make use of solutions of this type are numerous and can contribute to one of the most important areas of society - healthcare - since applications that make use of gesture recognition can be used to control physical exercising, physical therapy, activity detection, among others.

All the theoretical aspects investigated throughout this thesis, that were considered relevant to the proper implementation of the project, will be referred in this chapter.

In a first stage, related products and ongoing researches are discussed. This is followed by a review of the inertial sensors used for implementing the thesis goals.

2.1. RELATED WORK

In order to become fully aware of the existing solutions in the market that could be competitors to the one to develop, a survey of the main products that claim, with different approaches, to respond to the initial problem was done. This investigation allowed not only identifying the degree of maturity of the solutions, but also understanding what are the best approaches that could be followed in the project development and which improvements to existing solutions could be proposed.

The organization of this subchapter contemplates therefore a description of the products that exist on the same platform used for this project (Android Wear), as well as other solutions that have been created on different platforms, but aspire similar results. Finally, some proposals developed in a research context but not yet in the market, are presented.

2.1.1. RELATED PRODUCTS USING ANDROID WEAR

Until the writing of this document, only two software solutions, capable of producing similar results to the project objective, on Android Wear platform, were known. These solutions will be thoroughly described below.

TrackMyFitness company came up with the application "Exercise Tracker: Wear Fitness", a solution that include most of the features that are part of the final objective of this dissertation. Created for smartwatches, tries to follow the premise already mentioned by Google to develop standalone applications for this type of wearable devices, while providing an application for smartphone that serves primarily for the data records from the training conducted with the smartwatch [1]. Of its key features, the ones that stand out are:

- Exercise auto-detection;
- Counting repetitions;
- Logging Practice.

It features an intuitive interface with no need for user interaction to start your workout, as can be seen in Figure 1.

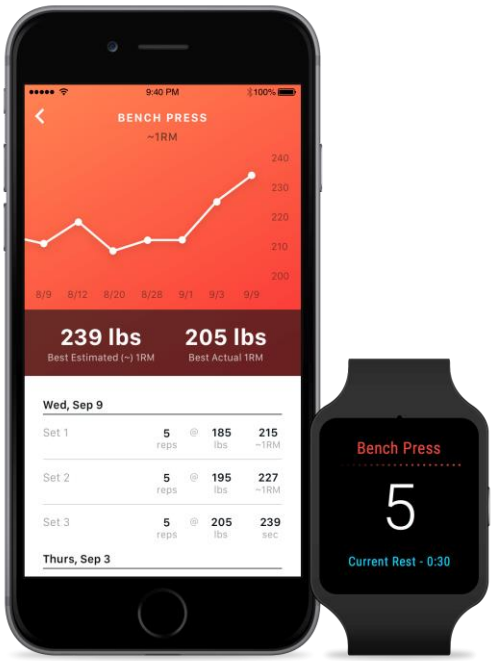


Figure 1: Application “Exercise Tracker” [1]

Information displayed on the smartwatch indicates the number of repetitions of the exercise in real-time and the user heart rate [1].

Another solution is the application "DoFit", created by the company with the same name, which was launched in beta and has some extra features. Also created to run on smartwatches, it uses an application for smartphone that collects data from conducted training exercises and saves performed exercises [2].

The main features of this solution are:

- Exercise detection in real-time;
- Count of repetitions;
- Exercise guided through animations;
- Activity Recognition (run, walk, stop).

An illustrative interface used by the solution is presented in Figure 2:

.

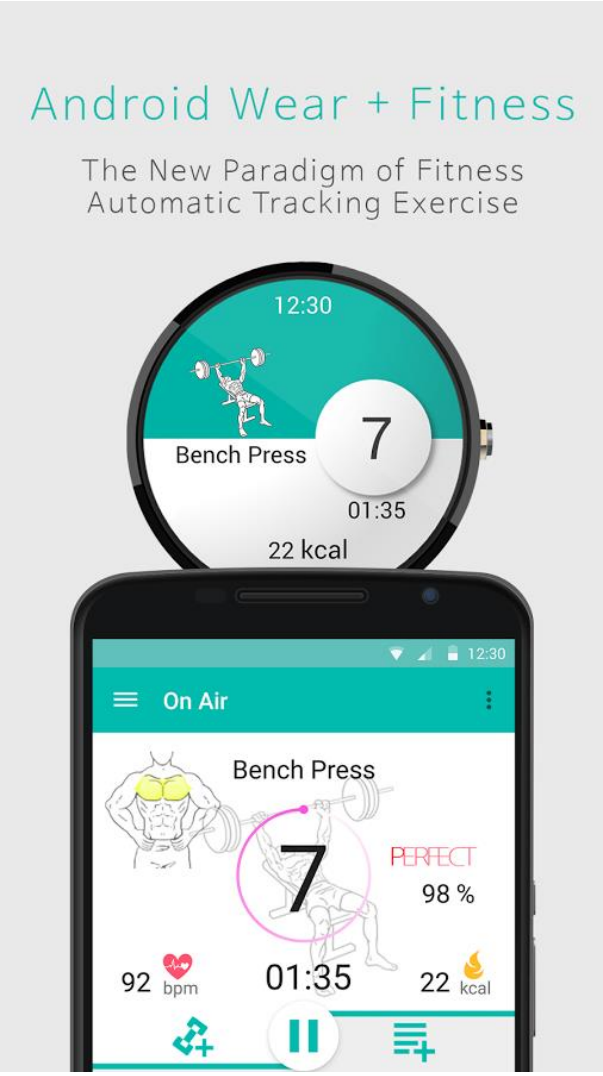


Figure 2: Application “DoFit” [2]

From Figure 2 it is easy to see how the interface of this solution behaves, presenting the user with a variety of information related to the performed exercise, including the name given to the exercise and an animation that shows in detail how this should be done [2].

Even though this is a very complete solution, it has an important drawback related to the fact that it needs the smartphone to store exercises and information about the exercise quality. This makes this solution not fully independent from the use of the smartphone, meaning that the comfort in its utilization can be compromised, although its functioning is as intended for such a problem.

2.1.2. WEARABLE DEVICES IN THE MARKET

The products that will be described have not been developed to run on the desired platform, but have similar objectives to the desired solutions. For this reason, it was considered relevant to analyze its features.

The *Moov* is a small device created by the company with the same name and which, as stated, has the features of a Personal Trainer and a Sports Tracker [3]. Regarded as a unique solution in the fitness area, this device not only detects activity, but presents also some interesting features such as:

- 3D Motion Capture;
- Resistence to Dust and Water;
- Count of Repetitions.

Combined with this device, the company has also created an application for smartphone that keeps track of data from the wearable and, from there, executes commands and makes decisions about the activity that the user is performing at the time [3].

Equipped with several sensors, this device has been used in sports such as swimming, running, boxing, cycling, among others. Figure 3 represents a purely illustrative image of this device.

Figure 3: Wearable “Moov” [4]

Associated with this device, a smartphone application was created that keeps track of all the results, being the wearable used to treat sensory data while the application to transmit such data to the user [3].

Jawbone is another company offering multiple wearable devices for tracking activities. Its top-of-range model - Jawbone UP3 - is the one with a higher number of features available [5], including:

- Calculation of calories burned;
- Detection and Tracking of sleep;
- Heart Rate Monitoring.

The approach used by this company, that consist of the creation of an application for smartphone that provides information and data obtained in wearable so that the user may keep track of his activity, is similar to the creator of *Moov*. This wearable consists of a 3-axis accelerometer, heart rate sensor and Bluetooth 4.0 Low Energy (BLE)[5].

The following figure illustrates the model "UP3" created by Jawbone company:

Figure 4: Wearable “Jawbone UP3” [5]

Using a similar approach to that used by the *Moov*, this solution makes use of Bluetooth for the transmission of data between the wearable and a mobile application, with no option to provide the data directly on the device.

Microsoft also entered the field of wearables with the "Microsoft Band" device, a solution that has an extensive list of features, some of which are:

- Detection and Tracking of sleep;
- Activity Recognition;
- Burned Calories Counter;
- Measuring Heart Rate;
- Guided Training;
- Tracking weights.

With these and other features, and comparing to other solutions, the Microsoft Band turns out to be one of the most complete in this area, and is compatible with most of the existing platforms (Android, IOS, Windows Phone), integrating also social networks and other intelligent notifications [6].

Consisting of a led display and various sensors (accelerometer, gyroscope, GPS, barometer, heart rate, thermometer, etc.) this solution provides a large number of parameters from the performed activity [6].

2.1.3. ONGOING RESEARCHES

The topic of this dissertation, specifically the detection of gestures and pattern recognition, is also an active area of research within the academic world. We will focus on two scientific papers that illustrate some of this work.

A work by M.Kalyan Chakravarthi, Rohit Kumar Tiwari and Shaunak Handa from VIT University in India [7] uses neural networks for gesture recognition using an accelerometer. With the title "Accelerometer Based Static Gesture Recognition and Mobile Monitoring

System Using Neural Networks" this research aimed to create a low-cost wearable device that could make the locomotion control in a robot using static gestures.

Under the proposed method, the accelerometer output is used to train a neural network using an algorithm called "Learning Vector Quantization" [8]. This neural network is used to identify three possible movements of a robot: moving forward, stopped and movement to the left. After a training phase, some random data is received from the accelerometer and the recognized gesture is transmitted via Bluetooth to the robot motors, which performs the intended movement through the system. A block diagram depicting the system is illustrated in Figure 5.

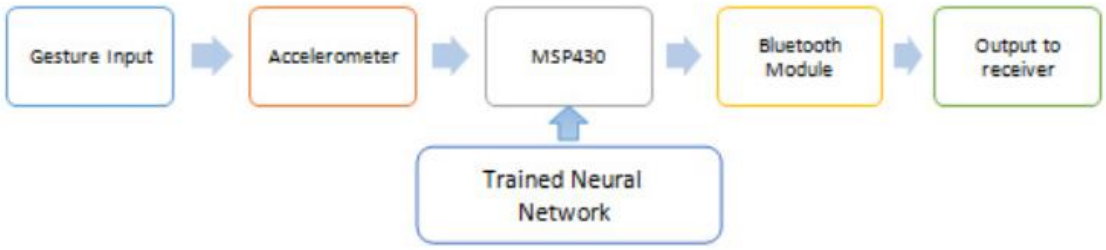


Figure 5: System Block Diagram [7]

A classification algorithm such as "Learning Vector Quantization" combined with the features of a neural network is capable of performing a non-linear rating effectively, being that the Learning Vector Quantization algorithm is based on the Euclidean distance calculation methods and classifiers as "nearest-neighbor".

To validate the proposed approach, 40 samples for each gesture were collected and testing enabled concluding, as expected, that the precision was directly related to the size of trained data, i.e., the longer the training dataset, the greater the success rate of their classification. Experiments for the given dataset resulted in a gesture recognition rate of 93% [7].

Another work is presented by Guillaume Plouffe and Ana Maria Cretu. With the title "Static and Dynamic Hand Gesture Recognition in Depth Data Using Dynamic Time Warping" [10], this research uses computer vision methods to get pictures from gestures made with the hand, through a Kinect camera, which are interpreted thanks to a pattern recognition algorithm called "Dynamic Time Warping".

According to the presented solution, image data are collected from the Kinect camera and obtained information about the depth of all pixels in the image. With the pixels depth, and through an algorithm that seeks and identifies all the points that are below a specific depth range, is obtained the hand contour of the person, as shown in Figure 6.

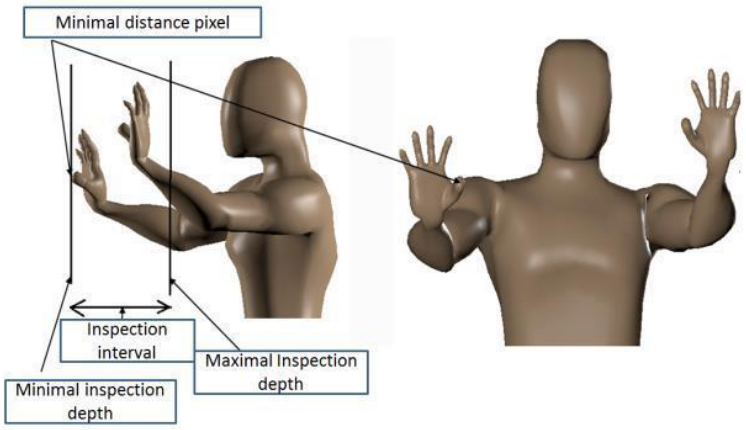


Figure 6: Method for Data Depth Recognition [10]

Figure 6 illustrates the approach used to detect objects, in this case the hands. Only if the hands are inside the image inspection interval, the system takes into account the resulting image for the subsequent calculations [10].

Following this, and after getting the hand contour image, the identification of the palm center and of the points of the fingers is done. For this is used the k-curvature algorithm [9], which is essentially based on the changes of angle slope of the tangent lines to the hand contour. These values are then stored as reference values that are used by the Dynamic Time Warping algorithm for gesture recognition.

Figure 7 illustrates a more detailed description of each phase of this process.

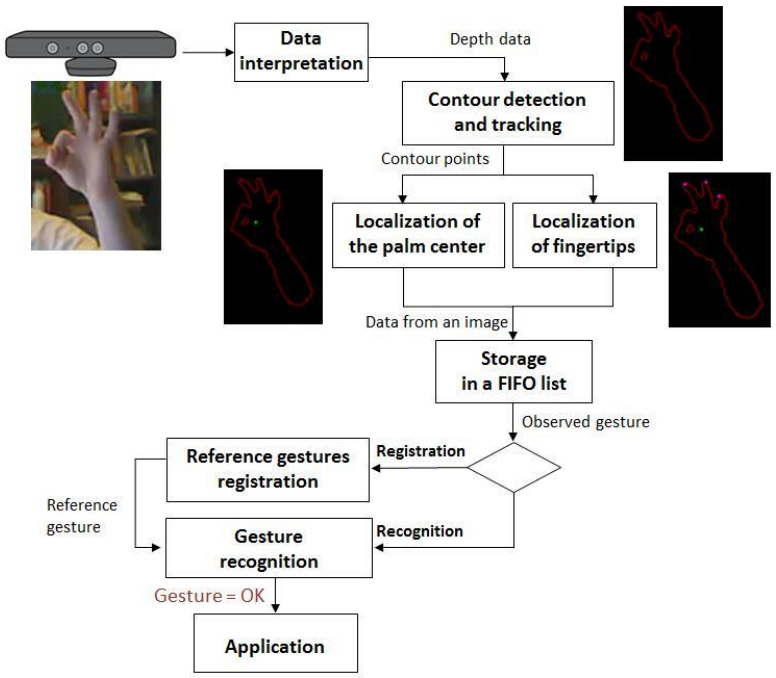


Figure 7: Operation of the System created Algorithm [10]

In Figure 7 is observed that the algorithm used for gesture recognition has two functions:

- Registration of the observed gesture information, if it is chosen to use as reference;
- Recognizing the observed gesture, comparing with the existing reference gestures.

Conducted experiments in order to distinguish from 16 gestures showed better and more consistent results when compared with other solutions. A precision between 90% and 100% was obtained [10].

2.2. INERTIAL SENSORS

Inertial sensor is the term used to define every device that can measure parameters such as acceleration and angular velocity along different axes, being used in applications that involve analyzing movements [12] [13]. Called "Inertial Measurement Unit" this electronic device generally consists of accelerometers, gyroscopes and magnetometers that through the data collected is able to study the movement in the plan and space [11].

In this dissertation, the use of sensors focused only in applications that involve obtaining movement and rotation data, because the goal is to detect gestures from a smartwatch. Given this, research focused on inertial sensors, leaving all the remaining sensors for another work.

The next sections provide some theoretical foundation for the three main inertial sensors: Accelerometer, Gyroscope and Magnetometer.

2.2.1. ACCELEROMETER

The Accelerometer is a device capable of measuring accelerations along several reference axis and its output is represented in the International System of Units, or SI, by m/s^2 or g-force. This sensor is part of one microelectromechanical systems circuit (MEMS) that measures acceleration forces caused by gravity movement [18]. The advantage of the integration of such circuits in most of today's smart devices relates to the fact that the cost and size associated to its construction has been drastically reduced without compromising the quality of the equipment [13][14].

Most existing accelerometers contain three axes (X, Y, Z) which are used to perform measurements of accelerations and determine the correct orientation of the device. This information can be used to obtain the speed and the device's position. The resulting data from a sensor of this type can be used to sort directions and movements, and taking into account the objective of this work, this is a very important device for obtaining data relating to gestures made with the equipment [47][45]. Moreover, as it is a sensor already integrated in a vast majority of devices in the market, such as smartphones and wearables, it is an excellent candidate to help implementing the desired functions.

Research on this type of sensors started in the mid-50s, with the creation of a device capable of measuring velocities and accelerations. The measurement of human movements linked to acceleration forces has been studied in more detail in 1970 due to technological advances at the time. With the evolution and improvement of nanotechnology and the rise of micro-electromechanical systems (systems that integrate mechanical, sensory and electronic elements in a very small chip), or simply MEMS, several Accelerometer features received a positive impact, such as their performance and the related energy consumption linked to its use [48].

In a simple way it may be noted that one of the basic operation principles of an accelerometer is based on the use of a test weight or seismic mass, clinging to a mechanical suspension system which is relative to a specific reference. The felt inertial force due to acceleration or gravity itself causes the test mass deviate according to Newton's second law, and this

deviation is subsequently measured electrically, resulting in an acceleration value in relation to a reference, a process illustrated in Figure 8 [45][46].

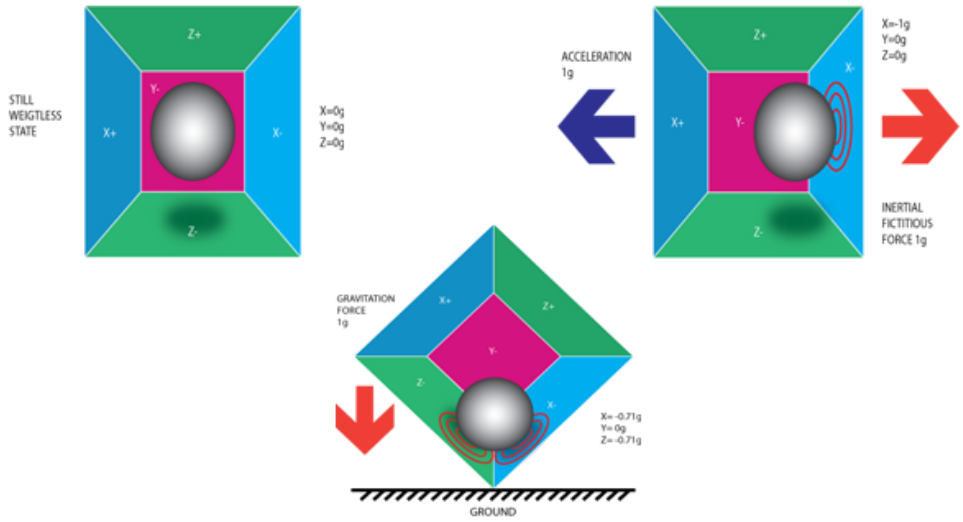


Figure 8: Representation of the Basic Principles of an accelerometer [19]

From a more technical point of view, accelerometers internally contain small fixed capacitive plates and other small springs attached to it that moves inwardly as acceleration forces acting on the sensor. The movement of these plates in relation to themselves generates a difference capacity, which by calculation is used to determine the acceleration experienced in the device as shown in Figure 9.

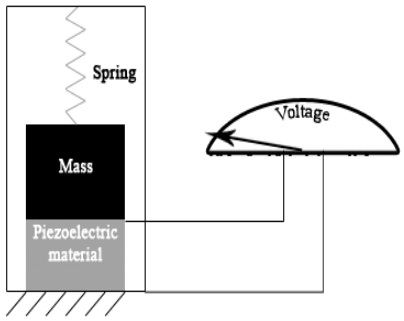


Figure 9: Accelerometer operation mode [20]

Other type of accelerometers based on different principles exist. Piezoelectric accelerometers, which depend on small crystals to record electrical changes, as soon as they are placed under acceleration effects, are an example.

2.2.2. GYROSCOPE

Unlike the accelerometer that obtains measurements relative to an external reference, the gyroscope performs measurements of its own rotation based on physical principles such as the angular movement. This sensor is also integrated in MEMS systems and therefore is widely used in intelligent devices [40][41].

A gyroscope in reality consists of a disc mounted on a frame such that it can be free to rotate rapidly under a shaft that itself can change its direction. If there is any external force applied to change the direction of rotation, a reaction force is established and applied to the disc in the opposite direction, being this the process responsible for calculating the device's orientation. From this calculation result, usually three values [43][42]:

- The device rotation with respect to the Z axis;
- The device rotation with respect to the Y axis;
- The device rotation with respect to the X axis.

From the above values, various information can be obtained. The most common is the conversion of these values in angles, called Euler angles, which enable a more accurate and reliable orientation of the device in question.

In a simplistic way, it can be said that a mechanical gyro is constituted by a rotating disk mounted in a frame, that are inside two metal rings perpendicular to each other, as shown in Figure 10 [41].

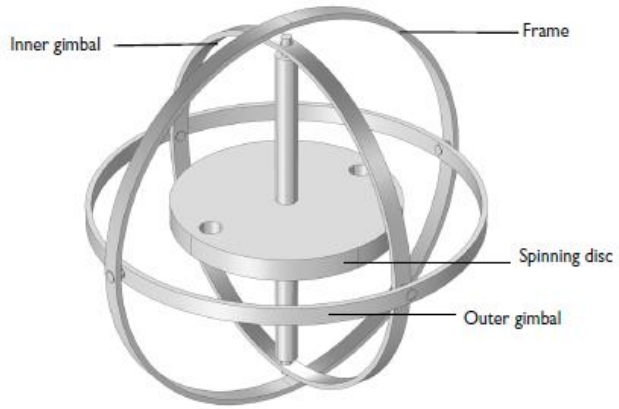


Figure 10: Constitution of a Mechanical Gyroscope [21]

From the above figure, it is possible to see which are the degrees of freedom of the disk's rotation, since it will rotate indefinitely and independently of the alignment of metallic rings to which it is hooked.

Although it has a relatively simple mode of operation, the implementation of gyroscopic electronic devices is very complex and involves methods of great precision. Mobile devices using this type of sensors usually have a higher power consumption and this might be an important limitation upon its use in applications [43].

Figure 11 presents an operation diagram of a MEMS gyroscope, identical to those used in most mobile devices.

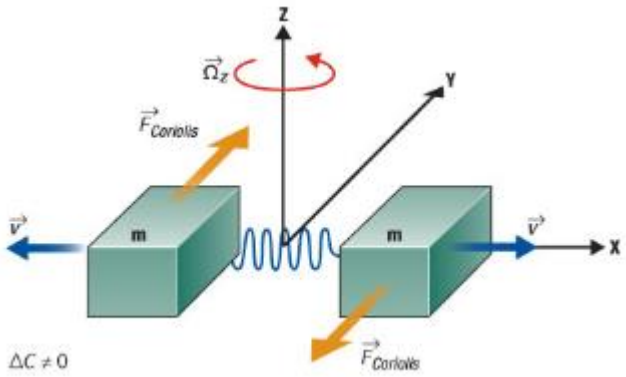


Figure 11: MEMS Gyroscope Operation [22]

The figure above describes the configuration used by most MEMS gyros, called tuning fork configuration. This type of configuration consists of two masses that are moving and constantly oscillate in opposite directions. If an angular velocity is applied to the system, the Coriolis force on each mass also acts in opposite directions, resulting in a capacity difference. The value that represents this capacity difference is then converted to voltage values, and the same value converted into units relating to the amount of rotation felt on the system in a given axis.

The orientation provided from this type of movement is not affected by the inclination of the machine where the disk is mounted, making this sensor a solution to be used in various applications to provide stabilization capabilities or even to maintain reference directions in navigation systems and autopilot solutions. Currently, most video games that run on smartphone present Gyroscope features to its control, and with the advent of wearable devices with virtual reality, the use of this type of sensor is more important than ever, since he is responsible for detecting orientation of the player's view, a feature where accelerometer features are not enough [47][45].

Within this dissertation, the use of a sensor such as a gyroscope becomes important due to the fact that a human gesture shall be described not only as a change in position, but also as a set of orientations that change over time and that are required for correct and accurate results.

2.2.3. MAGNETOMETER

Besides the collected movement data, through the accelerometer, and the rotation via gyroscope, another important sensor for monitoring inertial data and human movements, as is required in this dissertation, is the magnetometer.

A magnetometer is actually a compass that is attracted to the Earth's electromagnetic field. In electronic devices, magnets or magnetized needles are however not used, due to the possible damage of the equipment [45][47].

The great majority of magnetometers is based on phenomena such as the Hall effect, working with magneto-inductive materials, capable of measuring differences in terrestrial electromagnetic fields. This process may however be disturbed if the sensor is in a short distance from other electronic devices or even another type of ferromagnetic material.

For his functioning, a MEMS magnetometer reacts electrically when a magnetic field perpendicular to the semiconductor material that constitutes the sensor is applied. When a perpendicular electromagnetic field is applied to the semiconductor material of the sensor, there will be a distortion in the current flow that runs through the component. This current distortion will generate that the density of the electrons present be uneven, and this value will be reflected in a difference in potential to the semiconductor terminals, being this potential difference known by Hall effect voltage, an illustrated phenomenon in the following Figure 12 [45].

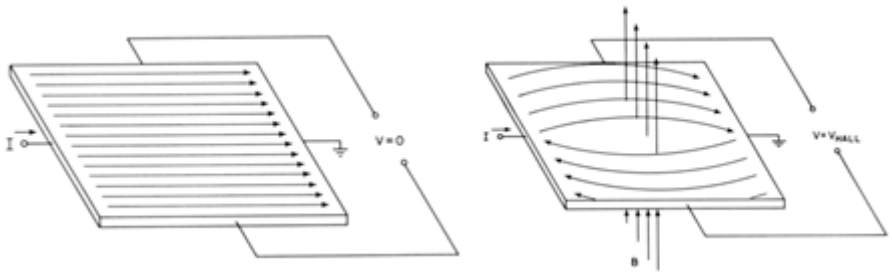


Figure 12: MEMS Magnetometer Operation

Through the above Figure 12 it is possible to observe that if an electromagnetic field is applied parallel to the conductor, there will not be current distortion, whereas if the magnetic field is perpendicular, the effect will no longer be the same. If the resulting current of the force exerted by the electromagnetic field is constant, the Hall voltage is directly proportional to the strength of that field. This Hall voltage results in the values of a magnetometer of this type, functioning as an electronic compass.

The use of sensors for detecting electromagnetic fields is rapidly growing, with typical applications of detection of metals and electronic compasses. Other uses of this type of device include one of the most innovative areas in the moment: the indoor positioning [44].

Researchers at the University of Oulu, in Finland, have shown that it is possible to obtain the correct position of objects within buildings through geomagnetic technology obtained from the Earth's electromagnetic field. Named *IndoorAtlas*, this solution obtains sensor data from a smartphone or tablet, and through changes in the electromagnetic field can create a kind of map of the surrounding area, and everything without the need for extra equipment for this purpose [23].

2.2.4. AREAS OF APPLICATION OF SENSORS

As previously mentioned, the use of inertial sensors is increasingly being adopted for the creation of intelligent applications capable of monitoring people and surroundings with less user interaction. Besides the case of indoor positioning, there are other important areas of applications that make use of inertial sensors, such as:

- Activity Recognition;
- Fall Detection;
- Gesture Detection;

2.2.5. SENSOR FUSION

The Sensor Fusion is a combinatorial process of sensory data from different devices whose main purpose is that the resulting data has a more complete and accurate information, a factor that is not possible when these sensors are used individually [49].

This process permits to overcome certain limitations of some sensors through correlations between each other. By fusing data from two different sensors, the error that sensor 1 displays can be fixed with some information from sensor 2 [49].

For the purpose of this thesis, the use of sensor fusion is relevant to the extent that it is necessary to obtain the rotation that the smartwatch makes.

A possible example of a sensor fusion process is the case of obtaining the orientation of the device through the inertial sensors. Using the accelerometer, magnetometer and gyroscope it is possible to get the relative angles of the device's orientation, although the accelerometer and the magnetometer suffer from noise in their readings [24].

Through sensor fusion it is possible to obtain a result derived from all the data collected from these three sensors and to achieve an output signal with little noise and without drift in readings. In a simple way, individual data can be summarized as:

- Accelerometer data correspond to the data related to gravity, felt in relation to the Earth's center;

- The Magnetometer data function as a kind of compass;
- The Gyroscope data result in angular rotation speeds for the three axes.

In an ideal scenario, the data from the accelerometer and magnetometer are sufficient to calculate the device's orientation. However, due to the noise problem experienced in these two sensors, the results turn out not to be as accurate as it would be expected. Gyroscope data, with a fast response time than the magnetometer, can be the solution to mitigate the global problem. One possible implementation of a sensor fusion process for obtaining the device rotation values, is illustrated in Figure 13. This method is described in the research of Paul Lawitzki [24].

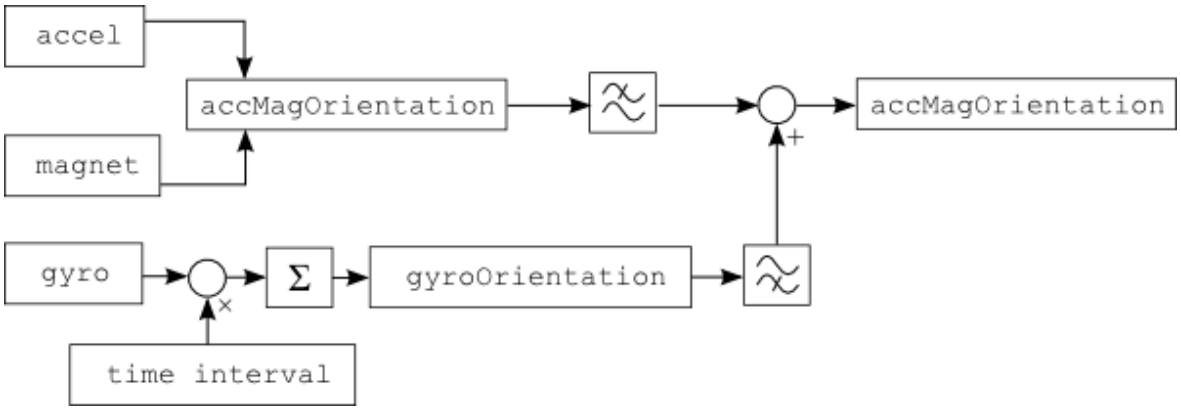


Figure 13: Sensor Fusion Diagram [24]

From the analysis of the figure, it is possible to observe the existence of filters, which together with the fusion of values, help alleviating some errors in the obtained readings. A low-pass filter can be applied to the Accelerometer and Magnetometer, used to support sensory information over long periods of time, while to the gyroscope data, a high-pass filter is applied to perform calculations on small time intervals.

Figure 14 illustrates the advantages of using this process for data fusion. The filtering process on the accelerometer and magnetometer decreases the noise problem and the filtering on the gyroscope values helps solving the drift problem. The final combination of these two outputs results in a value without these initial problems.

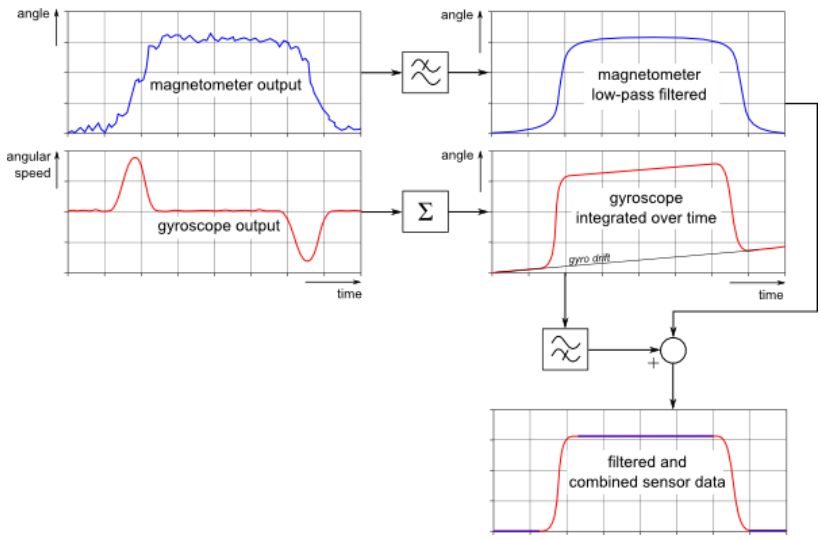


Figure 14: Graphical Results from a Typical Sensor Fusion Process [24]

The main drawback of this process is that it entails high processing to provide the results. This is even more relevant in the scenario of this thesis as the hardware platform in use, a smartwatch, has lower resources than a smartphone.

In order to deal with this high demand of sensor fusion processes, Android developed the so-called "Software Sensor". Software Sensors are high-level implementations that make use of the values read by one or more sensors simultaneously to provide significant results and which would not be possible to obtain in a direct way from these same sensors [50]. Thus, the sensor fusion processes described above become transparent to developers, and in this area, the Android platform has several sensors that implement this type of processes, which we can highlight:

- Rotation Vector
- Gravity Sensor
- Linear Acceleration Sensor

The Rotation Vector is the more relevant in the context of this thesis. According to the documentation provided by Android, the Rotation Vector Sensor represents a device orientation as a combination of an angle θ, in which it has exercised some sort of rotation, and an axis (x, y or z), where this rotation has been made [50].

The results from this implementation are expressed as follows:

- x*sin(θ/2)
- y*sin(θ/2)
- z*sin(θ/2)

Where sin(θ/2) represents the magnitude of rotation experienced by the device, and x,y and z the direction of the sensor for each axis of rotation, being these three axes defined in the same way as for the accelerometer. The reference coordinates system is defined as a direct orthonormal basis, as it can be seen in Figure 15.

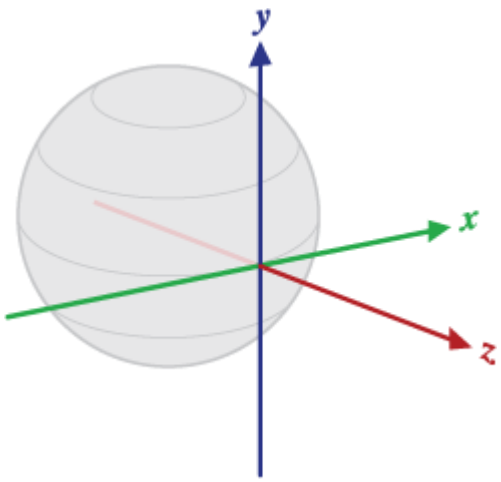


Figure 15: Rotation Vector Sensor Coordinate System [25]

The coordinate system shown in Figure 15 is described by the following characteristics [25]:

- X is defined as the vector product Y x Z. It is tangential to the ground at the device's current location and points approximately East.
- Y is tangential to the ground at the device's current location and points toward the geomagnetic North Pole.
- Z points toward the sky and is perpendicular to the ground plane.

That said, and in a way to consolidate all the theoretical reasoning for this sub-chapter, the obtained results after the sensor fusion process, for this thesis, are the Euler Angles that represent the rotation of the device on axis x, y and z, as shown in Figure 16.

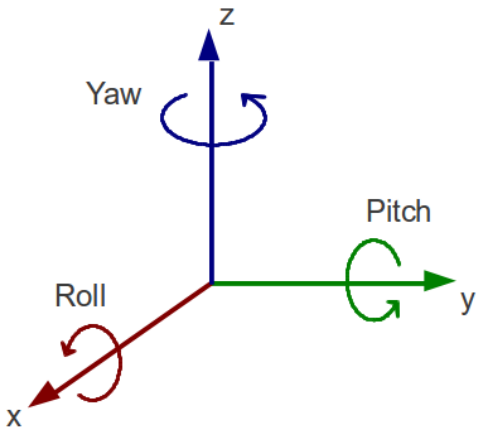


Figure 16: Euler Angles obtained from Sensor Fusion [65]

The obtained data of such a process is illustrated in Figure 16 and can be defined as:

- Yaw: Rotation on the vertical axis, Z;
- Pitch: Rotation under the lateral axis, Y;
- Roll: Rotation under the longitudinal axis, Z;

These values, along with other data, are the reference and the "core" responsible for collecting sensory information concerning the exercise performed by the user, and are therefore one of the most important factors in the solution set.

2.2.6. SENSOR SAMPLING RATE

In any product or solution in that its operation is dependent on the use of sensors, one of the important factors to take into account is the sampling frequency or sampling rate of the data intended to be read.

A high data sampling rate makes it possible to obtain a greater number of values for a period of time, which in some cases makes the interaction of the solution more precise, although choosing this factor must be taken in the account the operation and usability of the application.

Regarding the Android platform, the data sampling rate can be set by the application developer but have, however, to correspond to one of the four pre-defined sampling rates [50], which are:

- SENSOR_DELAY_NORMAL

  Sampling Rate suitable for screen orientation changes.

  Depending on each device, typically is around 200 microseconds.

- SENSOR_DELAY_UI

  Sampling Rate suitable for user interfaces.

  Depending on each device, typically 60 microseconds.

- SENSOR_DELAY_GAME

  Adequate sampling rate for video games.

  Depending on each device, typically 20 microseconds.

- SENSOR_DELAY_FASTEST

  Faster sampling rate that the device can perform

  In the ideal case, the delay between each sample should be near zero, or the fastest possible.

Through the use of these sampling rates, the Android platform has an idea of the minimum delay which has to wait between each sensor reading, although it can be concluded that it will not be a perfect accuracy between the sensor read time.

In addition to these sample rates, was introduced from the Android version 3.0 (API 11) a specification that allows developers refer to this delay reading between samples as an absolute value, in microseconds. Although in reality the system associates this absolute value to some pre-defined rates by it fees, and this value is only a suggestion for Android to process to the desired cadence [50].

Using an appropriate sampling rate established for the application, associated with the proper use of the sensor, makes it possible to decrease the primary problem of using these components in mobile devices: quick battery drain. Being this a continuous and repetitive process of the smartphone, to perform constant hardware readings, the autonomy of the

overall device is affected and, therefore, a poor management of the system resources makes the operation of the created solution itself at risk.

In the next chapter are gathered all the important information that involve the platform used in the thesis: Android Wear. Starting by an introduction to the technology and architecture, it is explained some important features and it ends with a brief explanation of the Integrated Development Environment (IDE), created for the Android development.

3. ANDROID WEAR OPERATIVE SYSTEM

Android Wear is a version of the Android Operating system designed for wearable devices, with the objective of making many features from a smartphone to be made available in smaller devices, like smartwatches.

The choice of this operating system for the development of the work proposed for this thesis was driven by the fact that it is an open-source solution, what contrasts with alternatives like watchOS [61], of Apple, which although equally powerful, does not provide the freedom of customization that the Google system does. Besides these two alternatives, there are other operating systems used in wearables [58][59][60] that can be highlighted:

- Tizen OS
- Pebble OS
- Firefox OS for wearables

Another important reason for having chosen the Android Wear is the fact that it has been adopted by many smartwatches brands and manufactures, as the operative system that runs on their devices.

In this chapter we provide a brief background on this operating system, with a description of its architecture and operation model, its main features, and some integrated development environments for the development of applications for this platform.

3.1. OVERVIEW

As previously mentioned, the Android Wear is an operating system created by Google, that runs on wearables, being mostly used in smartwatches. Using Bluetooth technology, this system uses a pairing mechanism with the smartphone, thereby receiving notifications and integrating different phone features on the smartwatch. An interesting curiosity is related to the fact that the smartphone operating system does not necessarily needs to be Android, because Android Wear provides compatibility with systems like Apple's iOS [29].

Announced on March 18, 2014, this operating system appeared as an answer to the existing technological needs for a more comfortable solution, but at the same time powerful, that could be used in smaller devices, such as watches. After the announcement of this system, several companies such as Motorola, Samsung and LG were presented as partners, and, just two months later, the first two smartwatches of the market were presented, with LG getting ahead of the competition, being the first company to market its device, dubbed "LG G Watch". Other products from different brands, of which we can highlight the Motorola Moto 360, were also presented. This device has an innovatively screen layout, adopting a circular screen, more like the conventional watches. Alongside these products, in December of the same year, the operating system itself suffers updates in a way to be consistent with the Google's policies regarding the Android 5.0 software for smartphones, and with this update a new range of devices from several technology companies were made available [54][29].

Figure 17: Android Wear Operative System (Moto 360 smartwatch) [29]

In order to provide a more practical and fast information to the user, this operating system had, until recently, an approach to create some kind of "appendix" of the smartphone, although the last conference from Google in May 18, 2016, has revealed that the way forward for this platform would be to use its full features with standalone applications, that is, that all the major smartphone features are included in the smartwatch itself, making the possibilities for the use of smartwatches superior than they were until now [54].

The list of smartwatches that use this operating system is already considerable, and some brands have chosen to develop the same device, but with different models for various activities [29]. Of all existing smartwatches, the following can be highlighted:

- LG G Watch e LG Watch Urbane
- Motorola Moto 360
- Asus ZenWatch e Asus ZenWatch 2
- Samsung Gear Live
- Sony SmartWatch 3
- Huawei Watch

- Fossil Q (Founder, Marshal e Wander)

It is possible to highlight from this list an interesting detail which is the fact that not only the leading technology companies are entering in the wearables market, but also the more traditional watch companies have presented their smart devices, as is the case of Fossil and Tag Heuer, showing that this may be a good path to follow from this technology.

Although the number of versions of this OS is not as significant as the ones provided for the standard Android OS, there are already some diversity as presented in Table 1.

Table 1: Android Wear Versions [30][31][32]

Android Wear Version	Relative Smartphone Version	Functionalities
Android Wear 1.0	Android 4.3 Jelly Bean	Bluetooth connection, Notifications, Voice commands
Android Wear 1.3	Android 5.0 Lollipop	Wi-Fi connection on the watch itself, interactive watch faces and battery and performance improvements
Android Wear 1.4	Android 6.0.1 Marsmallow	Navigation by gestures, voice messages and speaker support
Android Wear 2.0	To be Announced	Standalone Applications, virtual keyboard and handwriting recognition

From the examination of the Table 1 is possible to perceive a pattern in the upgrade versions of Android Wear, being that the announcement of a new update to the operative system of wearables is, almost always, followed by a major update made to the operative system for the smartphones.

The Android Wear software updates follow the same approach of the traditional Android, being that this process is received and performed, in most devices, with an over-the-air (OTA) process, giving a greater convenience to users, who are automatically notified by the device as soon as a new update is available. While this is a good approach, the fact that there is a wide range of Android Wear devices, in terms of hardware, makes so that software updates are not received all at the same time for all smartwatches, depending on this process not only Google, but also the manufacturers of the devices [17].

3.2. ANDROID AND ANDROID WEAR ARCHITECTURE

Android operative system is based on Linux and can be divided into six main layers, as shown in Figure 18.

At the bottom layer, there is the Linux kernel. As it was mentioned earlier, the Android platform was based in Linux, in part due to the open-source nature of this system, and to the constant security updates it receives. This layer provides the basic functionality of the system, such as the management of devices, procedures and memory [47][55].

The second layer of the Android architecture (Hardware Abstraction Layer) concerns the communication interface between Applications and existing Hardware resources, making the upper layers abstracted from all implementing low-level drivers and hardware [47][55]. According to several authors, this layer turns out to be a continuation of the Linux kernel. Therefore, there are many references that describe the overall architecture as consisting of only five layers.

There are two layers belonging to the same level where all the native libraries and the Android Runtime (ART) are found. The libraries contain a set of instructions that allows the device to handle different types of data, among which the following stand out:

- Media Framework: support for multimedia data playback and recording;
- Webkit: Browser Engine;
- SGL e Open GL: 2D and 3D graphics, respectively;
- SQLite: implementation of embedded SQL databases;

The Android Runtime provides a series of core libraries for java programming, that allow an easy use to all the device features. It is also in this layer that the Dalvik Virtual Machine is found. This virtual machine has been created specifically for Android to run Applications: each process instantiated by an Android Application has an instance from the Dalvik Virtual Machine, providing devices the ability to run multiple processes simultaneously, efficiently and optimally for a minimum consumption of memory. The virtual machine executes Dalvik Executable files (DEX) and has the capability to run Java classes that have been converted in this format, because Android does not have the capability to run Java classes [47][55].

The next layer (Application Framework) is what allows the creation of applications and is made up from several programs that control a wide variety of functions of the device, such as phone calls, location info, notifications, and data sharing. In Figure 18 the most important functions of this layer are presented.

Finally, in the highest-level layer is the layer of Applications, where most Android users interact with their devices, for the basic system functions, such as making calls, viewing contacts, browsing the Internet, etc. Through the services provided by this layer, developers can take advantage of its features for creating their applications in a simple and practical way.

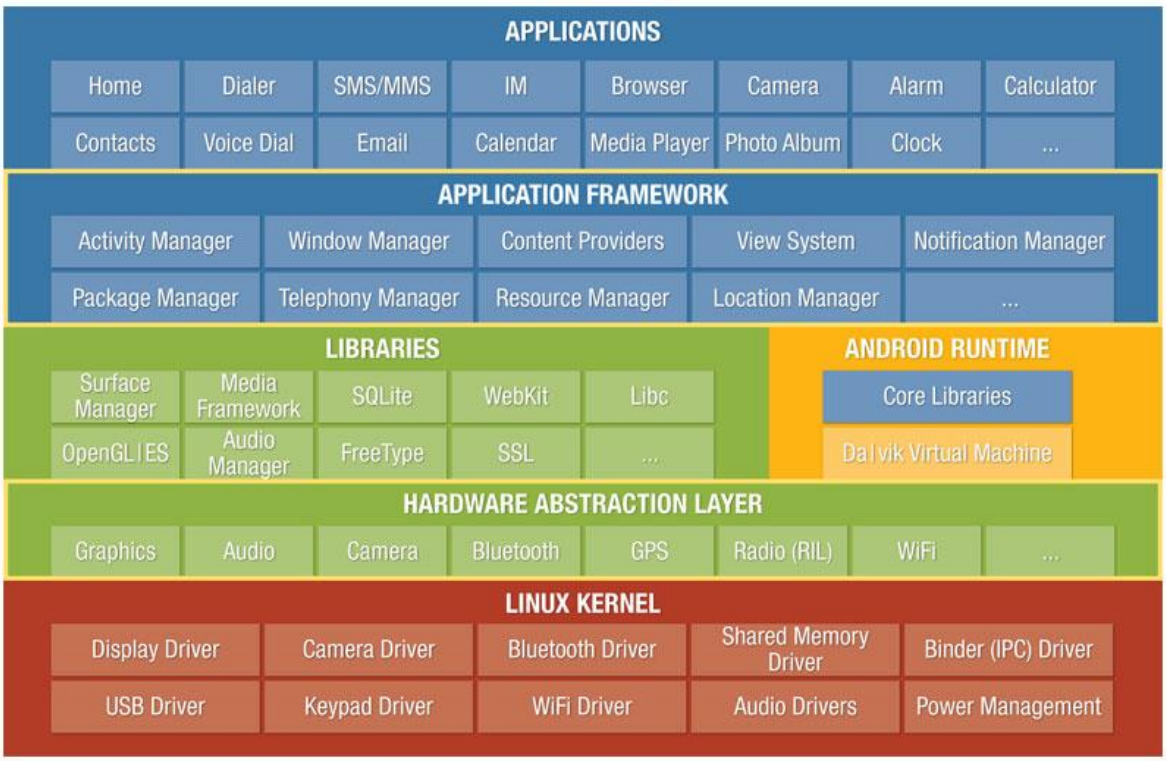


Figure 18: Android Operative System Layers Diagram [33]

Android Wear is identical to the traditionally Android. However, the operation of installing applications in these devices presents the approach illustrated in the Figure 19.

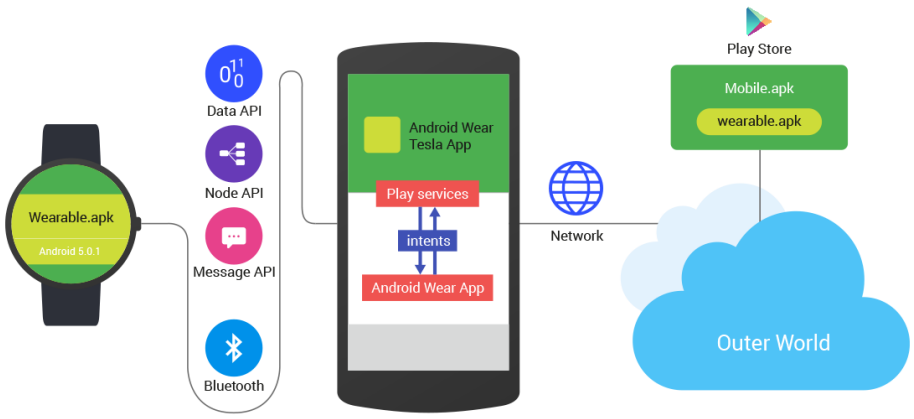


Figure 19: Android Wear Operation Diagram [56]

One of the advantages inherent to the use of Android Wear is the development for this platform being virtually identical to the development for a smartphone, requiring only the developer some care in the UI and some adaptations to the libraries. As shown in Figure 19,

the operation of a smartwatch, powered by Android Wear, is still largely dependent on the smartphone, and all communication between the two devices is done via Bluetooth. Some features and Android Wear APIs make use of the smartphone, as is the case of notifications, data from the Internet and even installing applications to the smartwatch. In this last feature the approach used by Google has the availability of applications directed to wearables in the same store of the smartphone applications, which requires that all applications from Android Wear have a module for the phone and one for the smartwatch, with no further the possibility of installing applications for wearables directly in the devices, or a native application store for this platform [56].

3.3. SPECIFIC FEATURES

Being this an operating system directed to specific type of devices, Android Wear has several features specifically designed for a simple and comfortable use. Being able to create a “bridge" of information between the smartwatch and the smartphone, it enables the user to get information without having to constantly pick up the phone. The ability of receiving notifications on the watch, are one of the most important features that can be identified [29].

Figure 20 presents one of the several layouts used for notifications on Android Wear, the so-called "notification card". This feature makes the smartphone use lower, as the user can receive and respond to notifications directly from the smartwatch. The user interface has also been carefully designed for this operative system, being the use of buttons replaced by swipe gestures, partly due to the small size of the device screen.

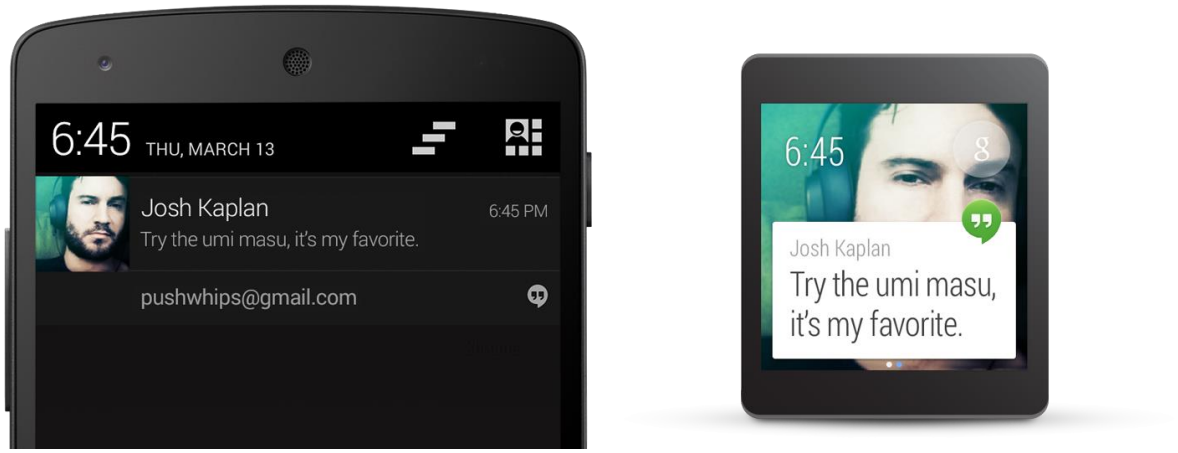


Figure 20: Android Wear notification card [34]

In order to facilitate access to the applications and settings of the smartwatch, the Android Wear has also available other types of interaction with the device, of which the following stand out:

- Voice recognition: the voice command "Ok Google," enables to automatically start the digital assistant Google Now that, with the latest software update 1.4, allows the user to perform searches with the Google search engine, send messages and even, depending on the smartwatch models, make audio and video calls, all with simple voice commands;

- Navigation gestures: also introduced in version 1.4 of the operative system, small gestures made with the wrist can now be used to navigate in the smartwatch interfaces.

Alongside these features, this operating system also has other features that distinguish it from other systems such as:

- Fitness Parameters Control (Pedometer, heart rate control);

- Remote control and location of the smartphone;

- Remote control of smartphone applications.

3.4. INTEGRATED DEVELOPMENT ENVIRONMENT

The development of applications for Android Wear ends up being very similar to the development of Android, because it uses the same programming language, Java, and the same Integrated Development Environment, Android Studio [57].

Announced as the official IDE for developing Android Applications, Android Studio is based on IntelliJ IDEA software and presented himself as the successor to the Eclipse IDE. Consisting of several tools aimed to facilitate and accelerate the development, this software has an intelligent code editor, a debugger, several libraries prepared for use, integration with version control system and the possibility of developing the same software for various Android platforms, as the case of Android Wear and Android TV [57].

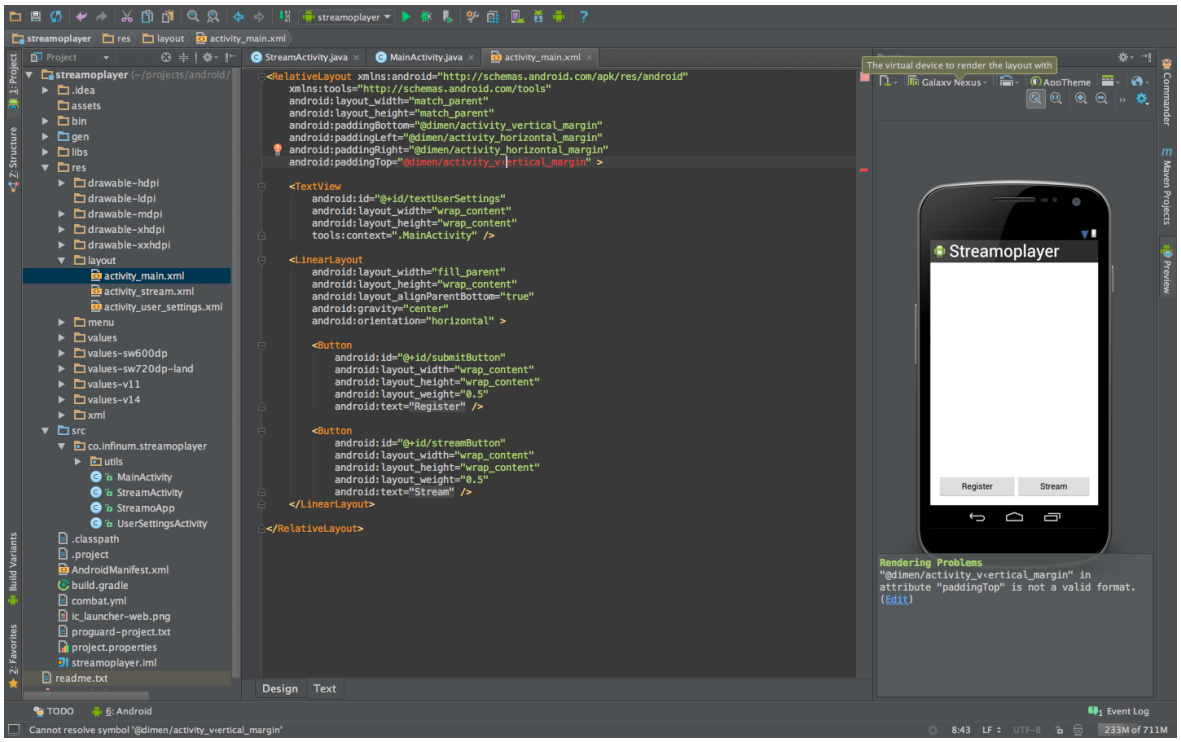


Figure 21: IDE Android Studio [35]

Based on Gradle build, through the Android Studio software it is possible to develop, compile, debug and deploy applications for any Android device, without the need to use other software.

Other of the interesting feature that this software offers is the fact that it includes an emulator of devices, which allows the testing of applications not only in real devices, but also make a pre-test on a virtual device. The connection between Android Studio and devices (real or virtual) is carried out by another Android Studio SDK tool, the Android Debug Bridge (ADB).

4. PATTERN RECOGNITION ALGORITHMS

The development of this thesis included a detailed study concerning algorithms for pattern recognition responsible for implementing the functionalities of analyzing and classifying the data. This is an important feature to be implemented to support physical exercise recognition.

This dissertation focused on two algorithms widely used in the pattern recognition area that are using two different approaches.

4.1. DYNAMIC TIME WARPING

The Dynamic Time Warping (DTW) algorithm uses its calculation capabilities for similarity classification between two time series sequences that can vary in speed [51]. This is an interesting feature in pattern recognition, because two data series are not always identical in size but similar in their pattern, minimum and maximum, as in the case of a performed gesture. Through DTW is possible to detect, whether the gesture has been executed in a slower or faster way than the reference with which is being compared.

In general, the DTW calculates the matching distance between two time series with certain restrictions, being this value generally corresponding to the Euclidean distance between two points in the Euclidean space [52].

Let *A [1 ... M]* and *B [1 ... N]* be two time series to be compared with DTW, the higher the similarity between the two, the smaller the resulting distance of the algorithm. The following Figure 22 illustrates the iterations made by the algorithm for calculating this distance between two time series.

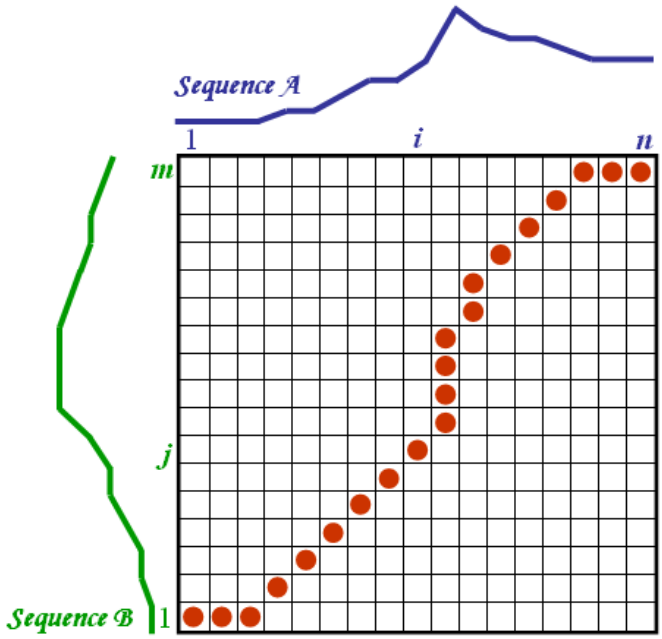


Figure 22: DTW operation for the match path calculation [26]

Through dynamic programming, the DTW iterates at every point of each time series, starting with the point (1, 1) until the end of the point (M, N). Figure 22 provides a graphical representation of the approach used by this algorithm over all the points of the two time series A and B. The optimal path between the starting point and any intermediate point (i, j) is obtained taken into account the distance between (1,1), (i, j) and previous neighbors of (i, j): (i-1, j) (i, j-1) and (i-1, j- 1) [46][45].

The final distance between (1,1) and (i, j) is then the actual distance between the referred points, plus the smallest distance between (1,1) and their neighbors (i, j). This algorithm then makes a calculation for all the combinations of points from the two time series, and returns

as output the maximum distance (matching cost) between the two time sequences, this being the minimum sum of all distances between each pair of corresponding points [46][51].

Figure 23 shows the correlation between points of two time series using the Euclidean distance and DTW. It is possible to see some differences when comparing the two figures, as in the case of using the calculation of the Euclidean distance, the correspondence points between the two time series is made only if the index of the points are the same, which does not happen with DTW, since the correspondence relationship of these calculations is done by looking for similarities between amplitude and scale the two series [45].

This factor causes that the two datasets do not have to be identical for a match to exist between each other, but the two may differ with respect to time and amplitude, only having to have the same pattern as a whole. In Figure 23 it is possible to see that the DTW Matching feature can associate the local maximum or minimum of each time series, and the Euclidean Matching can only associate the two time series from the index points of the datasets.

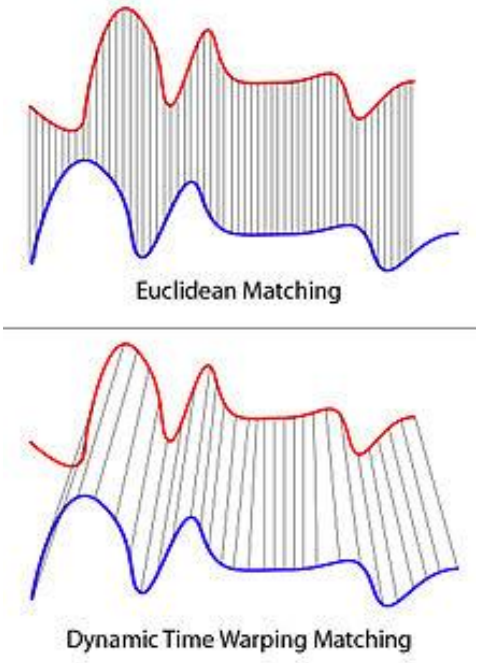


Figure 23:Comparison between Euclidean Matching and Dynamic Time Warping Matching Algorithms [27]

A possible pseudo-code of the Dynamic Time Warping algorithm is shown below:

```
DTW(v1: array [1..n], v2: array [1..m]) {
// v1 and v2 are the time series with n and m time points

  //S is a two dimensional data matrix that stores the similarity
  measures, such that S[0,…,n, 0,…,m], and i, j, are loop indexes.
```

```
  S := array [0..n, 0..m]

  S[0, 0] := 0

  FOR i := 1 to n DO LOOP
          DTW[i, 0] := ∞
  END

  FOR i := 1 to m DO LOOP
          DTW[0, i] := ∞
  END

  //Using pairwise method, incrementally fill in the similarity
  matrix with the differences of the two time series
  FOR i := 1 to n DO LOOP
      FOR j := 1 to m DO LOOP
          // function to measure the distance between the two
          points
          cost := d(v1[i], v2[j])
          S[i, j] := cost + minimum(S[i-1, j],    // increment
                                    S[i, j-1],      // decrement
                                    S[i-1, j-1])    // match
  END
END

Return S[n, m]
}
```

This algorithm has been used in various fields, such as speech and gestures recognition and multimedia data analysis. It can be easily used with any other data that can be converted into linear sequences.

In the context of this thesis, the features that an algorithm like DTW offers are ideal for a process of physical exercise recognition, because of the differences that can occur between two time series resulting from performed exercises. The fact that two gestures are never completely identical, even if performed by the same person, causes differences in speed and amplitude in the time series, an obstacle that is overcome by this algorithm.

4.2. HIDDEN MARKOV MODELS

The Hidden Markov Models are a statistical tool with several methods for calculation and modeling of time series data and can be well included in the fields of machine learning and data mining [28].

Developed in 1960 by Baum, the theory of this tool assumes the principle based on the mathematical theory developed by Andrei Markov in the early twentieth century, called Markov Process, which is described in Figure 24 [28][46].

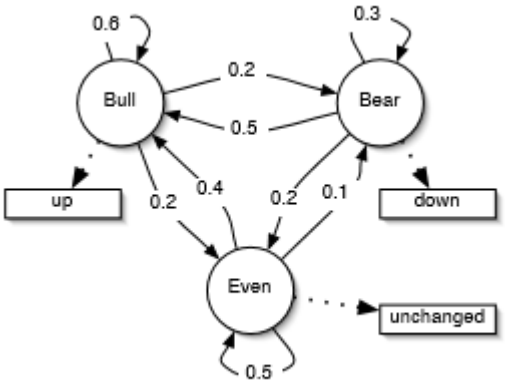


Figure 24: Markov Process Example [28]

The previous figure represents a state machine/diagram for a market share index, which describes a similar behavior to a Markov process. The model has three states (Bull, Bear and Even) and three points (up, down and unchanged), thus presenting a finite number of states with probabilistic transitions between each of them, such that, given any sequence of observations, it is possible to check which are the sequence of states that produced these observations. For example, a sequence of observations: down-up-unchanged can be easily verified that has been produced by the states: Bear-Bull-Even, being the probability of this sequence the product of the transitions, in this case 0.5 x 0.2 x 0.5.

As previously stated, this theory has served as the basis for a more complex implementation, the so called Hidden Markov Model, which may be represented by the Figure 25, as an extended model of the illustrated in Figure 24.

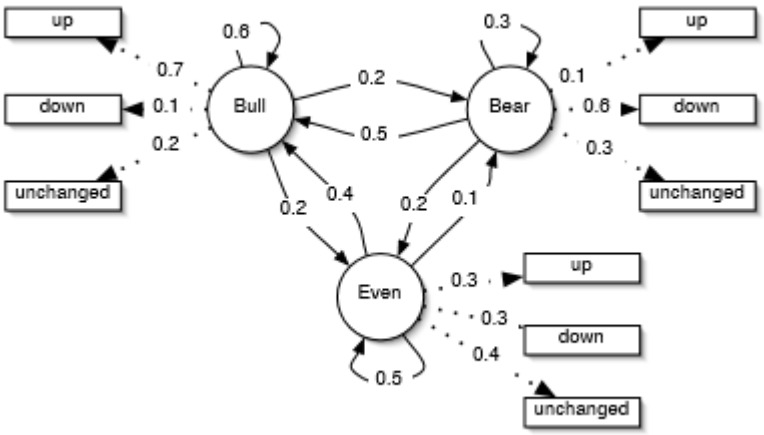


Figure 25: Hidden Markov Model Example [28]

The new model, shown in Figure 25, allows that all observations can be issued by each of the states with a finite probability, making the model much more complex and with the ability to better represent the human intuition, because in case of the stock market, this model states that the Bull market may have good or bad results, like the real world, although the largest probability concerns the increase of this market [28][46].

The main difference in the comparison between this model and a traditional Markov process, is that in a sequence of observations obtained through a Hidden Markov Model, it is not possible to define a sequence of states that gave rise to the observations, because the existing states have all the model observations, so the state remains "hidden", although it is possible to calculate the most likely sequence of state that has produced the referred observations.
Mathematically, a Hidden Markov Model can be defined by the following equation 1 [28]:

$$\lambda = (A, B, \pi) \tag{1}$$

- A is the transitions array, which keeps the probability of the state j, followed by the state i;
- B is the observations array, which keeps the probability of observing k be produced by the state j, regardless of the time;
- π is the initial array of the probabilities;

These parameters can be defined as the following equations 2,3 and 4:

$$A = [a_{ij}], a_{ij} = P(q_t = s_j | q_{t-1} = s_i) \tag{2}$$

$$B = [b_i(k)], b_i(k) = P(x_t = v_k | q_t = s_i) \tag{3}$$

$$\pi = [\pi_i], \pi_i = P(q_1 = s_i) \tag{4}$$

Where S is the states alphabet, V the observations alphabet, Q a fixed sequence of states, and O the corresponding observations, such that:

$$S = (s_{1,} s_{2,} \dots s_N) \tag{5}$$

$$V = (v_{1,} v_{2,} \dots v_M) \tag{6}$$

$$Q = q_{1,} q_{2,} \dots q_T \tag{7}$$

$$O = o_{1,} o_{2,} \dots o_T \tag{8}$$

So starting from a HMM, there are three main problems [53] of interest that need to be solved, in order to obtain results:

- The Evaluation Problem

  Given an observation sequence O, and an HMM λ, which is the probability of the observations that have been generated by the model, ie, P (O | λ). With the resolution of this problem is possible to evaluate how well the model predicts a certain sequence of observations, it can thus draw conclusions about the most appropriate model for a given set values [53].

- The Decoding Problem

  In this problem, given a sequence of observations O, and an HMM λ, the objective is to find the most probable sequence of states which produces O [53].

- The Learning Problem

  Given an HMM model λ and a sequence of observations O, how should the model parameters (A, B, π) be adjusted in order to maximize $P(O|\lambda)$. The resolution of the problem results in optimizations of the model in order that the observations, or training set, are fully adjusted for the intended application [53].

5. DEVELOPED SOLUTION

With the main objective of being able of detecting and recognizing physical exercises, a solution to a wearable device was developed. Figure 26 presents a simplified workflow that includes the process of sensory data acquisition on the wearable device, the phase of information processing and the output results. In the next sections we provide a detailed description of each of the modules.

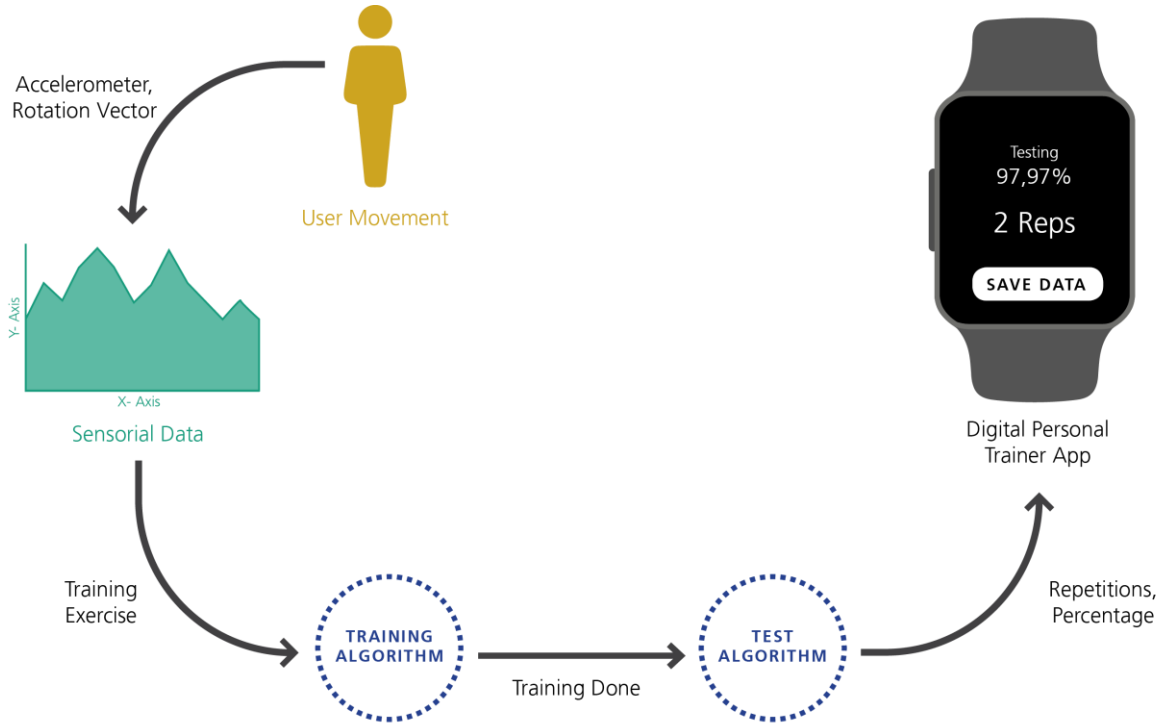


Figure 26: Digital Personal Trainer General Operation Diagram

5.1. USER MOVEMENT

The use of the application in a stable phase requires that reference data is acquired at the beginning. The user is then requested to perform a training exercise that characterizes the intended physical exercise. The following figure represents a possible exercise done by a user with this application.

Figure 27: Sample Exercise performed by the User with the Created Application [36]

Figure 27 shows that the exercises collected by this application can be made with or without instruments and tools used in gyms, as dumbbells and weight machines, making this solution more robust and practical for his target audience.

The user should perform the exercise that he wants to test, and the process of collecting data from the sensors to the creation of the reference exercise is done automatically by the application, making this operation fully transparent for the user.

5.2. SENSORIAL DATA

Initial data collected from the sensors is the key part of the operation. It is through this data that it will be possible to infer conclusions about the subsequent processes of this solution.

The user will then have to start the application in training mode and perform the exercise that he wants to test, so that the algorithm can store reference data for the calculations that will occur later.

During this training process, the application collects and stores the data obtained from the Accelerometer and Rotation Vector software sensor. Physical exercise acceleration data in the three main axes (x, y, z) is received from the Accelerometer while the Rotation Vector provides data relating to the rotation in yaw, pitch and roll.

Given that the data from the accelerometer suffers from noise, this limitation is overcome by the use of filters to smooth the received data, resulting in a generally more uniform waveform. In our implementation, we used an exponential smoothing filter [63] to successfully reduce the noise from the sensor and thus achieve a more uniform dataset.

This exponential smoothing method, often used for time series smoothing, associates exponentially smaller weights to older samples, or in other words, a relatively greater weight is given to recent samples in relation to previous ones [63][64]. The expression that describes this filter is given by:

$$s_t = \alpha \cdot x_t + (1 - \alpha) \cdot s_{t-1}, \qquad 0 \leq \alpha \leq 1 \tag{9}$$

Where x_t represents the obtained values from the accelerometer, s_t the data from the filter output, and α the smoothing factor, being that an α value of 1 causes that the filter output does not suffer from any kind of smoothing (input equal to output) and an α value of 0 makes

the maximum smoothing effect for the output, making it less responsive to recent changes [63][64].

Certain precautions were also taken into account in the reading process of the rotation values, in order to improve the subsequent exercise recognition processes.

As previously mentioned, the data from this sensor indicates the rotation of the device in the vertical (yaw), the lateral (pitch) and the longitudinal (roll) axis - the so-called Euler angles, received as angles of -180 ° to 180 °. As for the accelerometer, these rotational values have a limitation that, if not previously addressed in the receiving data process, may result in malfunctions of certain movements and exercises carried out with the smartwatch.

The referred limitation concerns the gimbal lock problem, which is a phenomenon that occurs when the orientation of the device is aligned by the three rotation axes (pitch angle of approximately 90 degrees), resulting in a loss of freedom for the system dimensions, which becomes able to move only in two dimensions, as can be seen in Figure 28.

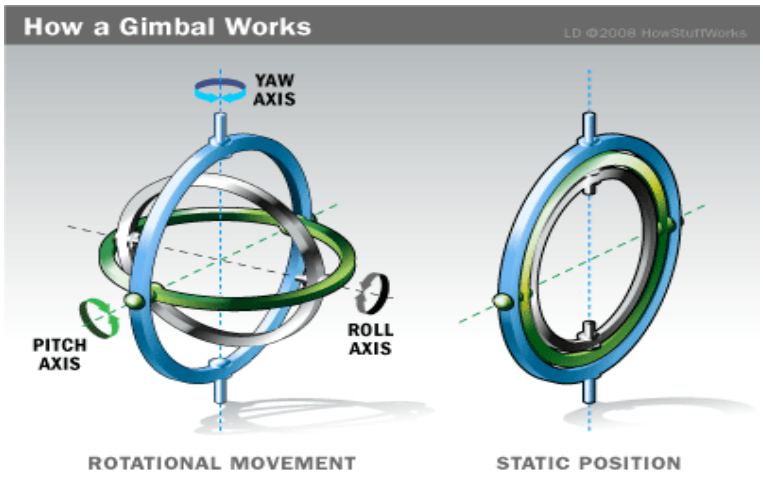


Figure 28: Gimbal Lock Representation [37]

As can be seen in the previous figure, a three-axis structure is used to illustrate the described problem, being that the mechanism named “Rotational Movement” describes the behavior of the system in normal operation, resulting in a rotation value for the grey gimbal, the green gimbal, and a last value for the blue gimbal. When the three gimbals are aligned, as shown in the mechanism named “Static Position”, this is in a gimbal lock state, which means that any change in orientation will result in rotation in only two gimbals.

One of the solutions used to overcome the problem of gimbal lock is the conversion of the obtained rotation angles in quaternions. The quaternion is a generalization of complex numbers, created by William Hamilton in 1843, and frequently applied in computer vision processes and three-dimensional space calculations [62].

The Quaternion representation extends the concept of rotational space from three to four-dimensions (s, x, y and z) and can be seen as the original device's orientation or the rotation applied to the object.

Therefore, and from the following two equations [62]:

$$i^2 = -1, j^2 = -1, k^2 = -1 \tag{10}$$

$$ij = k, jk = i, ki = j, ji = -k, kj = -i, ik = -j \tag{11}$$

A quaternion can be calculated by the following equation [62]:

$$Quaternion = s + xi + yj + zk, \quad \text{being } s, x, y, z \text{ scalar numbers} \tag{12}$$

For the rotation of a device, this value can be represented by a unit quaternion, ie:

$$Unit\ Quaternion = s + xi + yj + zk, \quad being\ s^2 + x^2 + y^2 + z^2 = 1 \tag{13}$$

Being this value obtained by converting the rotation angles (Yaw, Pitch and Roll), represented in Figure 29, by the following expression:

$$Unit\ Quaternion = \cos(\theta/2) + \sin(\theta/2)\, u_x + \sin(\theta/2)\, u_y + \sin(\theta/2)\, u_z k \tag{14}$$

Where $[u_x, u_y, u_z]$ is the unit vector which represents the rotation axis (X, Y, Z) and θ the angle of rotation (Figure 29).

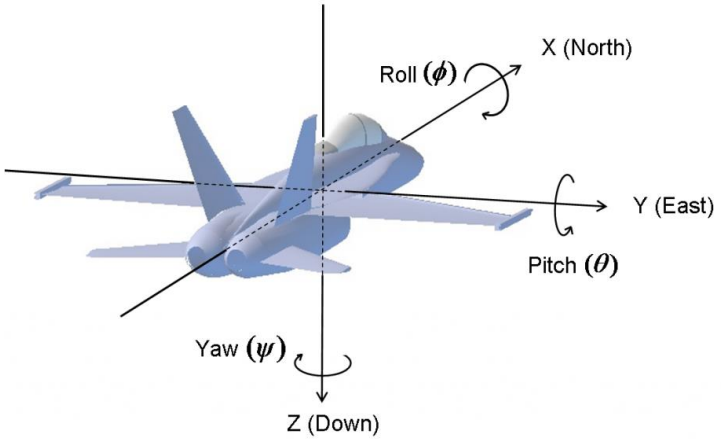


Figure 29: Representation of Unit Vector and Rotation Angles [38]

The rotations vector (x, y, z) with unit quaternions can be represented through the following orthogonal matrix [62]:

$$\text{Being } Q = s + xi + yj + zk, \text{ the Unit Quaternion}$$

$$R = \begin{bmatrix} s^2 + x^2 - y^2 - z^2 & 2xy - 2sz & 2xz + 2sy \\ 2xy + 2sz & s^2 - x^2 + y^2 - z^2 & 2yz - 2sx \\ 2xz - 2sy & 2yz + 2sx & s^2 - x^2 - y^2 + z^2 \end{bmatrix}$$

The values of the rotation vector can be obtained by multiplying the matrix R by the unit vector $[u_x, u_y, u_z]$, resulting in an easier representation without the introduction of imaginary numbers.

Therefore, the quaternions are like a mathematical way to represent rotations in fourth dimensional space, which have the main advantage, regarding this thesis, as a solution for the gimbal lock problem, giving the device complete freedom in orientation changes

To solve the problem of Gimbal Lock, Android provides a high-level method to automatically convert the sensor rotation values to quaternion units, as shown in the following command.

```
SensorManager.getQuaternionFromVector(qua, values);
```

The function getQuaternionFromVector causes the device's SensorManager to convert all values from the software sensor for a range of normalized quaternion, using the previous explained matrix calculations.

The overall processes of collecting sensor data in a platform such as Android Wear, is identical to the one used in a traditional Android platform, and implements the following phases:

1. Initialize the Android Sensor Manager

```
mSensorManager = (SensorManager)
this.getSystemService(SENSOR_SERVICE);
```

2. Choose the sensor to be obtained data

```
mAccelerometer =
mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

3. Initialize the responsible class created for the sensor listening process

```
sensorListener = new MySensorListener(blockingQueue);
```

4. Register the sensor with its sampling rate and start listening data

```
mSensorManager.registerListener(sensorListener, mAccelerometer,
SensorManager.SENSOR_DELAY_FASTEST);
```

Data can be collected at the declared class used for initialization and registration directives or in an external class, making the development process much more distributed and organized, by initializing the MySensorListener class:

```
public class MySensorListener implements SensorEventListener {}
```

In this case, through the use of the command *implements* and invoking the *SensorEventListener* interface, the system will provide all the methods to the class *MySensorListener* so that it is able to get the values from the sensors events. One of the available methods for receiving new values from the sensors is:

```
void onSensorChanged (SensorEvent event)
```

How is possible to understand by the name of the method, this occurs every time a new event is detected on the sensor, being that this event can be detected in the same sensor as long as the new reading takes place in a new timestamp.

Using this method data can be collected at a previously chosen sampling rate. When this task is performed for multiple sensors, it is only necessary to wait until the SensorEvent occurs and by a call of a simple command, such as *sensor.getType*, ask to which sensor system is treated the occurred event.

```
event.sensor.getType()
```

This command returns the name of the type of sensor where the event occurred, in which case could be one of two types:

- Sensor.TYPE_ACCELEROMETER
- Sensor.TYPE_ROTATION_VECTOR

There are other methods that can be associated with the sensor event in a way to get more information about these devices:

- `Sensor.getName()`

  Returns a string containing the sensor name.

  Ex: BMA150 3-axis Accelerometer.

- `Sensor.getPower()`

  Returns the consumed current, in mA.

  Ex: 0.45 mA.

- `Sensor.getVendor()`

  Returns the sensor vendor name.

  Ex: Motorola.

The reading and collecting data processes also required other care for the acquisition of data, because the multi-threading approach benefits the device segmentation commands in parallel, but introduces limitations on data competition, a problem that will be referred in the following sub-chapter.

5.3. TRAINING ALGORITHM

The training algorithm is one of the most important processes of the solution, as it is with the results of this process that it is possible to obtain the reference data of the physical exercise that describes what can be considered a perfect execution of a physical exercise. To guarantee the ideal conditions, this training process can be done with the user accompanied by the personal trainer or a physical therapist.

Using the same approach as the general diagram of the created solution, is shown in Figure 30, now in a more technically way, a possible operational diagram of the training algorithm, and then will be explained all the constituent processes and blocks, connected to the Android platform.

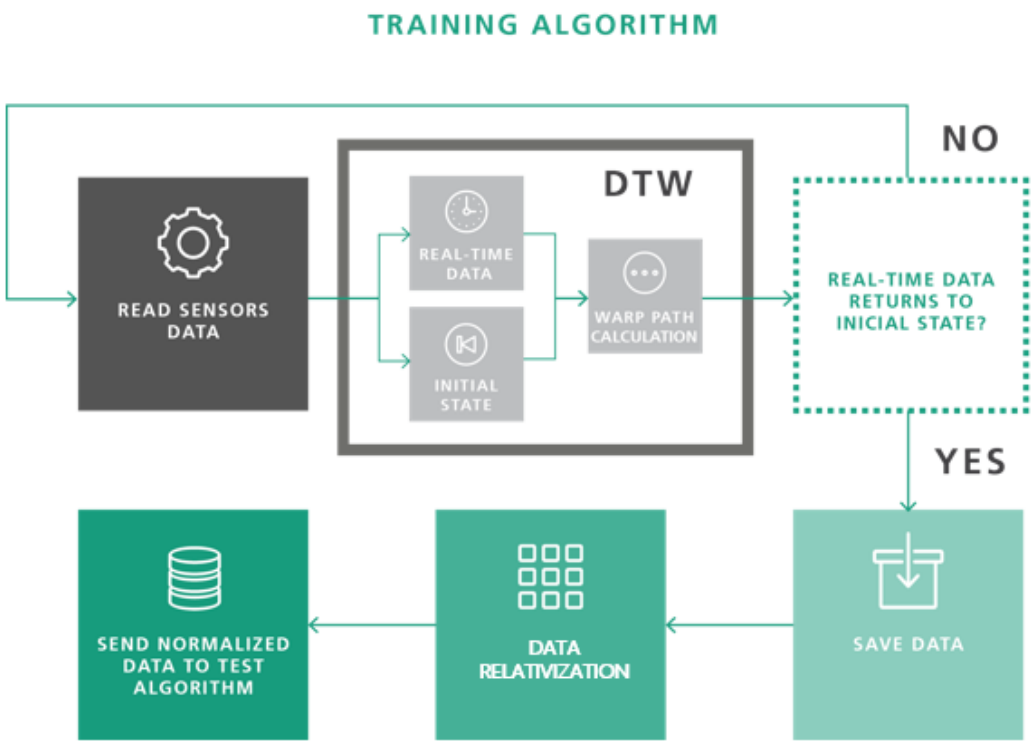


Figure 30: Representative Flowchart of the Training Algorithm Operation

The first task is obtaining the sensor data:

```
public class MySensorListener implements SensorEventListener {}
```

The class "MySensorListener" is responsible for collecting the data from the sensors and send it, through an own structure of the Android platform, called BlockingQueue.

The BlockingQueue is a structure used in data communication processes that, as the name implies, implements a kind of Queue with blocking capabilities in reading and writing processes. This structure supports operations that wait for responses of the queue itself indicating whether or not the space to insert/remove an element of it.

The implemented methods by this structure has four main modes of operation such as:

- Launch a code exception

  Ex: add(), remove() e element())

- Return a special value, usually zero or false

  Ex: offer(e), poll() e peek()

- Block the operational thread, indefinitely, until the operation is successful

  Ex:put() e take()

- Wait a limit time to end the operation before giving up

  Ex: offer(e, time, unit) e poll(time, unit)

By the methods mentioned above, it is possible to see that a BlockinQueue allows operations that insert, remove or examine data. For all these and other features, using this structure is safe for applications that involve multi-threading processes, and was specifically created for this purpose.

The use of such a structure is important in an application of this type due to the concurrent processes that run in parallel, because will have to be some method to execute some kind of blocking mechanisms between the data processing, so as not to occur competition errors which can influence the functioning of the system itself.

In a practical way, a BlockingQueue can be declared as follows:

```
protected final LinkedBlockingQueue<ValuesSensorObject> blockingQueue;
```

In this case, the `blockingQueue` variable is a variant of this structure, which in addition to all the features explained above, implements linking processes between all the data that make up the queue, providing greater freedom of access and modification of these same data.

The ValuesSensorObject class is responsible for organizing the blocking queue, which is constructed by:

- Values - Array of values where the readings in the queue will be saved;
- SensorType - Integer that represents the type of sensor: "1" identifies the Rotation Vector and "2" identifies the Accelerometer;

After the initial reading and processing of the data, it is necessary to send these values to the blocking queue, by simply filling an object of the `ValuesSensorObject` class with the values and the type of sensor, and through the `offer()` method, add the resulting object to the queue, as shown in the following commands:

```
ValuesSensorObject object = new ValuesSensorObject(Values, SensorType);
blockingQueue.offer(object);
```

The reception of these data is done in the class responsible for the Training Exercise, in specific structures of the Android Platform, called `LinkedLists`. These structures allow the user to initialize a list of values to store data without a predefined size and with the possibility to access different indexes positions of the list, making the allocation of memory transparent for the programmer/developer.

The training class is organized into three main threads:

1. Android Main Thread – Responsible for managing all the application's User Interface processes;
2. Sensors Thread – Responsible for saving all data from the blocking queues in clear structures for the class to operate and send to the Test class;

3. DTW Thread – Responsible for processing all the Dynamic Time Warping calculations and get all the conclusions related to trained exercise;

Continuing with the explanation from the flowchart of Figure 30, the data are currently prepared to be computed by the Dynamic Time Warping algorithm, and as shown in Figure 31, the comparison of values will be held between the data that the class is receiving at the moment and the data that were obtained at the initial instant that the user started the exercise.

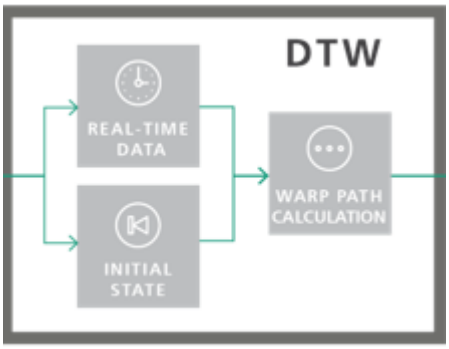


Figure 31: Time Series comparison using DTW

The implementation assumes that physical exercises begin and end in the same position, which is not a universal truth for all exercises, and may be a limitation of the current version.

The DTW module runs uninterrupted and performs continuous calculations seeking the similarity between the relative data from the sensors received at that time and a set of data lists, which are the recorded values identifying the exercise.

The process of relativizing the data from the sensors is obtained by subtracting all the values which are set by the first value, as shown in the following commands:

```
  accelx = new Float[LLaccelxtrain.size()];
for (int i = 0; i < LLaccelxtrain.size(); i++) {
  accelx[i]=LLaccelxtrain.get(i)-LLaccelxtrain.getFirst();

}
```

Two distinct processes can be identified in the relativization of values:

- *accelx* - Initialization of a variable with the type and size of the data relative to the linked list of values received in real-time (`LLaccelxtrain`);

- `for()` - Cycle that goes through all the values of the linked list and associates each one to an index of the *`accelx`* variable;

Through this process it is possible to transform the absolute values from the sensors into something relative and reference, which can be reused by the user to test the same exercise in other situations and in different directions, so there is no need to perform a training exercise every time that it needs to be tested, making it a more robust and automated solution.

The data from the Rotation Vector Software Sensor and Accelerometer are relativized in real-time, resulting in six time series (three sets representing the device's orientation in yaw, pitch and roll, and the remaining three sets to represent the acceleration values x, y and z) that will be compared with other six series representing the initial state of exercise.

After the DTW calculations determine that the exercise returned, approximately, to his starting position, it is activated a command that indicates to the Sensors Thread that can send the resulting data to the Test class.

When the DTW modules detects that the exercise returned, approximately, to its starting position, the Sensors Thread is informed and data is stored in a special Android class - `SharedPreferences` - that allows storing and retrieving persistent data over several application sessions and even if the device it turned off. The main advantage in using this class for the data storage is the ease of use, as shown by the following commands:

- Initialize a variable *SharedPreferences* interface and subclass *Editor*:

```
SharedPreferences sharedPrefs =
PreferenceManager.getDefaultSharedPreferences(getBaseContext());

SharedPreferences.Editor editor = sharedPrefs.edit();
```

- Store the desired data as a key-value pair:

```
editor.putBoolean("key_name1", true);     // Saving boolean
editor.putInt("key_name2", "int value");  // Saving integer
editor.putFloat("key_name3", "float value");  // Saving float
editor.putLong("key_name4", "long value");    // Saving long
editor.putString("key_name5", "string value");// Saving string
```

- Save the performed changes to *SharedPreferences*

```
editor.commit();
```

In a similarly way, retrieving the data previously stored can be done as:

- Initialize the Interface variable

```
SharedPreferences sharedPrefs =
PreferenceManager.getDefaultSharedPreferences(getBaseContext());
```

- Read the saved data

```
boolean user= pref.getBoolean("key_name1", true); //getting boolean
int pageNumber=pref.getInt("key_name2", 0);      // getting Integer
float amount=pref.getFloat("key_name3", null);   // getting Float
long distance=pref.getLong("key_name4", null);    // getting Long
String email=pref.getString("key_name5", null);   // getting String
```

In addition to all these features, `SharedPreferences` allows data to be represented using JSON (Javascript Object Notation), which helps creating an abstraction for the type of data to be sent for storage. Using libraries as Gson, which allow the conversion of Java objects to JSON and vice versa, data to be interpreted by `SharedPreferences` can be easily interpreted by the system as strings:

```
String azimuthjson = gson.toJson(azimuth);
editor.putString("azimuth2", azimuthjson);
```

By having this abstraction, the type of data to be stored by `SharedPreferences` are all stored as strings, being only need to convert these data at the time that it wants to read it, as shown in the following command:

```
String jsonazimuth = sharedPrefs.getString("azimuth2", null);
Float[] trainazimuth = gson.fromJson(jsonazimuth, Float[].class);
```

As it is possible to see, the methods implemented by Gson library, provide all the features for the conversion of String to the primitive data type, in this case an Array of Floats. The use of an interface as SharedPreferences becomes more intuitive for the developer and allows to store and retrieve persistent data in a simple way.

After all these processes the Training algorithm terminates its functions, and as indicated by the last block from the flowchart of Figure 30, the relativized data sets are stored for later use by the Test algorithm. In this last phase, all secondary Threads that may still be processing data are interrupted, and all sensors listener processes are finished, so as not to remain any residual processes to be computed when starting the Test algorithm and especially, for a good management of the smartwatch resources.

5.4. TESTING ALGORITHM

The Test algorithm is the process where the results from the solution itself are calculated, ie it is at this stage that the pattern recognition algorithm uses all its capabilities to find similarities between the reference data obtained in the Training process and the data that the user is performing in real-time.

In this chapter are reported all aspects and approaches used in the development that led to the creation of a process of recognition and evaluation of physical exercise, in a more technical and targeted way, directed to the Android Wear platform.

Figure 32 presents the flow chart for the operation of the test algorithm that will be used to identify an exercise through a real time pattern recognition algorithm.

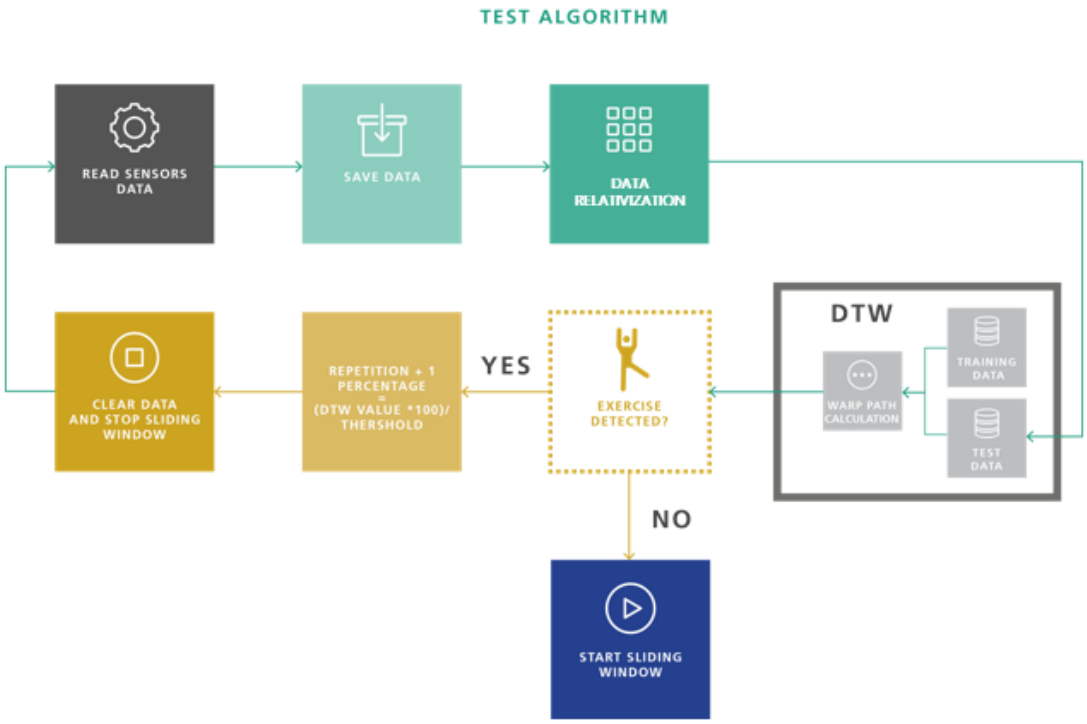


Figure 32: Representative Flowchart of the Testing Algorithm Operation

By analyzing the diagram above, it is possible to find some ways similar to the Training algorithm, as the case of reading and processing of sensory data, although the overall operation of the algorithm presents different features.

Assuming this algorithm collects the orientation and acceleration data from the smartwatch in the same way as done in the training process, and so the size of the collected data exceeds

a minimum threshold, the algorithm saves the data and performs the relativizing calculations, as shown in the previous sub-chapter.

After this process, the algorithm it is with the conditions required to perform the DTW calculations between the reference values and the values from the tested exercise, due to certain factors, such as:

- Both time series (training and test) have a non-null size and that does not exceed the acceptable limit for the DTW computing process;
- Both time series (training and test) passed from a relativization process and are at the same scale, in order to be compared in a consistent manner;

The similarity calculation is made between the six training time series and the six test time series, by calling the class responsible for implementing the DTW:

```
DTW dtw = new DTW(trainseries, testseries);
```

Next, represented by the "Exercise Detected" block, the algorithm enters in a recognition process, trying to find the minimum distance between the two exercises, which will involve the output of the DTW, for each of the comparisons effected is below a certain threshold. After this step the algorithm validates if the exercise that the user is doing has a pattern similar to the trained exercise, but still does not explain the user of this information in the smartwatch screen, waiting that the distance between the two exercises are the minimum possible to reveal the detection and percentage of similarity that exists between the exercise performed at the time and the previously trained exercise.

The process of comparing the two exercises, until a minimum distance is shown, indicates that the algorithm not only detects the exercise, but after this result, remains a computing similarity calculations until finding a minimum distance between the two. In practical terms, this process indicates that an exercise can start being detected soon after it was made only part of it, but if the exercise continues to be performed properly until the end, the distance will gradually decrease and the algorithm will remain listening until this value increase again, that is, the end of the exercise or a different exercise from the previously trained.

If the DTW calculation results in a greater value than the threshold and no exercise are detected, it activates a sliding window that runs the reading data and updates the time series that should be compared until the output of this process is less than the previous threshold, or if the test data size exceeds a certain limit.

The use of a mechanism such as a sliding window is important, as there are many factors that may differentiate the training exercise from the test exercise, even when the two exercises are the same:

- Exercise Rhythm - The training exercise can be performed in a faster or slower way than the test exercise.

- Exercise Amplitude – The user can perform only a part of the exercise previously trained.

The fact that the DTW algorithm is not affected by the speed or range of time series, makes the comparison of the exercises independent of these factors and can overcome this problem.

Through the use of a sliding window, it is thus possible to scroll and update the values of the test series, in case the two exercises are offset in time or in amplitude, in a way to ensure that reference pattern that is sought is found, and the processing carried out by the DTW does not exceed a reasonable limit of the smartwatch capabilities, which could jeopardize the functioning of the solution.

After this process, and detected the exercise, the algorithm proceeds to update the values of the repetitions and the percentage of similarity between the trained and tested exercise. To ensure that the data from the next exercise will be read in the right way, and that the DTW will not have too much data to process, data queues are cleaned:

```
LLaccelxtest.clear();
```

After all the testing process, and following the diagram of Figure 26, it is now discussed in the following subsection, the operation of the last block, which corresponds to the application itself, this time in an aspect more related to the usability parameters, and its user interface.

5.5. DIGITAL PERSONAL TRAINER APP

As a final result of all processes and algorithms described above, is the created application, which in a simple way for the user, describe some relating data to the exercise that the user his performing at this time.

Figure 33 represents the time flow of steps for operating the created application, describing each screen is the user throughout the use of the same.

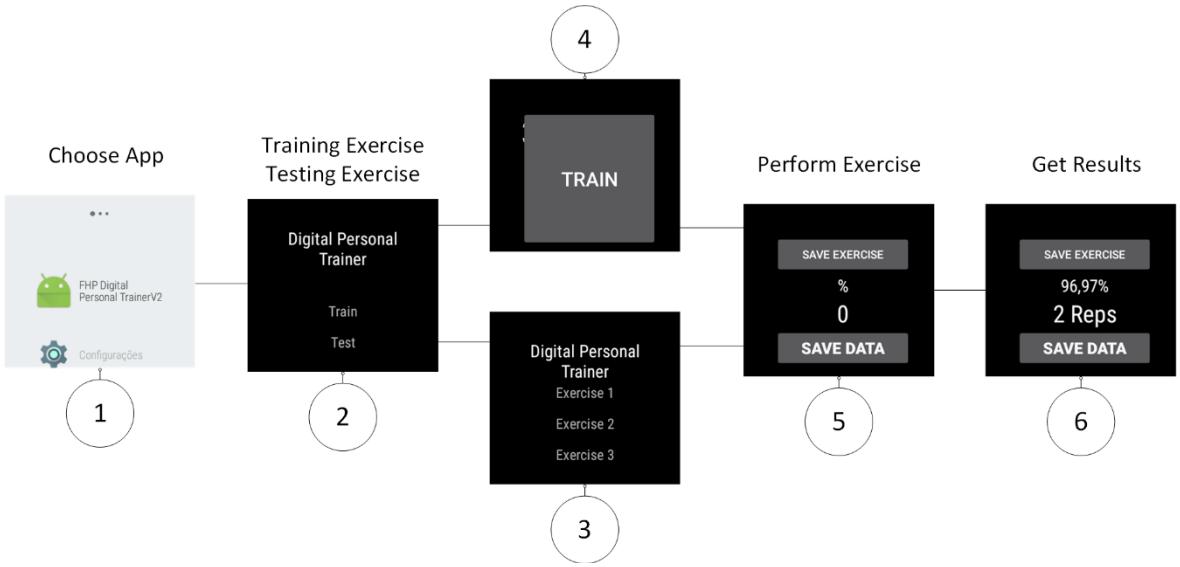


Figure 33: Set of possible screens of the created application

In the initial screen (Screen nr 2), the user has two possible ways to use the application:

1. Choose to train a new exercise, represented by screen 4
2. Testing a previously trained exercise, represented by screen 3

If the user chooses the training of a new exercise, he will find the 4th screen, shown in Figure 33. The created User Interface for the training process has gone through several updates and processes, to arrive at an approach that is considered to be the simple and automatic. Thus, after the user is in 4th screen he is asked to click on the "Train" button, which will trigger all the software processes for the collection of the reference data, and show the user the following screens.

Figure 34: Set of screens which form the training process of an Exercise

Through the diagram of Figure 34, it is possible to understand how it is treated the training process in terms of UI, being that, after the user indicates that he wants to perform a training exercise, through the activation of the "Train" button, the device starts a countdown of 3 seconds, allowing the user to take the starting position of the exercise he wants to train, and after this 3 seconds, the screen indicates that exercise can begin to be carried out with the "Start" message. After the exercise has been done the right way, the screen launches a new warning with the message "Train Saved", indicating to the user that the entire training process went as it was supposed to, and can now move on to the testing process.

The detection of the end of the training exercise, and later passage to the testing procedure is carried out automatically by the application, without having to be any kind of user interaction to indicate this fact, thus making this process more automatic and user- friendly.

The 5th and 6th screens of Figure 33 are used during the test phase and enable providing real time information to the user, and saving data or even the trained exercise itself. This includes the number of repetitions done, the similarity with the previously defined exercise (the trained exercise), and even a log of all sensory data obtained during the tests.

The first button, called "Save Exercise," is at the top of the screen and specifically serves to save the training exercise in the application, so that the user can test itself at a different time the same exercise without having to train. In technical terms, this process makes use, once again, from the offered features by SharedPreferences to keep the sensory data, and link them to a list.

The second button is used to store all data obtained either in the training process, as in the testing process on files. By clicking the "Save Data" button, the user indicates the application that he wants to store the collected sensor data in the training process, the sensor data collected in the testing process so far, and the results of repetitions and obtained percentages using for this the process of writing files to the internal memory of the smartwatch. This process allows the user, but mainly to the developer, perform a more critical analysis of the

results and compare each detected repetition by the algorithm with the sensor data, and thus ascertain and weaving conclusions on the operation of the application. Technically, the process of reading and writing files to SmartWatch may be effected in different ways, and in this application was used a library established by the Android, which already implements most of the methods for this process. A list of possible commands to elucidate the mechanical reading/writing files, this platform presented in the following section:

```
File myFile2 = new
File("/sdcard/Train/"+currentTimeStamp+"/"+"Accelerometer_train.txt
");
if (!myFile2.exists()){ myFile2.mkdirs();}
myFile2.createNewFile();
FileOutputStream fOut2 = new FileOutputStream(myFile2);
OutputStreamWriter myOutWriter2 =new OutputStreamWriter(fOut2);
myOutWriter2.append("Time,Azimuth,Pitch,Roll"+"\r\n"+sbaccelraw);
myOutWriter2.close();
fOut2.close();
```

This process requires the app to have I/O privileges defined in the Application Manifest file:

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

Regarding the received information in real-time from the testing process, in technical terms, notifications of repetitions of the exercise, performed by the user, as well as their percentage quantification are carried out on a secondary Thread, and therefore need to call special methods for "transfer" this information to the Main Thread, the only Thread with privileges processes to the User Interface. A possible command that describes this process is as follows:

```
MainActivity.this.runOnUiThread(new Runnable() {
    @Override
    public void run() {
mRepetitionText.setText("" + counter + " Reps");
textviewdistance.setText("" + String.format("%.2f",
textviewpercentage) + "%");

    }
});
```

By calling the "runOnUiThread" method, the system launches a warning that all the code that is present in this cycle will be processed by the Main Thread, and thus ensure that all

operations involving components calls that are directly linked with the device User Interface (TextViews, buttons, lists, etc .) are held in the corresponding thread.

As can be seen in the above command, in the cycle are two commands that perform the operations related to the UI components, in this case, writing TextViews notifications that are displayed on the screen smartwatch.

A list of stored exercises that can be tested at any time without the need for prior training is also available from screen nr 5. Through this implementation, it is possible to make the application even more automatic and transparent to the user, requiring only that he trains once the desired exercise, keep all reference data of the same, with the processes explained above, and the application itself adds the exercise to a list that, for user convenience, only have to choose which exercise he wants to test and start running it.

After all this theoretical and practical description of the created solution, it remains to emphasize that all described processes and operations make use only of the smartwatch capabilities, and there is no communication with a smartphone, as this was a differentiating factor of this dissertation, and one of the main goals.

6. TESTS AND RESULTS

This chapter reports all the tests made to the solution as well as the obtained results. The device used to carry out the entire development and testing processes, as well as a comparison with other devices of the same platform, are presented. The main advantages and disadvantages of the choices made for the implementation of the project are discussed.

The testing protocol, defined taken into account the objectives of this dissertation, is also presented.

6.1. TESTING

Tests were conducted with several people in order to validate the solution proposed in this dissertation.

Regarding the hardware used in this project, the options available for the development were composed by three smartwatches:

- Motorola Moto 360 (1st Generation)
- Lg G Watch W100

- Sony SmartWatch 3 SWR50

An investigation was then drawn up, for the available hardware, taking into account the specifications of each one, under various factors that are considered important to the operation of such an application, and after gathered all the information relating to each of the devices, Table 2 displays a comparison of different specifications from the three available smartwatches.

Table 2: Comparison between the Available SmartWatches

	Motorola Moto 360	Lg G Watch	Sony SmartWatch 3
CPU	1.0 GHz	Quad-core 1.2 GHz Cortex-A7	Quad-core 1.2 GHz Cortex-A7
Memory	Internal: 4GB RAM: 512 MB	Internal: 4GB RAM: 512 MB	Internal: 4GB RAM: 512 MB
Sensors	Accelerometer, Gyroscope, Magnetometer	Accelerometer, Gyroscope, Magnetometer	Accelerometer, Gyroscope, Magnetometer
Battery	Li-Ion 320 mAh	Li-Ion 400 mAh	Li-Po 420 mAh

In order to create the less favorable environment, the Motorola device, which has the lowest processing power, was chosen. As one of the main objectives was to develop a solution that could process all the data independently, having it validated this way would assure that it would easily work in other wearable platforms. In fact, later tests with the LG equipment showed that the only adaptation needed was down-sampling data acquisition not to force the DTW to compute calculations on too long time series.

Figure 35:SmartWatch Motorola Moto 360 (1st Generation) [66]

After this theoretical survey about the existing devices, and all the development made in the Moto 360 smartwatch, it was elaborated a testing protocol to which the volunteers, who wished to test the application, would have to fulfill, in order to validate the list of objectives proposed in this dissertation.

This protocol consists of the execution of two distinct physical exercises:

- Biceps Curl

Figure 36: Biceps Curl Exercise Representation [36]

- Chest Fly

Figure 37: Chest Fly Exercise Representation [39]

Volunteers were asked to perform the same set of tasks:

1. Training Exercise;
2. Perform 5 correct repetitions of the exercise;
3. Perform 5 repetitions of the exercise, alternating between a correct repetition, and the execution of only a portion of the exercise;
4. Stand still for 5 seconds.

This testing protocol enables validating the following procedures:

- Corrected Execution of the Training Process;
- Detection of the exercise repetitions;
- Analysis of the performance of the training
- Presence or absence of false positives in the performed repetitions;

Thus, they were prepared a series of tests performed by 9 subjects of both sexes, with different ages and with different physical conditions. The obtained results are then presented in the following subsection.

6.2. RESULTS

In this subchapter are obtained the main conclusions regarding the results of the tests carried out with the created application, and therefore this is an important part of this thesis, in that is in this section that the validation process of the proposed objectives is executed.

In the following tables are presented statistical data relating to the performed repetitions in each of the exercises, including the average and standard deviation of the percentages obtained in each of the repetitions.

Table 3: Results from Performing of Exercise 1

	5 Correct Repetitions			5 Alternating Repetitions	
Repetitions	Average (%)	Std. Deviation	Repetitions	Average (%)	Std.Deviation
1	90.2	2.66	1	91.3	3.90
2	90.8	3.04	2	66.0	8.30
3	91.8	2.71	3	92.5	2.17
4	91.4	3.60	4	72.2	8.24
5	90.5	3.30	5	91.2	2.50

Table 4: Results from Performing of Exercise 2

	5 Correct Repetitions			5 Alternating Repetitions	
Repetitions	Average (%)	Std.Deviation	Repetitions	Average (%)	Std.Deviation
1	93.78	1.48	1	90.78	2.04
2	92.89	3.21	2	75.89	7.06
3	91.78	2.68	3	93.14	2.73
4	92.25	1.98	4	75.86	8.78
5	92.00	2.58	5	90.29	3.40

As it can be seen, when users performed a series of 5 correct repetitions, relating to the trained exercise, the percentage corresponding to the similarity between the two exercises, presented values between 90% and 92%, indicating that the algorithm has recognized the performed repetitions as very similar to the trained exercise. Regarding the standard deviation, the values are relatively low indicating that, overall, all repetitions were carried out in the same range of values.

For the second series, volunteers had to perform five repetitions that included a correct repetition followed by an incomplete execution of the exercise. The first, third and last repetitions are related to the execution of a complete repetition, as evidenced by the high percentage of recognition (~ 90%), while the second and fourth repetitions indicate the execution of only half, or part, of the trained exercise, which is also recognized by the application with a lower percentage (~ 60% -70%), demonstrating the good recognition rate of this solution.

Regarding the standard deviation, it is possible to see a high value in relation to the two repetitions representing the execution of only part of the exercise, an aspect easily explained by the fact that the execution of this exercise is relative to each person, that is, as there is no certain limit to indicate what is the execution of only part of the exercise, each volunteer that performed this type of repetition, did the exercise in its own way, allowing that the range of data was higher, between 55% and 75 %.

Although the main results can be said to be reasonable, there was a need to be performed a second test, with 9 volunteers, where at the end of each session were saved all the obtained data, in a way that can perform a deeper evaluation of the application operation, being that, the used testing protocol also suffered a minor change, so that it can highlight certain aspects of the application, in a more uniform way, so the standards for the tests pass to be:

1. Training Exercise;
2. Performing 5 corrected repetitions of the exercise;
3. Performing 10 repetitions of the exercise, alternating between a correct repetition, and the execution of only a portion of the exercise;
4. Stand still for 5 seconds;

With this change to the protocol, it is possible to compare and demonstrate, in a more complete way, the quantitative differences between each repetition of the performed exercise.

In Table 5 and Table 6, using the same approach as the first test, presents the obtained results in each of the repetitions performed with the two exercises.

Table 5: Results from Performing of Exercise 1

	5 Correct Repetitions			10 Alternating Repetitions	
Repetitions	Average (%)	Std.Deviation	Repetitions	Average (%)	Std. Deviation
1	95.90	4.87	1	97.02	1.95
2	96.59	1.48	2	63.13	9.44
3	96.13	1.35	3	96.72	1.56
4	94.35	1.63	4	73.54	7.42
5	94.12	2.00	5	96.26	2.18
			6	71.86	7.78
			7	93.87	3.57
			8	61.49	14.44
			9	92.80	5.56
			10	63.81	14.39

Table 6: Results from Performing of Exercise 2

	5 Correct Repetitions			10 Alternating Repetitions	
Repetitions	Average (%)	Std. Deviation	Repetitions	Average (%)	Std. Deviation

1	97.41	2.69	1	98.44	0.41
2	97.74	1.02	2	68.99	10.97
3	96.95	2.10	3	97.52	1.13
4	95.72	3.25	4	67.80	9.47
5	91.53	6.35	5	96.40	1.84
			6	68.78	11.11
			7	94.65	3.82
			8	71.49	10.82
			9	92.61	6.05
			10	63.55	8.89

The change in the testing protocol, from 5 alternating repetitions to 10 alternating repetitions, was mainly due to make the whole process more consistent with each other, and get to infer conclusions regarding the repetitions running only part of the exercise, seen that in the first test only was possible to check this for two repetitions.

Changes in the algorithm of the application have brought advantages and corrections to the experienced limitations in the first test, as the improvement in sensitivity training process meant that the volunteers did not have to run several times the training exercise, until it is considered the correct, as well as the improved recognition of repetitions, within the process tests, led to the appearance of false positives was practically null. These corrections can be viewed, in a certain way, with the logging of sensory data, obtained in each session of tests, a factor which proved to be important to enhance the development of the solution itself.

With the introduction of the logging functionality of all obtained data, it is also possible to draw a number of conclusions regarding the trained exercise, and the very own repetition recognition process. It is presented, in the following figures, in a more graphic form, the

obtained data in the training and testing process of the performed exercise by one of the volunteers.

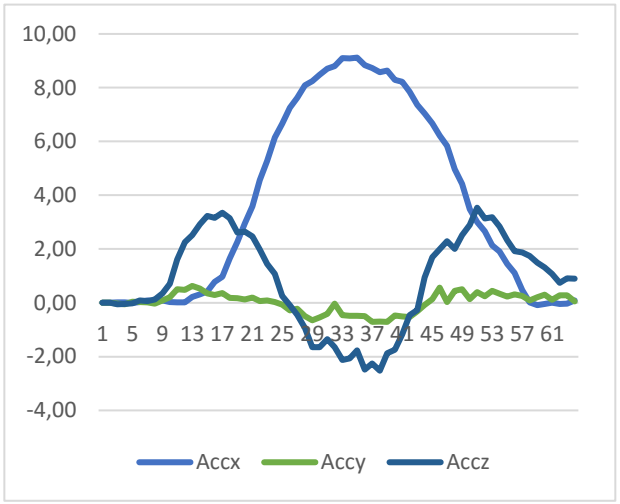


Figure 38: Sensory data from the Exercise Training Process

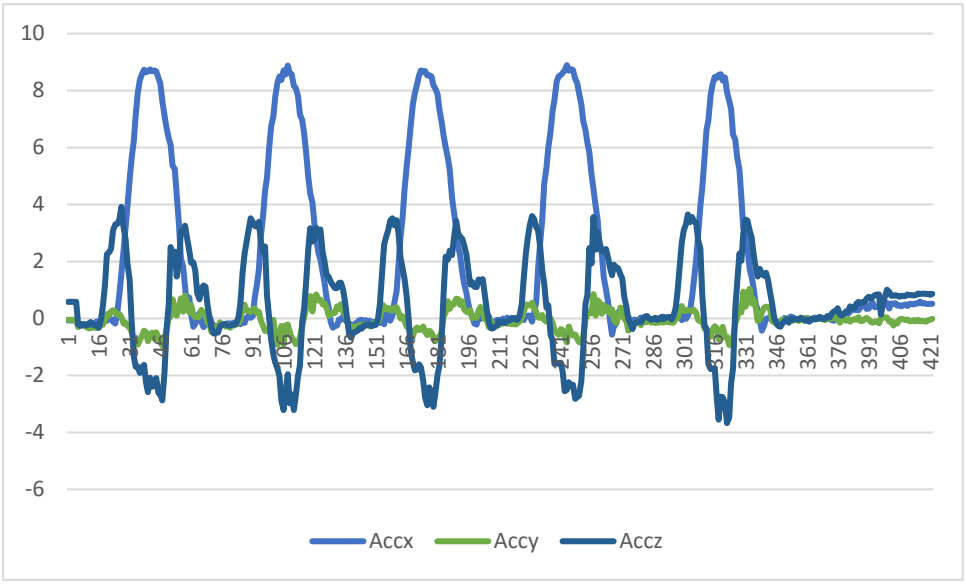


Figure 39: Sensory data from the 5 repetitions of the Exercise Testing Process

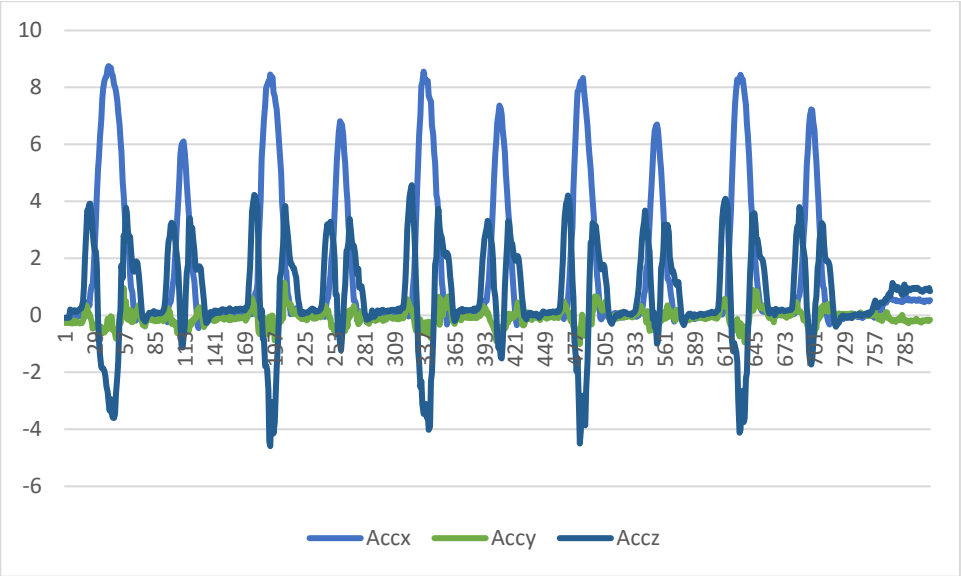


Figure 40: Sensory data from the Testing Process of 10 alternating repetitions of the exercise

Figure 38, Figure 39 and Figure 40 represent readings, only from the Accelerometer, for training exercises, 5 repetitions test and 10 alternate repetitions test, respectively. After further analysis, it is possible to show the pattern that results from each repetition of the exercise, a pattern that has to be, in a way, similar to the waveform of Figure 38, since this is the result of the training exercise, so the reference pattern for the application of the algorithms use in comparisons of test exercises.

In Figure 39 is possible observe 5 repetitions of the exercise trained in Figure 38, represented by each of the “slopes" present in the sample, being observed that the amplitude of the "slopes" is practically the same for all repetitions, as well as the scale of the movement, over time, albeit with minor differences, is also virtually identical in all repetitions of the exercise.

In relation to Figure 40, regarding the 10 alternate repetitions, are observed differences along the represented waveform, for some repetitions amplitude, being this amplitude that evidence the quality of the performed exercise. In this particular case, the 2nd, 4th, 6th, 8th and 10th repetitions exhibit a maximum amplitude of 6-7g while the remaining repetitions are around 8.5-9 g, and this factor, combined with the readings of values device rotation, and performing the DTW calculations, result in different percentage values of the exercise, for when the repetitions amplitudes are different of the reference range.

By analyzing the graphs of Figure 39 and Figure 40, it is possible to complete the proper functioning of the created application, as regards the pattern recognition process, or in this case of study, the recognition of repetitions of physical exercises.

In order to finalize this document, is presented in the following chapter, all the conclusions reached in the realization of this project, and put in perspective a possible future work that may add value to the created application.

7. CONCLUSIONS

The main objective of this thesis was to develop a solution for a wearable device that could be used to verify the accuracy of physical exercises. The prototype implements all the functionalities defined at the beginning of this project and results show a good accuracy on detecting and quantifying previously trained physical exercises.

Testing was performed using a device with low processing capabilities in order to ensure that the solution would have a broad market and could be deployed in a large number of hardware devices. To guarantee this, the developed algorithm uses parallel processing and threading. All the tests showed that the approach used was valid and the software could be used in Motorola 360 smart watch, one of the less sophisticated devices in the market currently.

To our knowledge, the proposed solution is innovative when compared to the existing products that aim a similar application, given that it is a standalone solution – there is no need to use a smartphone for processing the data as it happens in the other products.

Due to the fast technological evolution, it is quite likely that new devices with additional processing power will be made available by manufacturers, making it possible to easily introduce new functionalities in the current software version making it more versatile and

complete. We foresee application of our work in areas as sports, health, ambient assisted living, etc.

Some improvements, that would enable creating a final product in terms of usability and functionalities, are:

- Detect a physical exercise that does not start and end in the same position.
- Develop a better User-Interface and User-Experience of the Android Wear App.
- Test the solution in individuals with reduced mobility (Physiotherapy)
- Incorporate activity monitoring and fall detection algorithms in the Android Wear App, in order to create a complete and better experience.

References

[1] TrackMyFitness – "Don't Just Track Steps, track my fitness", http://trackmy.fit/ , [Accessed: March 2016]

[2] DoFit – "Smartwear Fitness", http://dofit.co/, [Accessed: June 2016]

[3] Moov – "The multi-sport wearable coach that talks to you as you work out", http://welcome.moov.cc, [Accessed: March 2016]

[4] Moov Now Multi-Sport Wearable Coach, https://www.athleteshop.com/moov-now-multi-sport-wearable-coach-fusion-red, [Accessed: June 2016]

[5] Jawbone UP3 – "A Smarter Activity Tracker for a Fitter You", https://jawbone.com/fitness-tracker/up3, [Accessed: June 2016]

[6] Microsoft Band, https://www.microsoft.com/microsoft-band/en-us, [Accessed: June 2016]

[7] CHAKRAVARTHI, M., Tiwari, R., & Handa, S., "*Accelerometer Based Static Gesture Recognition and Mobile Monitoring System Using Neural Networks*", Procedia Computer Science 70, 2015, pp. 683-687.

[8] YAMADA, A., & Keiji, S., "*Generalized Learning Vector Quantification*"

[9] K-curvature Algorithm, http://www.tmroyal.com/a-high-level-description-of-two-fingertip-tracking-techniques-k-curvature-and-convexity-defects.html, [Accessed: June 2016]

[10] CRETU, A.-M., & Plouffe, G., "*Static and Dynamic Hand Gesture Recognition in Depth Data Using Dynamic Time Warping*", IEEE Transactions on Instrumentation and Measurement, Volume: 65, Issue: 2, February 2016.

[11] "Xsens 3D Motion Tracking*",* https://www.xsens.com/tags/imu/, [Accessed: June 2016]

[12] FITZGERALD, A., "*A Practical Guide to MEMS Inertial*", Stanford PNT Symposium, 14 November 2013.

[13] RENAUT, Felix, "*MEMS Inertial Sensors Technology*", Swiss Federal Institute of Technology Zurich, 2013.

[14] "What is MEMS Technology?*",* https://www.memsnet.org/mems/what_is.html, [Accessed: May 2016]

[15] BAO, M. H., "*Handbook of sensors and actuators*", Elsevier B.V. 2000, ISBN 0-444-50558-X.

[16] WILSON, Jon S., "*Sensor Technology Handbook*", Elsevier 2005. ISBN 0-7506-7729-5.

[17] STROUD, Adam; MILLETE, "*Professional Android Sensor Programming*", John Wiley and Sons, 2012. ISBN 978-118-18348-9.

[18] "Accelerometer Basics", https://learn.sparkfun.com/tutorials/accelerometer-basics, [Accessed: February 2016]

[19] "Piezo Electric Accelerometer", https://diyhacking.com/arduino-mpu-6050-imu-sensor-tutorial/, [Accessed: June 2016]

[20] Norton, Harry N., "*Handbook of Transducers"*, 1989 Prentice Hall PTR. ISBN 0-13-382599-X.

[21] *"*A sketch of a classical gyroscope*",* https://www.comsol.com/blogs/modeling-the-dynamics-of-a-gyroscope/, [Accessed: June 2016]

[22] "Introduction to MEMS gyroscopes", http://www.findmems.com/wikimems-learn/introduction-to-mems-gyroscopes, [Accessed: June 2016]

[23] "Indoor Atlas", https://www.indooratlas.com/, [Accessed: July 2016]

[24] "Android Sensor Fusion Tutorial", http://plaw.info/2012/03/android-sensor-fusion-tutorial/, [Accessed: March 2016]

[25] "Android Developers – Motion Sensors", https://developer.android.com/guide/topics/sensors/sensors_motion.html, [Accessed: March 2016]

[26] "DTW Algorithm", http://www.psb.ugent.be/cbd/papers/gentxwarper/DTWalgorithm.htm, [Accessed: May 2016]

[27] YURTMAN, A., & BARSHAN, B., "*Automated evaluation of physical therapy exercises using multi-template dynamic time warping on wearable sensor signals*", Computer Methods and Programs in Biomedicine, Volume 117, Issue 2, November 2014, pp. 189-207.

[28] BLUNSOM, Phil, "*Hidden Markov Models*", August 2014.

[29] "Android Developers - Android Wear", https://developer.android.com/wear/index.html, [Accessed: June 2016]

[30] "Android Wear: Stay connected with interactive watch faces", http://officialandroid.blogspot.nl/2015/08/android-wear-stay-connected-with.html, [Accessed: August 2016]

[31] "Android Wear: Designed to your wrist", https://android.googleblog.com/2016/02/android-wear-designed-for-your-wrist.html, [Accessed: August 2016]

[32] "Android Wear 2.0", http://arstechnica.com/gadgets/2016/05/android-wear-2-0-promises-to-free-your-watch-from-your-phone/, [Accessed: August 2016]

[33] "Android Application Security Part 2 – Understanding Android Operating System", https://manifestsecurity.com/android-application-security-part-2/, [Accessed: June 2016]

[34] "Adding Wearable Features to Notifications", https://developer.android.com/training/wearables/notifications/index.html, [Accessed: June 2016]

[35] "Android Studio vs. Eclipse", https://infinum.co/the-capsized-eight/articles/android-studio-vs-eclipse-1-0, [Accessed: June 2016]

[36] "Ingafitness – The bicep curl", http://ingafitness.com/bicep-curl/, [Accessed: July 2016]

[37] "3D Rotations (flying a plane)", https://coolcodea.wordpress.com/2013/12/28/142-3d-rotations-flying-a-plane/, [Accessed: June 2016]

[38] "Understanding Euler Angles", http://www.chrobotics.com/library/understanding-euler-angles, [Accessed: June 2016]

[39] "Standing Chest Fly", http://www.menshealth.com.sg/taxonomy/term/7669, [Accessed: July 2016]

[40] "Gyroscope Uses", http://gyroscopes.org/uses.asp, [Accessed: June 2016]

[41] "Gyroscope Behaviour", http://www.gyroscopes.org/behaviour.asp, [Accessed: June 2016]

[42] RANGE, Shannon; MULLINS, Jennifer., *"Brief History of Gyroscopes*", 2015

[43] "Gyroscope in mobile and tablets", http://mobilegyros.blogspot.pt/, [Accessed: June 2016]

[44] "Indoor Atlas Study", https://gigaom.com/2013/09/25/indooratlas-uses-geomagnetism-to-map-buildings-gps-cant-reach/, [Accessed: June 2016]

[45] FOLGADO, Duarte, "*Measuring Repetitive Tasks using Inertial Sensors*". Master Thesis. Universidade Nova de Lisboa

[46] SILVA, Paulo, "*Smartphone Gesture Learning*". Master Thesis. Faculdade de Engenharia da Universidade do Porto

[47] MADUREIRA, João, "*Sensors on Mobile Devices for AAL*". Master Thesis. Instituto Superior de Engenharia do Porto

[48] WALTER, Patrick, "*The History of the Accelerometer*.", 2007 Sound & Vibration, vol. 41, nº4, pp. 16-25. ISBN 1541-0161

[49] ELMENREICH, Wilfried, "*An Introduction to Sensor Fusion*", Research Report 47/2001, Vienna University of Technology, Austria, November 2002.

[50] “Android Developers- Sensors Overview”, https://developer.android.com/guide/topics/sensors/sensors_overview.html, [Accessed: March 2016]

[51] TSIPORKOVA, Elena, “*Dynamic Time Warping Algorithm for Gene Expression Time Series*”, Presentation, Gent University, Belgium

[52] MULLER, M., “*Information Retrieval for Music and Motion*”, 2007. ISBN 978-3-540-74047-6

[53] “Three Basic problems of HMMs”, http://jedlik.phy.bme.hu/~gerjanos/HMM/node6.html, [Accessed: March 2016]

[54] “Android – History”, https://www.android.com/intl/pt_pt/history/, [Accessed: June 2016]

[55] “Android Architecture Guides for Beginners”, http://www.edureka.co/blog/beginners-guide-android-architecture/, [Accessed: July 2016]

[56] “Apple Watch vs Android Wear”, http://elekslabs.com/2015/03/apple-watch-vs-android-wear-time-to-drive-tesla-further.html, [Accessed: June 2016]

[57] “Android Studio and SDK Tools, https://developer.android.com/studio/index.html, [Accessed: June 2016]

[58] “Tizen”, https://www.tizen.org/, [Accessed: August 2016]

[59] “Pebble”, https://blog.getpebble.com/, [Accessed: August 2016]

[60] “Firefox OS”, https://www.mozilla.org/en-US/firefox/os/, [Accessed: August 2016]

[61] “Apple Watch – watchOS”, https://www.apple.com/pt/watchos/, [Accessed: August 2016]

[62] BARBIC, Jernej, “Quaternions and Rotations”, CSCI 520 Computer Animation and Simulation, University of Southern California

[63] “6.4.3.1 Single Exponential Smoothing”, http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm, [Accessed: June 2016]

[64] “Moving average and exponential smoothing models”, http://people.duke.edu/~rnau/411avg.htm, [Accessed: June 2016]

[65] “Quora – How to calculate total acceleration from the x,y and z g-force values given by an accelerometer”, https://www.quora.com/How-do-I-calculate-total-acceleration-from-the-x-y-and-z-g-force-values-given-by-an-accelerometer, [Accessed: June 2016]

[66] “Moto 360 – Motorola Support”, https://motorola-global-portal.custhelp.com/app/product_page/faqs/p/2815,9141/, [Accessed: June 2016]