UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Improving the detection of the transport mode in the MobilitApp Android application

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Antonio Cárdenas García

In partial fulfilment

of the requirements for the degree in

GRAU EN CIÈNCIES I TECNOLOGIES DE TELECOMUNICACIÓ

Advisor: Mónica Aguilar Igartua

Co-advisor: Silvia Puglisi

Barcelona, September 2016

Abstract

This study aims to improve an existing application for citizen mobility analytics: MobilitApp. Two modules were added to the app in order to develop the desired functionalities.

The first module has been added is an accident detector. Because of the previous work involves dealing with the accelerometer, it seemed interesting to try to get more out of the accelerometer and include the functionality to detect when the phone has suffered an impact. Once the impact has been detected, an emergency message is sent.

The second module consists of a pedometer, MobilitApp counts the number of the user's steps and presents them with an approximation of the distance walked and the number of calories burned.

As a last step, we have made an analysis of the graphs that have been obtained from the user's accelerometers travelling on train or subway, and we have checked their resemblance to a sinusoidal signal. This feature will help in future works to better identify the type of public transportation system being used by the user. Besides, it would help to save battery since the GPS signal will be less used.

Resum

L'objectiu ha estat el de millorar l'aplicació MobilitApp ja existent, per això s'han afegit dos mòduls.

El primer mòdul que s'ha afegit ha estat un detector d'accidents. A causa que en els anteriors treballs s'estava tractant amb l'acceleròmetre, ens va semblar interessant intentar treure més profit a l'acceleròmetre e incloure la funcionalitat de poder detectar quan el mòbil ha patit un impacte. Un cop l'impacte ha estat detectat s'envia un missatge d'emergència.

El segon mòdul consisteix en un podòmetre, MobilitApp compte els passos de l'usuari i se'ls presenta amb una aproximació de la distància recorreguda y de les calories cremades.

Com a últim pas, em fet un anàlisi sobre les gràfiques que s'han obtingut dels usuaris preses de l'acceleròmetre sobre metro o tren, i hem comprovat la seva semblança amb una senyal sinusoïdal. Aquesta característica ens ajudarà en treballs futurs per identificar millor el tipus de sistema de transport utilitzat per l'usuari. A més, seria de gran ajuda per estalviar bateria ja que el senyal GPS seria menys utilitzat.

Resumen

El objetivo ha sido el de mejorar la aplicación MobilitApp ya existente, para ello se han añadido dos módulos.

El primer módulo que se ha añadido ha sido un detector de accidentes. Debido a que en los anteriores trabajos se estaba tratando con el acelerómetro, nos pareció interesante intentar sacar más provecho al acelerómetro e incluir la funcionalidad de poder detectar cuando el móvil ha sufrido un impacto. Una vez el impacto ha sido detectado se envía un mensaje de emergencia.

El segundo módulo consiste en un podómetro, MobilitApp cuenta los pasos del usuario y se presenta junto a una aproximación de la distancia recorrida y del número de calorías quemadas.

Como último paso, hemos realizado un análisis sobre las gráficas que se han obtenido de los usuarios tomadas del acelerómetro sobre metro o tren, y hemos comprobado su parecido con una señal sinusoidal. Esta característica ayudará en futuros trabajos para identificar mejor el tipo de sistema de transporte utilizado por el usuario. Además, sería de gran ayuda para ahorrar batería ya que la señal GPS sería menos utilizada.

Dedicated to my girlfriend, my friends and my parents for giving me financial help to finish my studies and all for supporting me

Acknowledgements

I would like to start thanking the last student who worked on this project before me (Gerard Marrugat), for having helped me to start with this study. He helped me in my introduction to Android sending me a link to an online tutorial that was very useful, also he sent me an own project to learn how he used the accelerometer which is then incorporated into MobilitApp. In addition, when I handed the Raspberry Pi, he passed me a document written by him about how to start up the server. Moreover, he helped me with a series of tasks to be performed with new programmers such as: incorporation of the designer in the group of developers on Google Play, on your computer to manage Google APIs or configuration of the web as the page where it was registered recently changed.

And especially, to thank my Advisor, Monica Aguilar, who introduced me to the project they were working and that encouraged me to participate in it and with my Co-Advisor, Silvia Puglisi, they have given me support throughout the project concerning doubts, comments, modifications, features, suggesting improvements for implementation and helping to find possible failures in the application.

Table of contents

Abstract 1
Resum 2
Resumen 3
Acknowledgements 5
Table of contents 6
List of Figures 9
List of Tables: 13
Glossary of acronyms 14
1. Introduction 15
2. State of the art 16
2.1. Emergency Call Applications 16
2.1.1. Alpify 16
2.1.1.1. Description and Functionality 16
2.1.1.2. Similarities 16
2.1.1.3. Differences 16
2.1.2. SOSmart notificación de choque 16
2.1.2.1. Description and Functionality 16
2.1.2.2. Similarities 17
2.1.2.3. Differences 17
2.2. Step Counter Applications 17
2.2.1. Podómetro y Entrenador de Peso 17
2.2.1.1. Description and Functionality 17
2.2.1.2. Similarities 17
2.2.1.3. Differences 17
3. Crash Detection Module 18
3.1. Emergency Call 18
3.2. Listening Mobile Sensors 18
3.2.1. Motion Sensors on Android devices and the Accelerometer 18
3.3. Approach of the solution 19
3.3.1. Architecture of the solution 20
3.3.2. Application 21
3.3.2.1. Launching and Stopping the service and settings. 21

3.3.2.2. Graphic User Interfaces 22
3.4. Development 25
3.5. Algorithm 28
3.6. Emergency Messages 29
3.6.1. SMS 29
3.6.2. Message to the server 30
3.7. Code 31
3.7.1. AccelerometerServiceListener 31
3.7.1.1. Troubleshooting 34
3.7.2. EmergencyService 35
3.7.3. SendEmergencyMessage 37
3.7.4. AndroidManifest.xml 39
4. Step Counter Module 41
4.1. Step Counter 41
4.2. Step Counter Sensor 41
4.2.1. Devices using MobilitApp 41
4.3. Approach of the solution 42
4.3.1. Architecture of the solution 43
4.3.2. Application 44
4.3.2.1. Load settings 44
4.3.2.2. Graphic User Interfaces 45
4.4. Development 46
4.4.1. How the sensor Works 46
4.4.2. Step Counter Activity 47
5. Analysing Mobility Data 48
5.1. Subway 48
5.1.1. Sample 1 49
5.1.2. Sample 2 51
5.1.3. Sample 3 54
5.1.4. Sample 4 57
5.1.5. Conclusions 59
5.2. Train 60
5.2.1. Sample 1 61
5.2.2. Sample 2 64

5.2.3. Sample 3 66
5.2.4. Sample 4 68
5.2.5. Conclusions 70
5.3. Conclusions of the Analysis 71
6. Conclusions and future development: 72
6.1. Conclusions 72
6.2. Future Work 72
6.2.1. Crash Detector 72
6.2.2. Step Counter 72
7. Annexes 73
7.1. Annex A - Setting a server up with Raspberry Pi 73
7.2. Annex B – Web Page 74
Bibliography: 75

List of Figures

Figure 1: Axis Orientation 18
Figure 2: Android Manifest 19
Figure 3: Crash Detection Packages 20
Figure 4: MainActivity.java 21
Figure 5: MainActivity.java 22
Figure 6: Main Activity 22
Figure 7: Notify accident off 22
Figure 8: Notify accident off 22
Figure 9: SMS on 23
Figure 10: No phone saved 23
Figure 11: Contact Selected 23
Figure 12: Medical Info 23
Figure 13: ECall 24
Figure 14: Emergency 24
Figure 15: Activities 25
Figure 16: DifferentTypes 25
Figure 17: Entities 26
Figure 18: ILastLocationSaved 26
Figure 19: IEmergencyData 26
Figure 20: IEmergencySensorData 26
Figure 21: HandlersAndServices 27
Figure 22: Emergency SMS 29
Figure 23: Emergency Message EmergencyButton 30
Figure 24: Emergency Message NoEmergency 30
Figure 25: Emergency Message EmergencyAccelerometer 30
Figure 26: AccelerometerServiceListener Class 31
Figure 27: Register and Unregister the AccelerometerServiceListener 31
Figure 28: onCreate AccelerometerServiceListener 32
Figure 29: onDestroy AccelerometerServiceListener 32
Figure 30: onSensorChanged AccelerometerServiceListener 1 33
Figure 31: onSensorChanged AccelerometerServiceListener 2 33
Figure 32: onSensorChanged AccelerometerServiceListener 3 34
Figure 33: onSensorChanged AccelerometerServiceListener 4 34

Figure 34: EmergencyService onCreate 35
Figure 35: EmergencyService methods 1 36
Figure 36: EmergencyService methods 2 36
Figure 37: SendEmergencyMessage Class 1 37
Figure 38: SendEmergencyMessage Class 2 37
Figure 39: SendEmergencyMessage Class 3 38
Figure 40: SendEmergencyMessage Class 4 38
Figure 41: AndroidManifest Activities 39
Figure 42: AndroidManifest Services 39
Figure 43: AndroidManifest Permissions 39
Figure 44: AndroidManifest uses-features 40
Figure 45: Android Versions Use 41
Figure 46: Step Counter Packages 43
Figure 47: MainActivity.java 44
Figure 48: Step Counter on 45
Figure 49: Step Counter off 45
Figure 50: Main Activity 45
Figure 51: Step Counter Use 45
Figure 52: PreferencesTypesSetpCounter 46
Figure 53: StepCounterActivity 47
Figure 54: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 1 Part 1 49
Figure 55: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 1 Part 2 49
Figure 56: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 1 Part 3 49
Figure 57: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 1 Part 1 50
Figure 58: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 1 Part 2 50
Figure 59: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 1 Part 3 50
Figure 60: Acceleration Axis Z Sample 1 Part 1 50
Figure 61: Acceleration Axis Z Sample 1 Part 2 50
Figure 62: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 2 Part 1 51
Figure 63: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 2 Part 2 51
Figure 64: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 2 Part 3 51
Figure 65: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 2 Part 1 52
Figure 66: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 2 Part 2 52
Figure 67: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 2 Part 3 52

Figure 68: Acceleration Axis Z Sample 2 Part 1 53
Figure 69: Acceleration Axis Z Sample 2 Part 2 53
Figure 70: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 3 Part 1 54
Figure 71: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 3 Part 2 54
Figure 72: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 3 Part 3 54
Figure 73: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 3 Part 4 54
Figure 74: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 3 Part 1 55
Figure 75: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 3 Part 2 55
Figure 76: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 3 Part 3 55
Figure 77: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 3 Part 4 55
Figure 78: Acceleration Axis Z Sample 3 Part 1 56
Figure 79: Acceleration Axis Z Sample 3 Part 2 56
Figure 80: Acceleration Axis X vs Sinusoid 0.063 Hz Sample 4 Part 1 57
Figure 81: Acceleration Axis X vs Sinusoid 0.063 Hz Sample 4 Part 2 57
Figure 82: Acceleration Axis X vs Sinusoid 0.063 Hz Sample 4 Part 3 57
Figure 83: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 4 Part 1 58
Figure 84: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 4 Part 2 58
Figure 85: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 4 Part 3 58
Figure 86: Acceleration Axis Z Sample 4 Part 1 58
Figure 87: Acceleration Axis Z Sample 4 Part 2 58
Figure 88: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 1 Part 1 61
Figure 89: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 1 Part 2 61
Figure 90: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 1 Part 3 61
Figure 91: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 1 Part 1 62
Figure 92: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 1 Part 2 62
Figure 93: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 1 Part 3 62
Figure 94: Acceleration Axis Z Sample 1 Part 1 63
Figure 95: Acceleration Axis Z Sample 1 Part 2 63
Figure 96: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 2 Part 1 64
Figure 97: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 2 Part 2 64
Figure 98: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 2 Part 3 64
Figure 99: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 2 Part 1 64
Figure 100: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 2 Part 2 64
Figure 101: Acceleration Axis Z Sample 2 Part 1 65

Figure 102: Acceleration Axis Z Sample 2 Part 2 65
Figure 103: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 3 Part 1 66
Figure 104: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 3 Part 2 66
Figure 105: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 3 Part 3 66
Figure 106: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 3 Part 1 66
Figure 107: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 3 Part 2 67
Figure 108: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 3 Part 3 67
Figure 109: Acceleration Axis Z Sample 3 Part 1 67
Figure 110: Acceleration Axis Z Sample 3 Part 2 67
Figure 111: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 4 Part 1 68
Figure 112: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 4 Part 2 68
Figure 113: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 4 Part 3 68
Figure 114: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 4 Part 1 69
Figure 115: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 4 Part 2 69
Figure 116: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 4 Part 3 69
Figure 117: Acceleration Axis Z Sample 4 Part 1 69
Figure 118: Acceleration Axis Z Sample 4 Part 2 69
Figure 119: Notify accident Web Page 74
Figure 120: Step Counter Web Page 74

List of Tables:

Table 1: Average Power Consumption 18
Table 2: Degree precision versus length [9].....29
Table 3: Transport Samples vs Sinusoid.....71

Glossary of acronyms

API: Application Programming Interface

APK: Android Application Package

APP: Application

BMI: Body Mass Index

CO2: Carbon Dioxide

CSV: Comma-Separated Values

DNS: Domain Name Server

DRY: Don't Repeat Yourself

ECall: Emergency Call

GPS: Global Positioning System

GUI: Graphical User Interface

HTTP: HyperText Transfer Protocol

SDK: Software Development Kit

SOLID: Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion

SSH: Secure Shell

XML: eXtensible Markup Language

1. Introduction

In this project, we are going to continue with the development of MobilitApp [1]. MobilitApp is an Android application to obtain data regarding the mobility of citizens in Barcelona.

We want to give a practical utility to the user because the application collects data for us but does little for the user.

For this reason it was decided to incorporate practical features for the application to earn more interest by the community.

The main objectives of this Degree Thesis are:

- Develop a module to detect accidents.
- Develop a module to count steps.
- Study the data collected from the accelerometer that comes from samples taken at subway and train.

Just to meet the first two objectives, the two modules have been developed independently of the project. That is, these modules could be easily incorporated into another project, only it wold have to adapt the GUI and launch the services [6].

This project is a continuation of others, MobilitApp has grown gradually. In my case it has been necessary first phase of learning because I had never worked with Android. In my learning phase I was following a tutorial in which small basic applications were made. This way I learned to work with Android gradually. Moreover there is a part of learning about how the MobiliApp project works as there are many classes and is important to be familiar with it.

For the development of the application, it has been used the Android Studio IDE [11]. With this tool the complete application has been generated using Java [12] classes and graphical interfaces using XML files.

Also, the application is multilingual. Changes depending on the language of the phone, being the three languages used: Spanish, Catalan and English. All texts are stored in a Strings file for each language.

To study the data has been used Excel [13] to make a basic analysis, and only the graphics of metro and train have been analysed because these are the ones that we thought that they would be more periodic.

This document is structured with seven chapters plus references and two annexes.

The Chapter 2 introduces the context where is located our research.

The next ones, Chapters 3 and 4 describes the design and implementation process of our two new modules, giving details on how it runs and how it interacts with the user.

Later, on Chapter 5 we report the results of the accelerometer data analysis coming from samples collected by metro and train.

After that, Annex A explains how the server has been set up and Annex B shows the extra features added to the project like the web page and the promotional video.

2. State of the art

There are some applications on Google Play that perform functions such as those developed for MobilitApp.

2.1. Emergency Call Applications

There are some applications intended to accidents detector and ask for help. Each of them are focused on a series of activities or specific cases.

2.1.1. Alpify

This application is primarily intended for mountain climbers, Alpify connects with 112 services when the user requests it.

2.1.1.1. Description and Functionality

The application sends an emergency signal to the 112 services when the user presses the red button. By doing this, Alpify also sends the GPS position.

In addition, the route that has been made is saved so it can be helpful for the emergency services.

In case you have no internet, the app sends an SMS to the Alpify system and they will contact with 112.

2.1.1.2. Similarities

MobilitApp also allows the user to send an emergency signal and sends an SMS. In both cases the two applications send the GPS position.

2.1.1.3. Differences

Unlike MobilitApp, Alpify need a complex system on the server, this is responsible for managing all the emergencies. They need a service staff to treat all generated emergencies and guiding the emergency personnel to the specific location.

Moreover, Alpify does not detect accidents automatically to send an emergency signal without having to wait to the user.

Finally, MobilitApp does not send the route that has been followed unlike Alpify.

2.1.2. SOSmart notificación de choque

It is an application designed to detect car accidents and to face emergencies.

2.1.2.1. Description and Functionality

SOSmart automatically detects when the user is involved in a car accident, and sends the location of the accident to some emergency contacts previously selected.

It has a panic button to send an emergency signal to your contacts.

It offers a list of hospitals near your location and displays a route to get there.

When you open the app it shows the approximate address in which are you located.

2.1.2.2. Similarities

Both of them detect automaticly when the user is involved in an accident and sends the location in a SMS. Also, they have a panic button.

2.1.2.3. Differences

MobilitApp sends the SMS to a unique contact, SOSmart can send the SMS to more than one. Another difference is that the panic button in MobilitApp is only available when the app detects an accident.

Moreover, SOSmart has the option of offer a list of hospitals and it can show the approximate address of the user.

2.2. Step Counter Applications

We found that we have a variety of applications to count the number of steps. All of them are very similar among them.

2.2.1. Podómetro y Entrenador de Peso

This application is intended to carry a healthy lifestyle.

2.2.1.1. Description and Functionality

This application counts steps and also helps us to control weight, incorporates achievements, offer graphics, averages, body mass index and allows us to share data with friends.

2.2.1.2. Similarities

The similarities are simply control steps, distance and calories calculation. Moreover, both allow us to see the number of total steps and the steps have been made in the present day,

2.2.1.3. Differences

This application incorporates several extra features that MobilitApp does not have. Among them are the creation of graphics, shows the number of steps by days and control the body mass index with charts.

3. Crash Detection Module

3.1. Emergency Call

Our goal is to create a module in our project that can identify a possible emergency and make an emergency call (eCall).

3.2. Listening Mobile Sensors

The previous version of the application read data from smartphone's sensors (accelerometer and gyroscope [2]) and we thought interesting to use this to develop an emergency call when a possible accident is detected.

3.2.1. Motion Sensors on Android devices and the Accelerometer

Smartphones contain many sensors and allow us to obtain much information about the device.

In general terms, we can differentiate into two types:

- Hardware-Based: physically incorporated in the device.
- Software-Based: where the data comes from other sensors.

We have focused on the accelerometer, which is a hardware-based sensor that measures the acceleration of the device in the three coordinates (x, y and z).

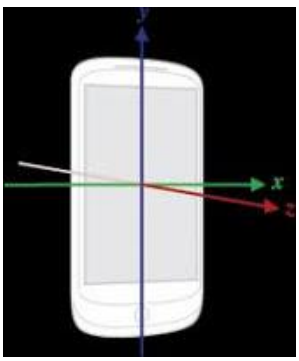

Figure 1: Axis Orientation

Our main aim is to detect possible traffic accidents, falls or similar. To do this, we will read accelerometer data in search of some sudden acceleration which would indicate an impact on the phone.

Moreover, we want to know the increase in consumption that we have due to use the accelerometer. In the following table we can see the average consumption values for different sensors / features:

Sensor/Feature	Average Power Consumption
Accelerometer(Hardware)	0.23 mA
Magnetic Field(Hardware)	6.8 mA
Orientation(Software)	13.13 mA
Gyroscope(Hardware)	6.1 mA
WiFi	330 mA
3G	210 mA

Table 1: Average Power Consumption

As we can see, the accelerometer has a very low power consumption compared to other sensors. Also, we could have combined the use of the accelerometer with gyroscope, but considering that our application keeps activated the sensors all the time, we have chosen to use only the accelerometer.

As a result, battery consumption will be much lower and we fulfills perfectly with our objective.

Looking ahead to the application, we need to indicate which sensors are used. This is configured in the project in the Android Manifest [3].

```
<uses-features
    android:name="android.hardware.sensor.accelerometer"
    android:required="true" />
<uses-features
    android:name="android.hardware.sensor.gyroscope"
    android:required="true" />
```

Figure 2: Android Manifest

3.3. Approach of the solution

From the outset, the solution has been raised for it to be modular and scalable.

Corresponding to the eCall module, it has been designed with the aim that can be incorporated into another project and make it work without having to touch the logical layer or the business logic, we only should adapt the UI.

This module is in a separate package and well-structured in small packages.

This design is given due to the interest of wanting to follow the SOLID principles [4]. As a project progresses over time and each part is made by a different developer, it has thought proper to follow the SOLID principles in order to create a clean code, easy to understand, easy to maintain and easy to upgrade and incorporate new functionalities.

3.3.1. Architecture of the solution

In the next image we present the architecture of our CrashDetection module:

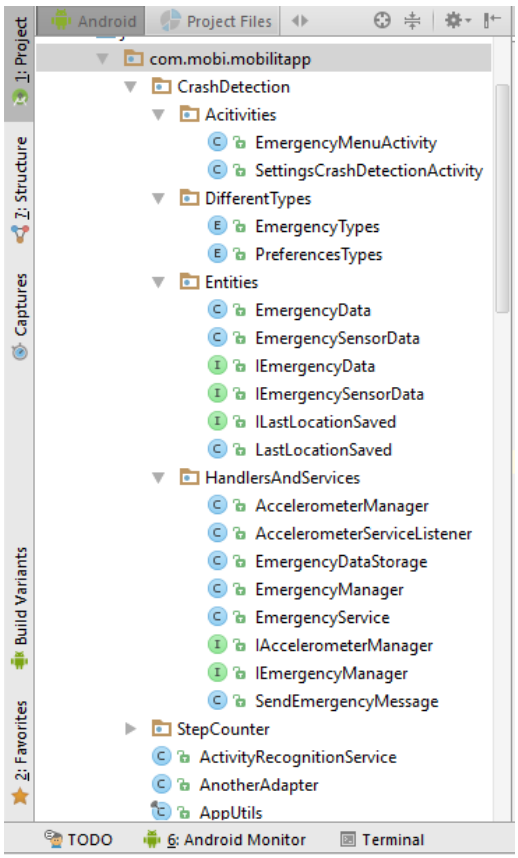

Figure 3: Crash Detection Packages

We have separated the module in these four packages:

- Activities: This package contains the user interface, this part is the one that cannot be used in other projects as it is designed specifically to work with MobilitApp. For this reason, this should be the lightest layer and must not contain any business logic.
- DifferentTypes: The MobilitApp application uses SharedPreferences [5] so I continued with this practice when I needed to store some user data. These SharedPreferences use strings as a key, I think that is good practice to put these strings in an enum class, so it will help to future programmers when they need to access or modify these user preferences.
- Entities: This package contains the entities with which we work, a series of Java objects that have been created to save the data. Specifically we use 3 and have been used interfaces following the SOLID principles (Liskov substitution principle) with which we have a dependency on a contract and not to a particular object, this implies that in the future we could create a new object that incorporates new features and easily replace the old one simply fulfilling a contract established.
- HandlersAndServices: And finally, this package contains the handlers that will help us manage different activities and services [6] we have and run in the background.

3.3.2. Application

The crash detection module has been developed in a separate project, we have verified that had the expected behavior and then copied and pasted the module to the final project. After this, we only had to add a few lines of code to launch or stop the service in the app and customization settings.

3.3.2.1. Launching and Stopping the service and settings.

The main class is: MainActivity.java is in this class where personal user settings are loaded and the service will be launched if it is necessary.

The user settings are saved by SharedPreferences:

```
prefs = getSharedPreferences("prefs", Context.MODE_PRIVATE);
```

These preferences were designed to save user settings so it has seemed good idea to take advantage of these resources.

```
if (prefs.getString("switchCrashOnOff", "no_selection").equalsIgnoreCase("no_selection")) {
    prefs.edit().putString("switchCrashOnOff", "off").commit();
}

if (prefs.getString("sensitivityCrashMaximumThreshold", "no_selection").equalsIgnoreCase("no_selection")) {
    prefs.edit().putString("sensitivityCrashMaximumThreshold", "25").commit();
}

if(prefs.getString("timeAtLastLocation", "no_time").equalsIgnoreCase("no_time")){
    prefs.edit().putString("timeAtLastLocation","0").commit();
}

if(prefs.getString("latitudeAtLastLocation", "no_latLoc").equalsIgnoreCase("no_latLoc")){
    prefs.edit().putString("latitudeAtLastLocation","0").commit();
}

if(prefs.getString("longitudeAtLastLocation", "no_lonLoc").equalsIgnoreCase("no_lonLoc")){
    prefs.edit().putString("longitudeAtLastLocation","0").commit();
}

if(prefs.getString("switchCrashOnOff", "no_selection").equalsIgnoreCase("on")){
    startService(new Intent(this, AccelerometerServiceListener.class));
}
```

Figure 4: MainActivity.java

This code will must be called every time MobilitApp is open, so we have to locate it in his method:

```
protected void onCreate(Bundle savedInstanceState)
```

These 6 conditional could be encapsulated in a method, but I wanted to respect the programming style of the previous programmers, also are short lines easy to read. Thus the next programmers who want to modify or add new preferences only have to go to onCreate [7] method and look for these lines.

It is also important to stop the service whenever you exit the application. Therefore we have to stop the service in its onDestroy [7] method:

```
@Override
protected void onDestroy() {
    stopService(new Intent(this, AccelerometerServiceListener.class));
    super.onDestroy();
    Log.v("aki", "aki_onDestroyMainAct");
}
```

Figure 5: MainActivity.java

These are the modifications to be made to incorporate the new feature. As we discussed it is very simple to move the crash detection block to other projects. Finally, we would add graphical interfaces for the user has a simple and intuitive way to activate, deactivate or adjust the service.

3.3.2.2. Graphic User Interfaces

Two **GUI**s have been created so that the user can modify the parameters of the sensor, activate or stop so easy and intuitive.

These user interfaces has been designed with a very simple look for 3 reasons:

- Intuitive interface, easy to use.
- Quick, reduces the time required to make any action or adjustment.
- Clear and concise, unambiguous.

For this reason it would not be necessary to explain what they do, they should have all the information you need to know.

Screen's diagram:

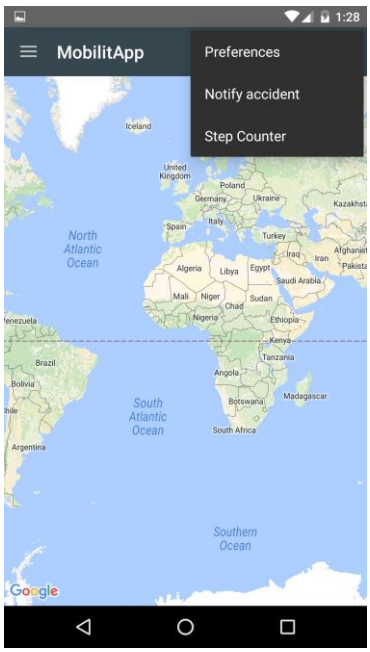

Figure 6: Main Activity

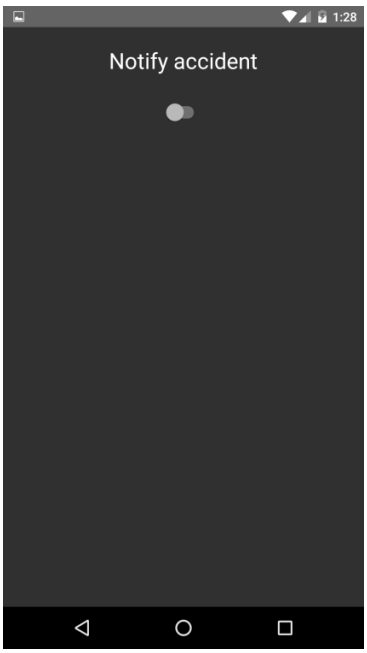

Figure 8: Notify accident off

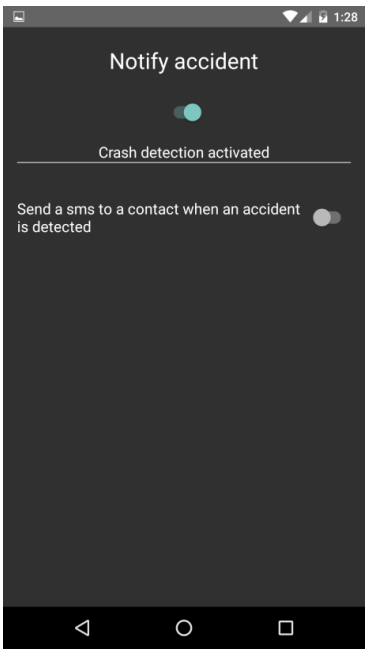

Figure 7: Notify accident off

In previous versions, we had 3 radio buttons to adjust the sensitivity level but it was too complex for the user, so we decided to remove them and establish a unique level of sensitivity for everything.

This is how the crash detection functionality can be activated, fast and easy. And now, the app is asking us if we want to send a SMS to a contact when the app detects an accident.

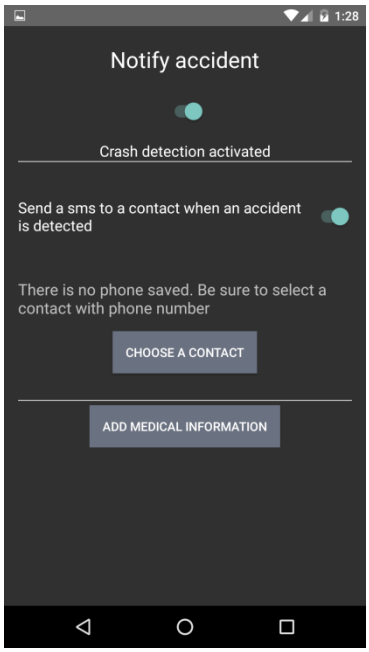

Figure 9: SMS on

When we select de option of sending a SMS to a contact, we see this image (Figure 9: SMS on).

At this point we have to select a contact from our contact list after clicking on the "Choose a contact" button.

If we try to go back without selecting a contact, we will see the next image (Figure 10: No Phone Saved). This is an alert that is saying us that there is no phone saved. If we click "Yes" we will go back to the previous screen and the SMS option will be disabled. On the other side, if we want to pick a contact, we can say "No" and the app will show us the contact list to select one contact. In this last case, the phone will be saved and we will see something like the image (Figure 11: Contact selected).

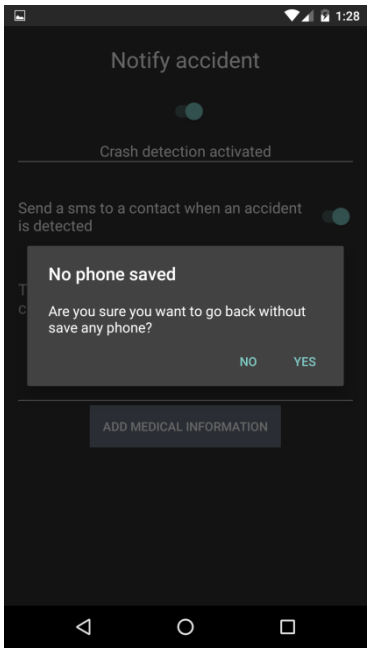

Figure 10: No phone saved

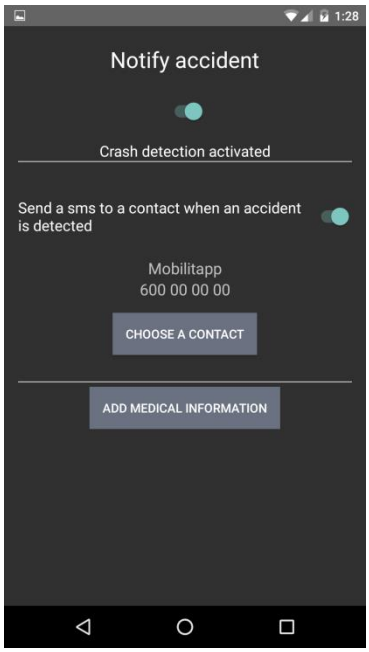

Figure 11: Contact Selected

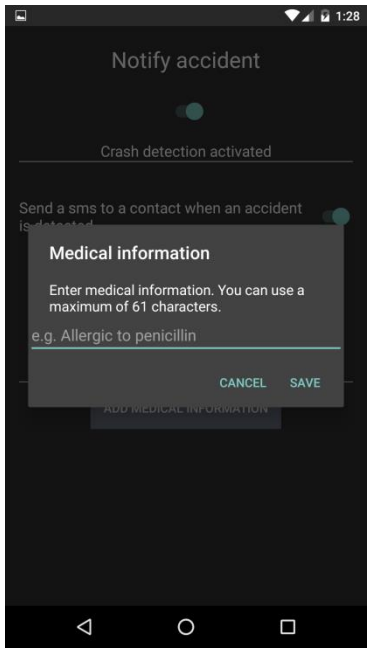

Figure 12: Medical Info

Finally, MobilitApp allows us to introduce medical information to be included at the end of SMS (we will have a maximum number of characters because a SMS has a maximum of 150 characters).

Following this, when the crash detection is activated the app is listening for any impact on the device. When MobilitApp detects an accident, we will listen a short beep sound and the figure (Figure 14: Emergency) will be showed and if we click on the “Emergency” button we will see the message of the figure (Figure 13: Ecall).

Figure 14: Emergency

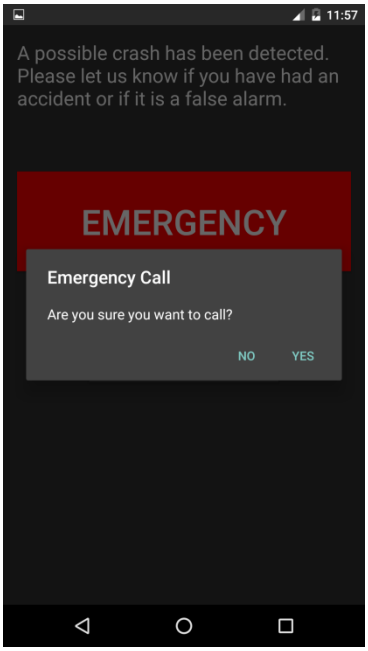

Figure 13: ECall

If we say “No” to the Emergency Call or we click on the “False Alarm” button we go back to the previous screen and everything will continue as before. But if we say “Yes” to the Emergency Call, all the emergency actions will be launched. These actions depend on the user settings and how the app is programmed.

If an accident is detected and we do not touch the mobile, MobilitApp follows an algorithm to guess if the impact received is a false alarm or a real emergency.

3.4. Development

Here is the brain of the Crash Detector Module, all the Java classes that controls the crash detection. We have seen the architecture of the module in 3.3.1, and in this section we will see what the packages do:

- **Activities**:
  This package is composed only by two classes. These two classes are the two Activities that we have analyzed on the previous section (3.3.2.2), one of them is for the Emergency Menu and the other for the Crash Detection Settings.

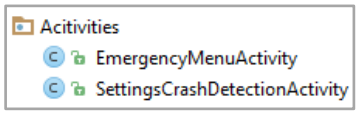

Figure 15: Activities

  They show the screens to the user, listen where the user clicks and manage with two services (the AccelerometerServiceListener and the EmergencyService).

  This is the top layer therefore must be as light as possible. Here we do not have business logic.

- **DifferentTypes**:
  This is a very simple package with two enum classes, where we have the “keys” of the SharedPreferences and the types of Emergency typified as we have commented on the 3.3.1 section.

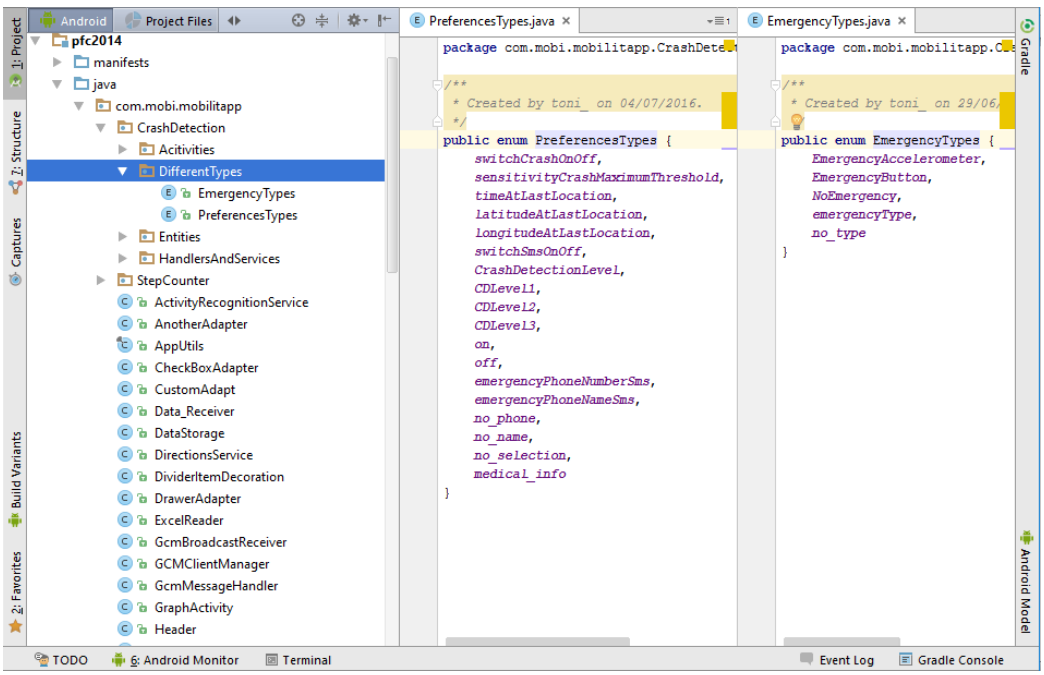

Figure 16: DifferentTypes

- **Entities**:
  Here we find 3 interfaces and 3 classes that implement each of the interfaces. These classes are used to manage data on the app. As mentioned, the interfaces are created following the principles SOLID with the aim of establishing a permanent contract with the interface but not force to follow a specific implementation.

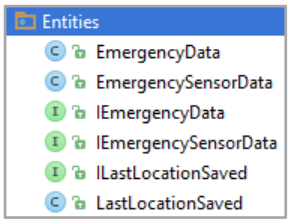

Figure 17: Entities

```
public interface ILastLocationSaved {
    String getLatitudeAtLastLocation();

    void setLatitudeAtLastLocation(String latitudeAtLastLocation);

    String getLongitudeAtLastLocation();

    void setLongitudeAtLastLocation(String longitudeAtLastLocation);

    String getTimeAtLastLocation();

    void setTimeAtLastLocation(String timeAtLastLocation);

    void refreshData(String time, String latitude, String longitude);
}
```

Figure 18: ILastLocationSaved

```
public interface IEmergencyData {
    double getLongitude();

    void setLongitude(double longitude);

    double getLatitude();

    void setLatitude(double latitude);

    String getTime();

    void setTime(String time);

    String getType();

    void setType(String type);

    void setData(String time, double latitude, double longitude, String type);
}
```

Figure 19: IEmergencyData

```
public interface IEmergencySensorData {

    String getTime();

    void setTime(String time);

    double getX();

    void setX(double x);

    double getY();

    void setY(double y);

    double getZ();

    void setZ(double z);

    void setValues(String time, double x, double y, double z);

}
```

Figure 20: IEmergencySensorData

- **HandlersAndServices**:
  In this package is where the services, handlers and managers are located.

  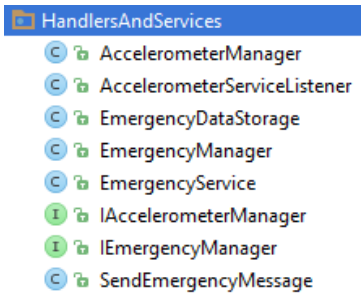

  Figure 21: HandlersAndServices

  There are two interfaces:
  - **IAccelerometerManager**: It has to manage with the states of the AccelerometerServiceListener.
  - **IEmergencyManager**: It has to manage with the states of the emergencies.

  There are two services:
  - **AccelerometerServiceListener**: This service is listening to the accelerometer sensor and launches the actions of emergency when an accident is detected.
  - **EmergencyService**: Its function is to deal with performing the necessary actions in the background. For example it takes care of sending a message to the server and send an SMS if the user has activated this function.

  And there are two more handlers:
  - **SendEmergencyMessage**: Its responsibility is to collect the data and launch the thread to send the message to the server and send the SMS
  - **EmergencyDataStorage**: It is responsible for creating the csv file and sends it to the server. This file contains the time, position (latitude and longitude) and the type of emergency,

3.5. Algorithm

An algorithm has been designed to detect accidents and false alarms when the user does not answer.

When the device detects a sudden change in the accelerometer it emits a short and powerful sound that warns the user that the impact have been detected and displays the EmergencyActivity. At this time, the timer is reset.

The app read accelerometer data using a threshold to determine if the user is still moving or standing.

Once passed 60 seconds, if the accelerometer's readings indicate that the user is moving, MobilitApp considers that the user is right or that it can notify emergency if necessary, in such a case, after unlocking the phone appear the screen of emergency to send a emergency message quickly.

If not move, after 65 seconds, the phone starts ringing to ask the user to confirm whether it is an emergency or false alarm. In any case, if the user moves, the normal state is restored.

By the timer to 90 seconds and if the device has not moved, the emergency call is automatically launched.

To this day, the application sends a message to the server and send an SMS (if the user has configured the SMS option in the settings).

3.6. Emergency Messages

We have two types of messages, the messages that are sent by SMS to a contact or the messages that are sent to the server as a CSV file.

However, for any type of message, the GPS coordinates are sent using 7 digits. This provides the accuracy needed to locate someone in an emergency.

Decimal Places	Decimal Degrees	Qualitative scale that can be identified
0	1.0	Country or large region
1	0.1	Large city or district
2	0.01	Town or village
3	0.001	Neighborhood, street
4	0.0001	Individual street, land parcel
5	0.00001	Individual trees
6	0.000001	Individual humans
7	0.0000001	Practical limit of commercial surveying
8	0.00000001	Specialized surveying

Table 2: Degree precision versus length [9]

3.6.1. SMS

This is an example of a possible SMS sent when a user has had an accident.

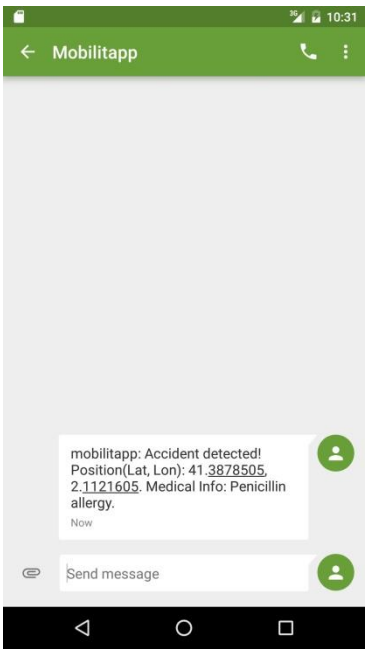

Figure 22: Emergency SMS

We can see that the message is headed by the name of the application and then reports an accident. After that indicates the position with GPS coordinates and medical information entered by the user, such as might be allergic to penicillin.

The message is designed to be as short as possible, so the user has the maximum possible number of characters to enter medical information. The SMS have a maximum of 150 characters.

Additionally, all SMS that is has included the sender and the time at which it is sent.

Moreover, the presentation of the coordinates must be in a correct format without words in between, so that the receiver of the SMS can copy the position and paste it into any browser to easily find the place.

3.6.2. Message to the server

The CSV file sent to the server, in addition to the position, it incorporates the current time and the type of the emergency, this type defines the origin of the signal.

Emergency Types:

- EmergencyButton: This type of emergency indicates that the user has pressed the "emergency" button.

```
"time","latitude","longitude","type"
"08:14:23","41.3878508","2.1121603","EmergencyButton"
```

Figure 23: Emergency Message EmergencyButton

- NoEmergency: This type of emergency indicates that the user has pressed the "false alarm" button.

```
"time","latitude","longitude","type"
"08:15:12","41.3878507","2.1121601","NoEmergency"
```

Figure 24: Emergency Message NoEmergency

- EmergencyAccelerometer: This type of emergency indicates that the message was automatically sent by the Crash Detector without interacting with the user.

```
"time","latitude","longitude","type"
"08:17:48","41.3878502","2.1121591","EmergencyAccelerometer"
```

Figure 25: Emergency Message EmergencyAccelerometer

3.7. Code

3.7.1. AccelerometerServiceListener

The main class of the CrashDetector Module is the AccelerometerServiceListener.

Let's see the principal parts of this class:

```
public class AccelerometerServiceListener extends Service implements SensorEventListener {
    private SensorManager mSensorManager;
    private Sensor mAccelerometer;
    private IAccelerometerManager IAccelerometerManager;
    private EmergencyManager emergencyManager;

    private long start_time, currentTime;

    private long time30Seconds;
    private long time60Seconds;
    private long time65Seconds;
    private long time90Seconds;

    String timestamp;
    double x, y, z;

    private static final String MyTAG = "MyTAG";
    private IEmergencySensorData lastSensorData;
    private final float THERESLHOLD = (float) 1;
    private float maximumAccelerationThreshold;
    private PowerManager.WakeLock mWakeLock;
    SharedPreferences prefs;
    ILastLocationSaved lastLocation;
    Context context;
```

Figure 26: AccelerometerServiceListener Class

For the correct operation we create methods to register and unregister it.

```
    /*
     * Register this as a sensor event listener.
     */
    private void registerListener() {
        // Instance the accelerometer and gyroscope
        mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        // registerListener(EventListener, Sensor, rate)
        mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
    }

    /*
     * Un-register this as a sensor event listener.
     */
    private void unregisterListener() {
        mSensorManager.unregisterListener(this);
    }
```

Figure 27: Register and Unregister the AccelerometerServiceListener

In the method onCreate we initialize the service.

We have to initialize variables and register the Listener.

```
@Override
public void onCreate() {
    super.onCreate();

    time30Seconds = 30000;
    time60Seconds = 60000;
    time65Seconds = 65000;
    time90Seconds = 90000;
    mWakeLock = null;

    // Instance the SENSOR_SERVICE
    mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

    registerListener();

    IAccelerometerManager = new AccelerometerManager();
    emergencyManager = new EmergencyManager();
    lastSensorData = null;

    PowerManager manager = (PowerManager) getSystemService(Context.POWER_SERVICE);
    mWakeLock = manager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, MyTAG);

    prefs = getSharedPreferences("prefs", Context.MODE_PRIVATE);

    //Deprecated. To set different senseibility levels
    //String valueMaximumAccTh = prefs.getString(PreferencesTypes.sensitivityCrashMaximumThreshold.toString(), "16");
    //maximumAccelerationThreshold = Float.parseFloat(valueMaximumAccTh);
    maximumAccelerationThreshold = 18;
    lastLocation = new LastLocationSaved();
}
```

Figure 28: onCreate AccelerometerServiceListener

And we do the opposite with the onDestroy method.

When the onDestroy method is called, the Crash Detection Method should be stopped.

```
@Override
public void onDestroy() {
    super.onDestroy();

    unregisterListener();

    mWakeLock.release();
    stopForeground(true);

    //Stops timerRunnable
    timerHandler.removeCallbacks(timerRunnable);

    Log.v(MyTAG, "AccelerationData_OK");
}
```

Figure 29: onDestroy AccelerometerServiceListener

Each time the accelerometer reads a new value, the onSensorChanged method is executed:

First thing we do is check if the new value of the accelerometer collected exceeds a threshold that we set to see if the device is moving.

```
@Override
public void onSensorChanged(SensorEvent event) {

    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {

        double deltaX = 0, deltaY = 0, deltaZ = 0;

        deltaX = x - event.values[0];
        deltaY = y - event.values[1];
        deltaZ = z - event.values[2];

        //alpha is the LPF constant, alpha = t/(t + dT); t = LPF time constant, dT = time interval between input data
        final float alpha = (float) 0.8;

        if (deltaX > THRESLHOLD || deltaY > THRESLHOLD || deltaZ > THRESLHOLD) {
            //Define a thereshold that means that the device is on moving

            //We collect data from acceleration peaks but we haven´t marked the end of each peak yet
            timestamp = time(currentTime, "mm:ss:SSS");

            if(lastSensorData==null){
                x = event.values[0];
                y = event.values[1];
                z = event.values[2];

                lastSensorData = new EmergencySensorData();
                lastSensorData.setValues(timestamp,x,y,z);
```

Figure 30: onSensorChanged AccelerometerServiceListener 1

When we found that the lastSensorData variable is different of null, this indicates that already had previous samples. In that case we proceed to see if the accelerometer reading exceeds the maximum threshold which is considered possible accident (maximumAccelerationThreshold). If the threshold is exceeded, the state of emergency is activated.

```
            } else {

                if (deltaX > maximumAccelerationThreshold || deltaY > maximumAccelerationThreshold
                        || deltaZ > maximumAccelerationThreshold) {
                    IAccelerometerManager.activateEmergency(currentTime);
                    emergencyManager.askAboutEmergency(this);
                    emergencyManager.beepSound();
                }
                //This Try-Catch prevents from an unusual error when the accelerometer get a null value.
                try{
                    //We use a low pass filter to remove short-terme fluctuations(Jitter) on our samples
                    //x = alpha * event.values[0] + (1 - alpha) * AccelerationData.get(position - 1).getX();
                    x = alpha * event.values[0] + (1 - alpha) * lastSensorData.getX();
                    y = alpha * event.values[1] + (1 - alpha) * lastSensorData.getY();
                    z = alpha * event.values[2] + (1 - alpha) * lastSensorData.getZ();
                }catch (Exception e){
                    x = event.values[0];
                    y = event.values[1];
                    z = event.values[2];
                }

                if (IAccelerometerManager.isAlertState() == true) {
                    if (IAccelerometerManager.elapsedTimeSinceCrash(time60Seconds,currentTime)) {
                        IAccelerometerManager.reStart();
                        emergencyManager.reStart();
                        //Everything is okey, there is no emergency.
```

Figure 31: onSensorChanged AccelerometerServiceListener 2

According to the algorithm discussed, at this point it is where the application will be checking to see if the device is moving or is stopped and depending on how many seconds have passed since the accident perform some action or other.

```
                } else if (IAccelerometerManager.elapsedTimeSinceCrash(time30Seconds,currentTime)
                        && IAccelerometerManager.isEmergencyState()) {
                    IAccelerometerManager.setEmergencyState(false);
                    //The device is moving after a crash, we keep on alert state.
                }
            }
        }
        lastSensorData.setValues(timestamp, x, y, z);
    }
```

Figure 32: onSensorChanged AccelerometerServiceListener 3

```
    else //The device doesn't move
    {
        if (IAccelerometerManager.isAlertState() == true) {

            if (IAccelerometerManager.elapsedTimeSinceCrash(time65Seconds,currentTime)
                    && emergencyManager.isCallingEmergencies()==false) {
                emergencyManager.beepSound();
            }

            if(IAccelerometerManager.elapsedTimeSinceCrash(time90Seconds, currentTime))
            {//Crash detected
                    if(!emergencyManager.isCallingEmergencies()){
                        emergencyManager.setCallingEmergencies(true);
                        context = this;

                        //We send the message to the server with a Service to separate it from the principal goal: eCall()
                        prefs.edit().putString(EmergencyTypes.emergencyType.toString(),
                                EmergencyTypes.EmergencyAccelerometer.toString()).commit();
                        startService(new Intent(this, EmergencyService.class));

                        prefs.edit().putString(PreferencesTypes.switchCrashOnOff.toString(),
                                PreferencesTypes.off.toString()).commit();
                        this.stopSelf();
                    }
            }
        }
    }
}
```

Figure 33: onSensorChanged AccelerometerServiceListener 4

3.7.1.1. Troubleshooting

When the app detects an accident have to perform certain actions such as sending emergency signals.

Initially, these actions were carried out in a Thread [8] that allowed emergency tasks are preformed asynchronously.

When that part was developed, a series of tests were performed an the result was that the application performance became slower.

To solve this problem they were changed for services [6].

Services running in background but run in the same main thread, therefore is not necessary to generate new threads. In this way the app improves performance.

3.7.2. EmergencyService

This is the service that uses MobiliApp to run all the tasks that must be carried out when an accident is detected.

The aim of this service is to run the necessary actions in case of emergency but it does not have to implement any of these actions. It has a number of methods, that they have to be short and clear pieces of code, in this way any programmer fully understand the method onCreate and it is easy to understand the methods.

Any emergency action that has to be incorporated should go on this service but the implementation should be in another class destined to that single action. Thereby the SOLID [4] principles are fulfilled.

```
public class EmergencyService extends Service {

    private static final String MyTAG = "MyTAG";

    ILastLocationSaved lastLocation;
    SharedPreferences prefs;
    String typeEmergency;
    String switchSmsEmergency;
    String on = PreferencesTypes.on.toString();

    @Override
    public void onCreate() {
        super.onCreate();
        lastLocation = new LastLocationSaved();

        loadSharedPreferences();
        refreshLastLocationData();
        sendEmergencySMS();

        if(hasActiveInternetConnection() ==true){

            sendEmergencyMessageToTheServer();
            Toast.makeText(this,"Message sent to the server", Toast.LENGTH_SHORT).show();
        }
        else{
            Toast.makeText(this,"Connection failure", Toast.LENGTH_SHORT).show();
        }

        stopSelf();
    }
```

Figure 34: EmergencyService onCreate

All methods are responsible for dealing with small features, such as:

- Load the SharedPreferences [5] and refresh the strings that we use to see if the users selected the option of send the SMS.
- Refresh the las location data, and store it on an object which implements the ILastLocationSaved interface.
- Send the emergency message to the server. This method calls to the static method sendMessageToTheServer from the SendEmergencyMessage Class.
- Send the SMS. This method calls to the static method sendSMS from the SendEmergencyMessage Class.
- Check the internet connection, because if the device does not have internet connection it will not be able to send the message to the server.

```
private void loadSharedPreferences() {
    prefs = getSharedPreferences("prefs", Context.MODE_PRIVATE);
    typeEmergency = prefs.getString(EmergencyTypes.emergencyType.toString(), EmergencyTypes.no_type.toString());
    refreshUserPreferences();
}

private void refreshUserPreferences() {
    switchSmsEmergency = prefs.getString(PreferencesTypes.switchSmsOnOff.toString(),PreferencesTypes.no_selection.toString());
}

private void refreshLastLocationData() {
    lastLocation.refreshData(
            prefs.getString(PreferencesTypes.timeAtLastLocation.toString(), "0"),
            prefs.getString(PreferencesTypes.latitudeAtLastLocation.toString(), "0"),
            prefs.getString(PreferencesTypes.longitudeAtLastLocation.toString(), "0")
    );
}

private void sendEmergencyMessageToTheServer(){
    SendEmergencyMessage.sendMessageToTheServer(
            lastLocation.getTimeAtLastLocation(),
            lastLocation.getLatitudeAtLastLocation(),
            lastLocation.getLongitudeAtLastLocation(),
            typeEmergency,
            this
    );
    Log.v(MyTAG, "Message sent to the server");
}
```

Figure 35: EmergencyService methods 1

```
private void sendEmergencySMS(){
    refreshUserPreferences();
    if(switchSmsEmergency.equalsIgnoreCase(on)){
        SendEmergencyMessage.sendSMS(
                lastLocation.getLatitudeAtLastLocation(),
                lastLocation.getLongitudeAtLastLocation(),
                typeEmergency,
                this,
                prefs.getString(PreferencesTypes.emergencyPhoneNumberSms.toString(), PreferencesTypes.no_phone.toString()),
                prefs.getString(PreferencesTypes.medical_info.toString(), "")
        );
    }
}

public boolean hasActiveInternetConnection() {
    ConnectivityManager connectivityManager
            = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo activeNetworkInfo = connectivityManager.getActiveNetworkInfo();
    return activeNetworkInfo != null && activeNetworkInfo.isConnected();
}
```

Figure 36: EmergencyService methods 2

3.7.3. SendEmergencyMessage

This class is responsible for sending the messages.

In the case of the SMS, it is responsible for composing and sending the message.

In the case of the server it launches a Thread to do it because the task of creating a csv file with the format we want is more complex and has to be a class intended only to do it, moreover, in this way is identical to sending messages to the server by the accelerometer's and the gyroscope's readings so the code is more homogeneous.

```
public class SendEmergencyMessage {

    private static final String MyTAG = "MyTAG";

    public static void sendMessageToTheServer(String time, String lat, String lon, String type, Context context){

        double latitude = Double.parseDouble(lat);
        double longitude = Double.parseDouble(lon);

        IEmergencyData IEmergencyData = new EmergencyData();
        IEmergencyData.setData(time, latitude,longitude, type);

        new EmergencyDataStorage(context).execute(IEmergencyData);
    }

    public static void sendSMS(String lat, String lon, String type, Context context, String phone, String medicalInfo){

        if(type.equalsIgnoreCase(EmergencyTypes.EmergencyButton.toString())){
            sendSmsEmergencyButton(lat, lon, context, phone, medicalInfo);
        }

        else if(type.equalsIgnoreCase(EmergencyTypes.EmergencyAccelerometer.toString())){
            sendSmsPossibleEmergency(lat,lon, context, phone, medicalInfo);
        }
    }
```

Figure 37: SendEmergencyMessage Class 1

```
    private static void sendSmsEmergencyButton (String lat, String lon, Context context, String phone, String medicalInfo){

        if(!phone.equalsIgnoreCase(PreferencesTypes.no_phone.toString()) && phone.length()>0){
            String message = context.getString(R.string.alertMessageSMSButton);
            message = message +" "+ lat + ", " + lon;

            if(!medicalInfo.equalsIgnoreCase("") || !medicalInfo.isEmpty()){
                message = message + ". Medical Info: " + medicalInfo;
            }

            composeAndSendTheSms(message, phone, context);
        }
    }

    private static void sendSmsPossibleEmergency (String lat, String lon, Context context, String phone, String medicalInfo){

        if(!phone.equalsIgnoreCase(PreferencesTypes.no_phone.toString()) && phone.length()>0){
            String message = context.getString(R.string.possibleAlertMessageSMS);
            message = message +" "+ lat + ", " + lon;

            if(!medicalInfo.equalsIgnoreCase("") || !medicalInfo.isEmpty()){
                message = message + ". Medical Info: " + medicalInfo;
            }

            composeAndSendTheSms(message, phone, context);
        }
    }
```

Figure 38: SendEmergencyMessage Class 2

```
private static void composeAndSendTheSms(String message, String phone, Context context) {
    final String SENT_SMS_ACTION            =   "SENT_SMS_ACTION";
    final String DELIVERED_SMS_ACTION       =   "DELIVERED_SMS_ACTION";

    //Create the setIntent parameter
    Intent sentIntent = new Intent(SENT_SMS_ACTION);
    PendingIntent sentPI = PendingIntent.getBroadcast(context, 0, sentIntent, 0);
    //Create the deliveryIntent parameter
    Intent deliveryIntent = new Intent(DELIVERED_SMS_ACTION);
    PendingIntent deliverPI = PendingIntent.getBroadcast(context, 0, deliveryIntent, 0);

    SmsManager smsManager = SmsManager.getDefault();

    try{
        ArrayList<String> multipartSmsText = smsManager.divideMessage(message);
        int smsSize = multipartSmsText.size();

        //Create the arraylist PendingIntents for use it.
        ArrayList<PendingIntent> sentPiList = new ArrayList<>(smsSize);
        ArrayList<PendingIntent> deliverPiList = new ArrayList<>(smsSize);

        for (int i=0; i<smsSize; i++) {
            sentPiList.add(sentPI);
            deliverPiList.add(deliverPI);
        }
```

Figure 39: SendEmergencyMessage Class 3

```
        //Try to send the sms message
        smsManager.sendMultipartTextMessage(phone, null, multipartSmsText, sentPiList, deliverPiList);
    }catch (Exception ex){
        Toast.makeText(context,"Error sending the SMS", Toast.LENGTH_SHORT).show();
    }
}
}
```

Figure 40: SendEmergencyMessage Class 4

In addition, two types of SMS can be made, depending on whether the emergency signal has been pressed the button and therefore the user is asking for help, or if on the other hand, the message is automatically sent and may be a false alarm.

The template of the different SMS and all the text is on the Strings file, and it is available on Spanish, Catalan and English, as all the texts included in the application.

3.7.4. AndroidManifest.xml

When we introduce new activities must be added in the AndroidManifest and the same goes for services that have to be included as well.

```
<activity
    android:name=".CrashDetection.Acitivities.EmergencyMenuActivity"
    android:label="@string/title_activity_emergency" />
<activity
    android:name=".CrashDetection.Acitivities.SettingsCrashDetectionActivity"
    android:label="@string/title_activity_settings_crash_detection" />
<activity
    android:name=".StepCounter.StepCounterActivity"
    android:label="@string/title_activity_step_counter"></activity>
```

Figure 41: AndroidManifest Activities

```
<service
    android:name=".CrashDetection.HandlersAndServices.AccelerometerServiceListener"
    android:enabled="true" />
<service
    android:name=".CrashDetection.HandlersAndServices.EmergencyService"
    android:enabled="true" />
```

Figure 42: AndroidManifest Services

In addition, as already mentioned, we must add all permissions required by the application.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.USE_CREDENTIALS" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Figure 43: AndroidManifest Permissions

Furthermore, it is necessary to add to the AndroidManifest the accelerometer sensor. However, as the application already incorporated it, is not necessary to introduce it again.

```
<uses-features
    android:name="android.hardware.sensor.accelerometer"
    android:required="true" />
```

Figure 44: AndroidManifest uses-features

4. Step Counter Module

4.1. Step Counter

Our goal is to create a module in our project that can count the steps of the user. From the number of steps, mileage and calories burned are estimated.

4.2. Step Counter Sensor

Since the Android KitKat (API 19) has been incorporated a hardware sensor using an algorithm implemented by Google gives us back the number of steps that have been made. However some manufacturers do not include this functionality, such as a Doogee DG310 which is a cheap device from a Chinese manufacturer, despite of having Android KitKat does not have this sensor. Nevertheless, in a Nexus 5, a cell phone that was released in late 2013, is working properly.

4.2.1. Devices using MobilitApp

In Google Play Developer Console we can see the android version of all the users that have installed the app. On August 17,2016 we have the following statistics:

- Android 4.4: 37,5%
- Android 5.1: 25,0%
- Android 6.0: 37,5%

Moreover, from Android Studio we can see the use of each version of Android.

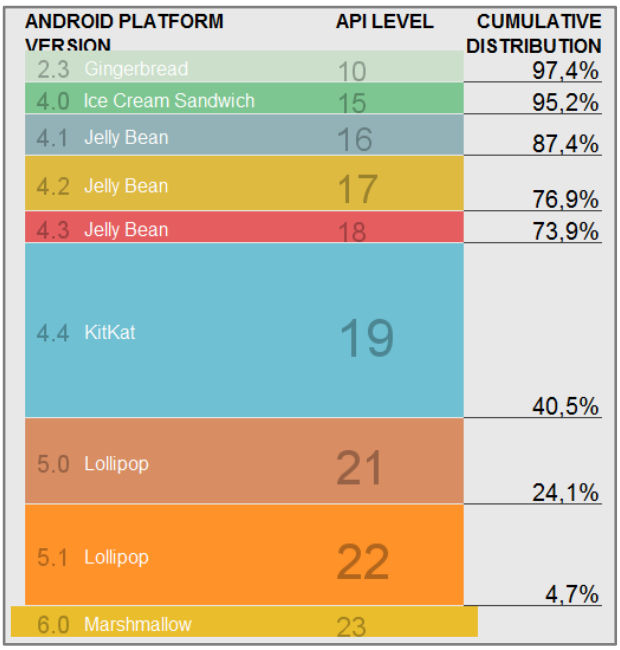

Figure 45: Android Versions Use

Although the pedometer only works for android 4.4 or higher, we can serve all the users who already MobilitApp has and cover much of the market for new users who may download it.

4.3. Approach of the solution

From the outset, the solution has been raised for it to be modular and scalable.

Corresponding to the step counter module, it has been designed with the aim that can be incorporated into another project and make it work without having to touch the logical layer or the business logic, we only should adapt the UI.

This module is in a separate package and it only has two classes. The reason of use only two classes is because the business logic, the part that count the steps, is given by the OS so we only need a activity to show the steps and the calculated data as distance or calories.

4.3.1. Architecture of the solution

In the next image we present the architecture of our Step Counter module:

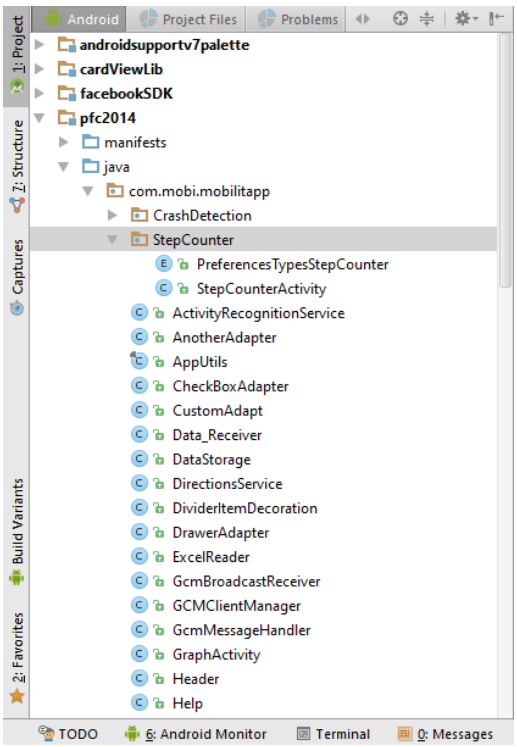

Figure 46: Step Counter Packages

The architecture is much simple in this case.

We have an activity to show the data and an enum class for manage the SharedPreferences.

4.3.2. Application

The step counter module has been developed inside the MobilitApp project, because of the simplicity of the architecture.

4.3.2.1. Load settings

The main class is: MainActivity.java is in this class where personal user settings are loaded and the service will be launched if it is necessary.

The user settings are saved by SharedPreferences:

```
prefs = getSharedPreferences("prefs", Context.MODE_PRIVATE);
```

These preferences were designed to save user settings so it has seemed good idea to take advantage of these resources.

```
if(prefs.getString(PreferencesTypesStepCounter.stepCounter.toString(),PreferencesTypesStepCounter.off.toString())
        .equalsIgnoreCase(PreferencesTypesStepCounter.off.toString())){
    prefs.edit().putString(PreferencesTypesStepCounter.stepCounter.toString(),
            PreferencesTypesStepCounter.off.toString()).commit();
}

if(prefs.getString(PreferencesTypesStepCounter.totalSteps.toString(),"")
        .equalsIgnoreCase("")){
    prefs.edit().putString(PreferencesTypesStepCounter.totalSteps.toString(),
            "0").commit();
}

if(prefs.getString(PreferencesTypesStepCounter.todaySteps.toString(),"")
        .equalsIgnoreCase("")){
    prefs.edit().putString(PreferencesTypesStepCounter.todaySteps.toString(),
            "0").commit();
}

if(prefs.getString(PreferencesTypesStepCounter.todayDataTime.toString(),"")
        .equalsIgnoreCase("")){
    prefs.edit().putString(PreferencesTypesStepCounter.todayDataTime.toString(),
            PreferencesTypesStepCounter.noDataTime.toString()).commit();
}

if(prefs.getString(PreferencesTypesStepCounter.lastCounterStepsFromSensor.toString(),"")
        .equalsIgnoreCase("0")){
    prefs.edit().putString(PreferencesTypesStepCounter.lastCounterStepsFromSensor.toString(),
            "0").commit();
}
```

Figure 47: MainActivity.java

This code will must be called every time MobilitApp is open, so we have to locate it in his method:

```
protected void onCreate(Bundle savedInstanceState)
```

4.3.2.2. Graphic User Interfaces

Only one GUI is needed, the interface is created to turn the step counter on or off and to show the data.

Screen's diagram:

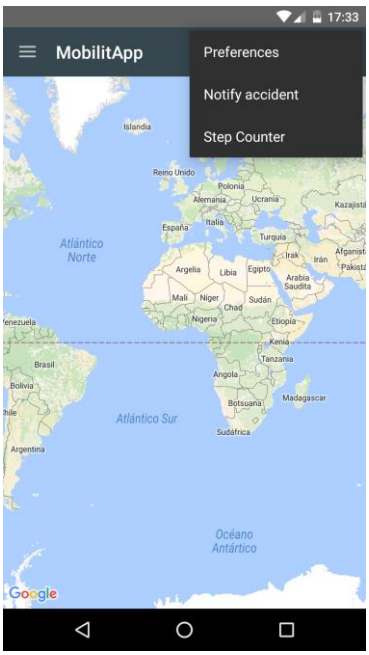

Figure 50: Main Activity

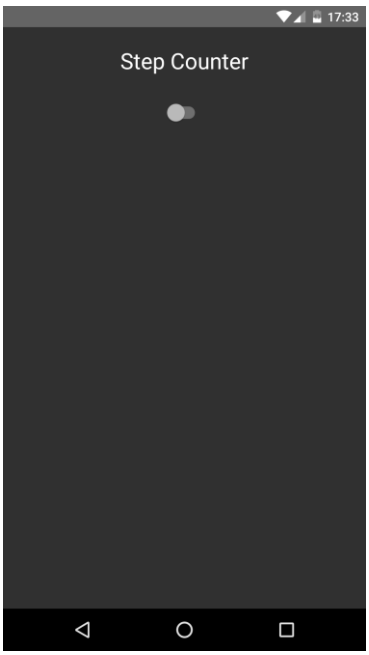

Figure 49: Step Counter off

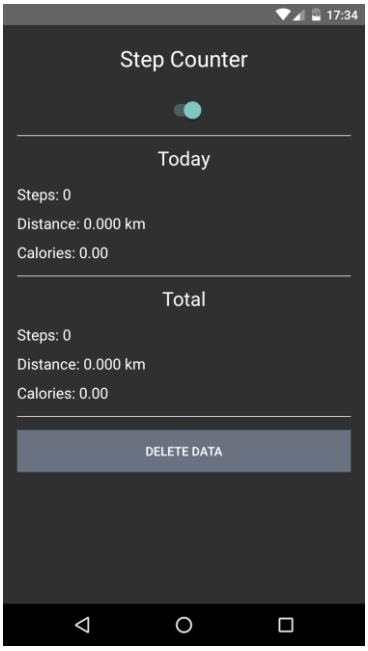

Figure 48: Step Counter on

Once activated, the application count all the steps made by the user. MobilitApp show all the steps made since the last time the data was reset (using the "Delete Data" button) and also all the steps that have been made during the current day are displayed.

Here we have an example of use:

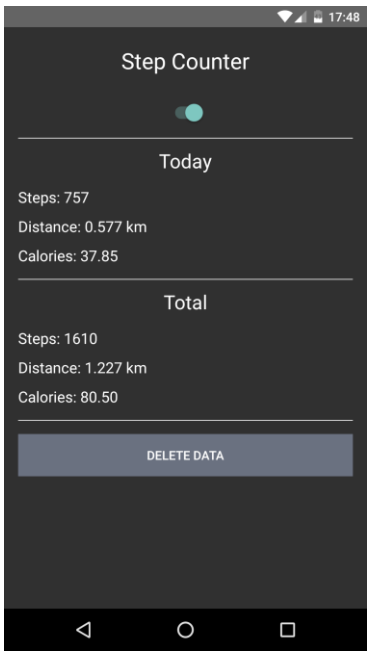

Figure 51: Step Counter Use

4.4. Development

The first class that was developed was PreferencesTypesStepCounter. This class, like we saw with the Crash Detection Module is used to unify all the keys that we use with SharedPreferences.

```
public enum PreferencesTypesStepCounter {
    stepCounter,
    on,
    off,
    totalSteps,
    todaySteps,
    todayDataTime,
    noDataTime,
    lastCounterStepsFromSensor
}
```

Figure 52: PreferencesTypesSetpCounter

Once this is done, it is time to perform the activity with which the user will interact.

To do this, we must first understand how the sensor that Android KitKat provides works.

4.4.1. How the sensor Works

To use the sensor, first thing we need is a SensorManager:

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

Now, we can check if the device has the step counter sensor:

```
if(mSensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER) != null)
```

If so, we will keep the sensor:

```
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
```

Finally, we need to start up the sensor:

```
mSensorManager.registerListener(this, mSensor,
SensorManager.SENSOR_DELAY_FASTEST);
```

And programe it to stop the sensor when desired:

```
mSensorManager.unregisterListener(this);
```

To use the sensor, the activity we use must implement the interface SensorEventListener, this interface force us to implement the method:

```
public void onSensorChanged(SensorEvent sensorEvent)
```

Every time a new step is detected, this method will be executed and we will able to reach the number of steps through:

```
sensorEvent.values[0]
```

4.4.2. Step Counter Activity

For the development of this activity, we have created several small methods to divede the class into small fragments, this fragments are small pieces of code that handle a single task.

```
@Override
protected void onCreate(Bundle savedInstanceState) {...}
@Override
protected void onResume() {...}

private void unregisterListenerSensor(){...}
@Override
public void onSensorChanged(SensorEvent sensorEvent) {...}
@Override
public void onAccuracyChanged(Sensor sensor, int i) {...}

private void loadConfigurations() {...}

private void loadData(){...}

private String getTodayDate(){...}

private void refreshValues(){...}

public void resetStepData(View v){...}

private void dialogNoStepCounterHardware(){...}

private void recalculateAndRefreshSteps(int steps){...}

private void refreshStepsValuesToday(int steps){...}

private void refreshStepsValuesTotal(int steps){...}

private double calculateCaloriesBySteps (int steps){...}

private double calculateKmBySteps (int steps){...}
```

Figure 53: StepCounterActivity

In the future, this will allows us to change that part of code to another one with a different implementation.

For example, in the case of the calculations of calories and kilometres it is not taken into account the height or weight of the user, depending on the height the mileage may vary and the same goes with weight and calories. Thanks to having the calculation in an isolated method, it can be exchanged with another. If the lines of code were embedded within the method onSensorChanged could work the same but would be more difficult to change it, this is strongly related to the principle of open/close from the SOLID principles.

In the same way, it could help us, using some user settings, to create a new method to calculate the distance in miles instead of kilometers and depending on these user settings call one method or the other one.

5. Analysing Mobility Data

We have analyzed the data obtained through the accelerometer users traveling by subway and train.

We have selected a few svc files, specifically, those who had more number of samples in order to perform a more thorough analysis.

Moreover, all the samples are on different days and at different hours.

We have compared the collected signal with a sinusoid. We have been varying frequency and looking which is really fits with all cases.

5.1. Subway

The signal from the accelerometer corresponding to the Z axis is the more random.

The X axis looks like a sinusoid of frequency 0.189 Hz, this means a period of 5.3 seconds.

The Y axis resembles a 0.063 Hz frequency sinusoid with period of 15.9 seconds.

5.1.1. Sample 1

April 20, 2015 at 19:27.

Acceleration of the accelerometer on the axis X compared with a sinusoid of 0.189 Hz.

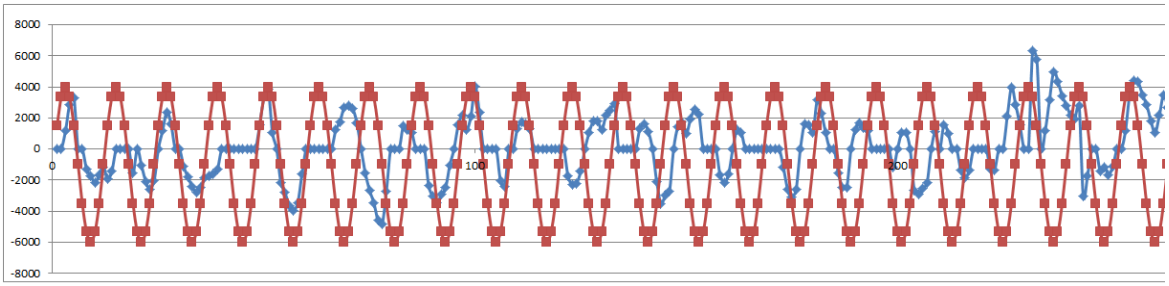

Figure 54: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 1 Part 1

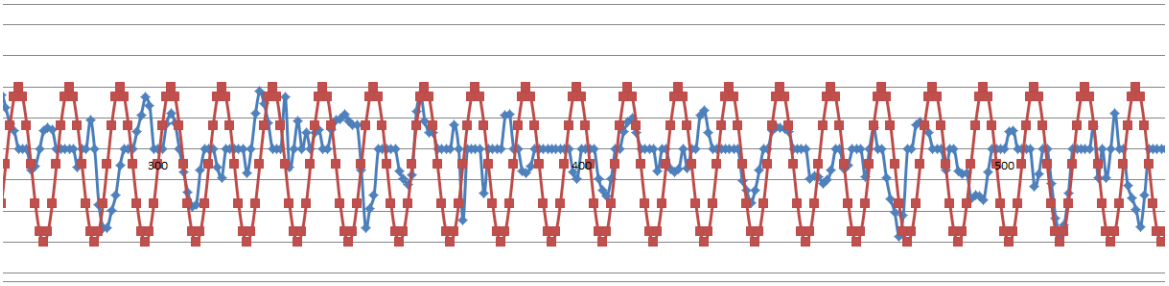

Figure 55: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 1 Part 2

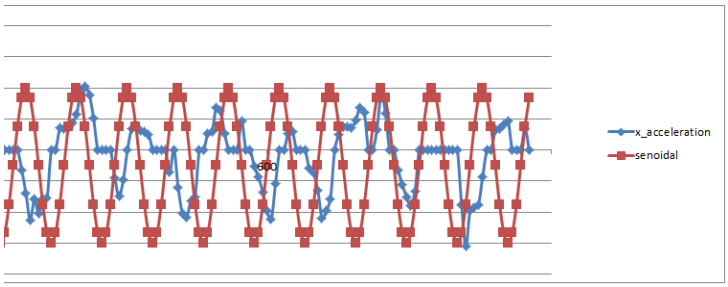

Figure 56: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 1 Part 3

Acceleration of the accelerometer on the axis Y compared with a sinusoid of 0.063 Hz.

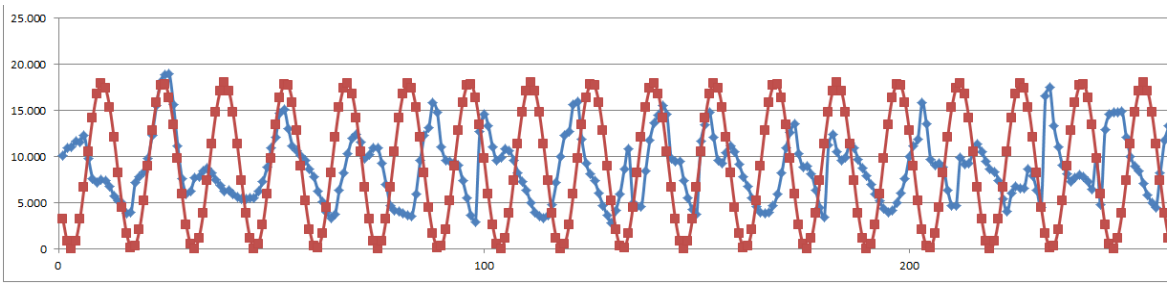

Figure 57: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 1 Part 1

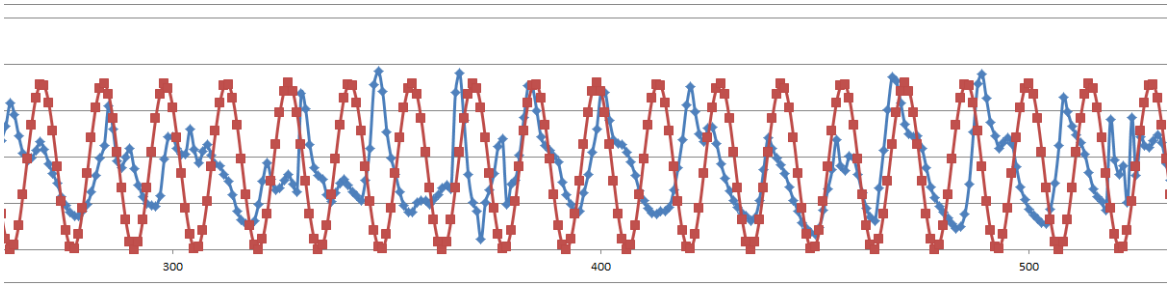

Figure 58: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 1 Part 2

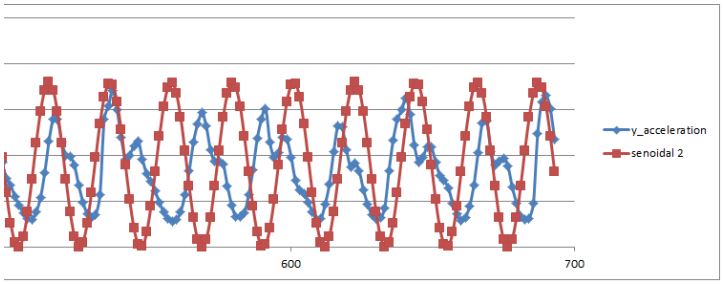

Figure 59: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 1 Part 3

Acceleration of the accelerometer on the axis Z, it is too aleatory.

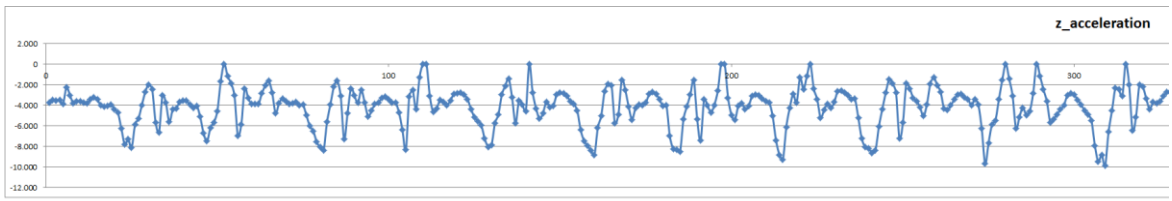

Figure 60: Acceleration Axis Z Sample 1 Part 1

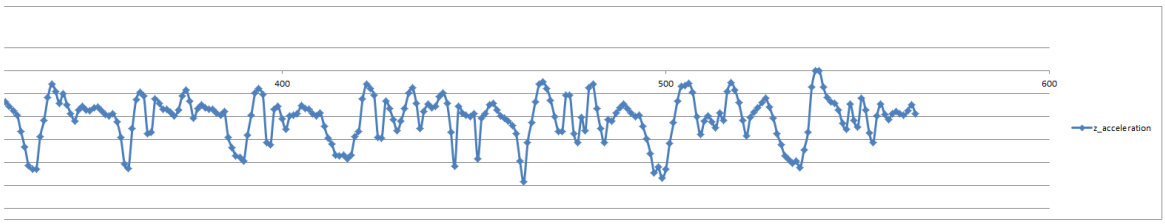

Figure 61: Acceleration Axis Z Sample 1 Part 2

5.1.2. Sample 2

April 27, 2015 at 6:50.

Acceleration of the accelerometer on the axis X compared with a sinusoid of 0.189 Hz.

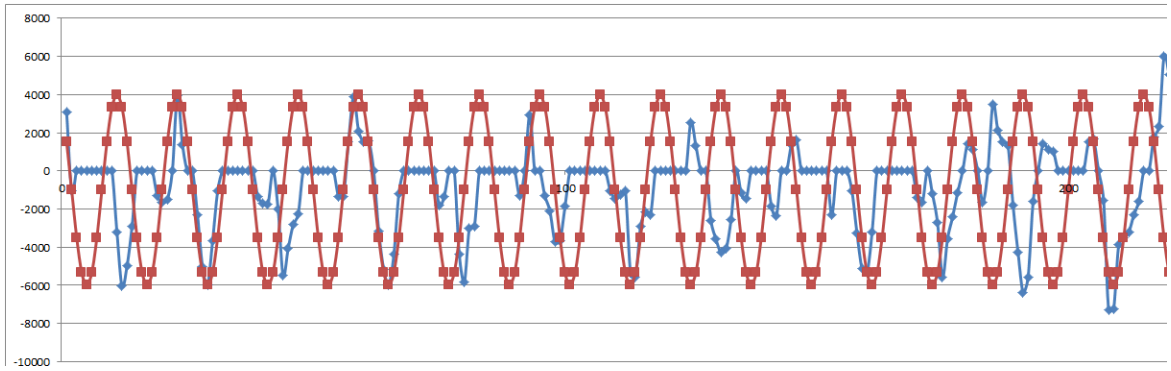

Figure 62: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 2 Part 1

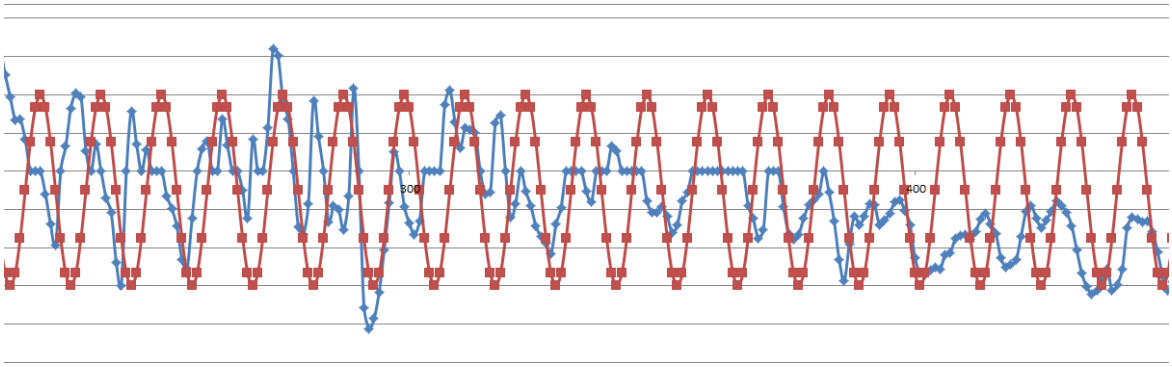

Figure 63: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 2 Part 2

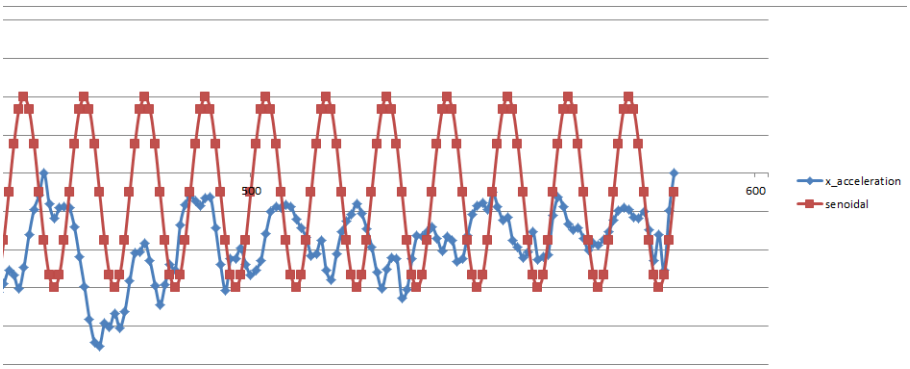

Figure 64: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 2 Part 3

Acceleration of the accelerometer on the axis Y compared with a sinusoid of 0.063 Hz.

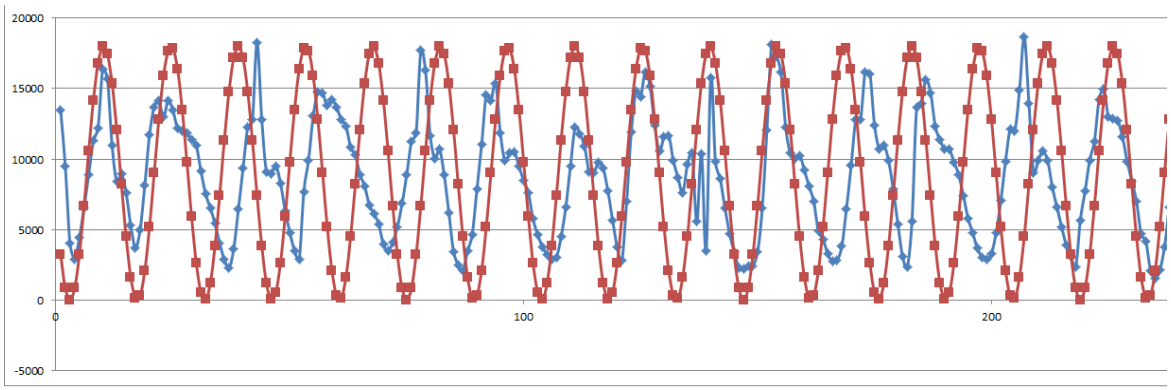

Figure 65: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 2 Part 1

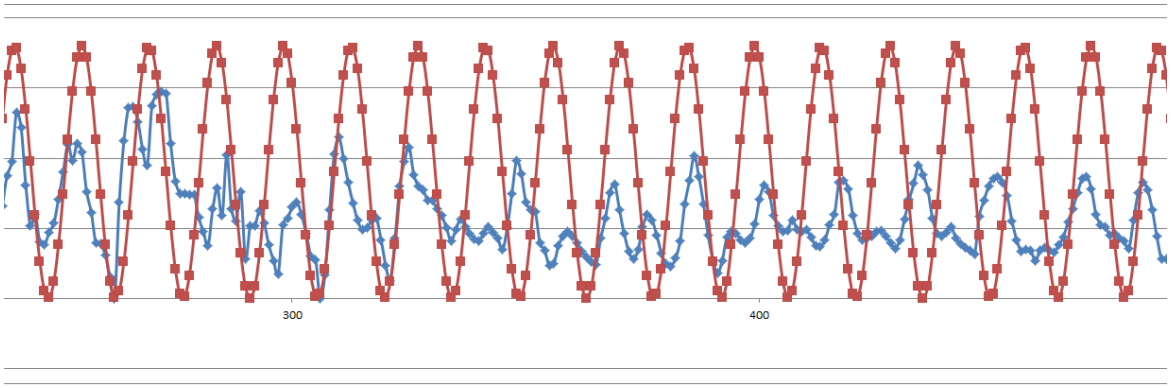

Figure 66: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 2 Part 2

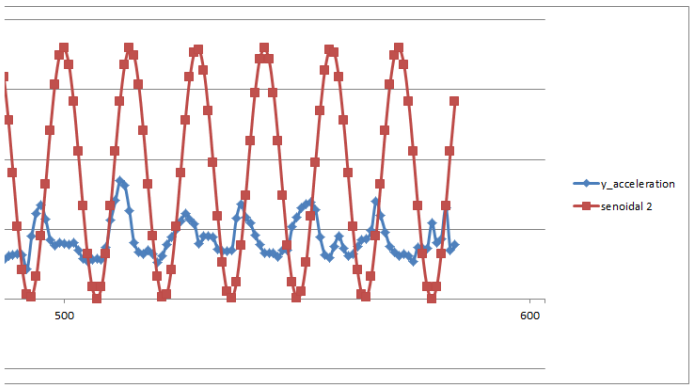

Figure 67: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 2 Part 3

Acceleration of the accelerometer on the axis Z, it is too aleatory.

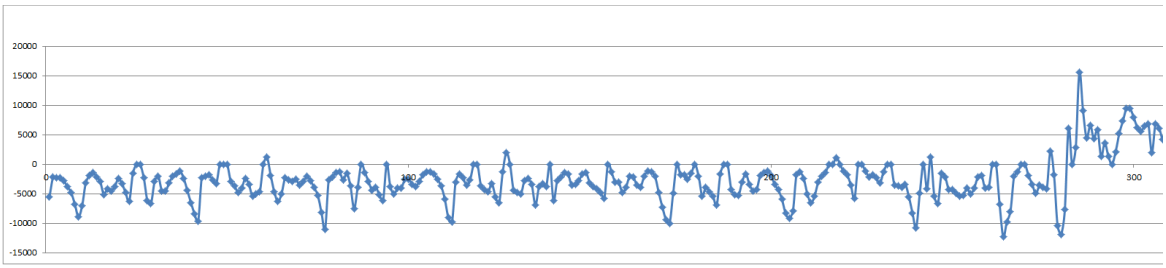

Figure 68: Acceleration Axis Z Sample 2 Part 1

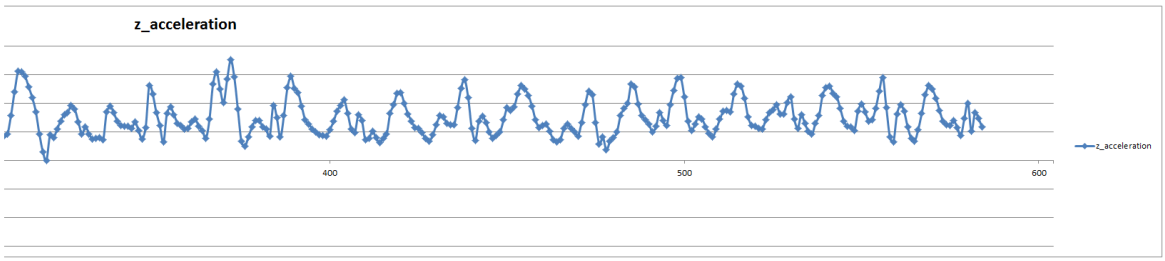

Figure 69: Acceleration Axis Z Sample 2 Part 2

5.1.3. Sample 3

April 23, 2015 at 21:20.

Acceleration of the accelerometer on the axis X compared with a sinusoid of 0.189 Hz.

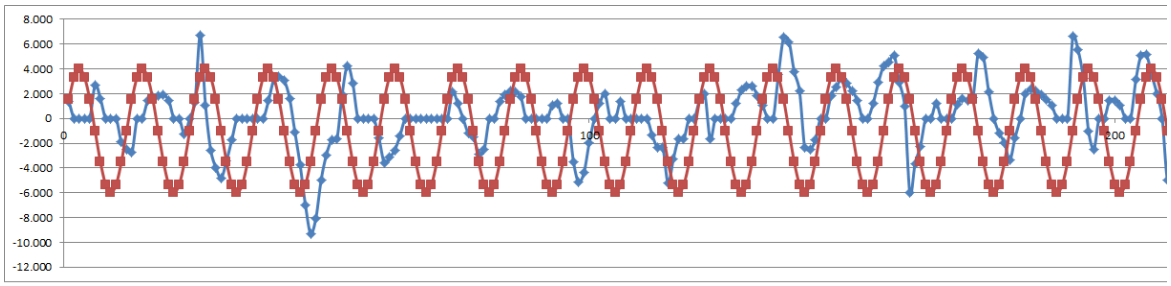

Figure 70: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 3 Part 1

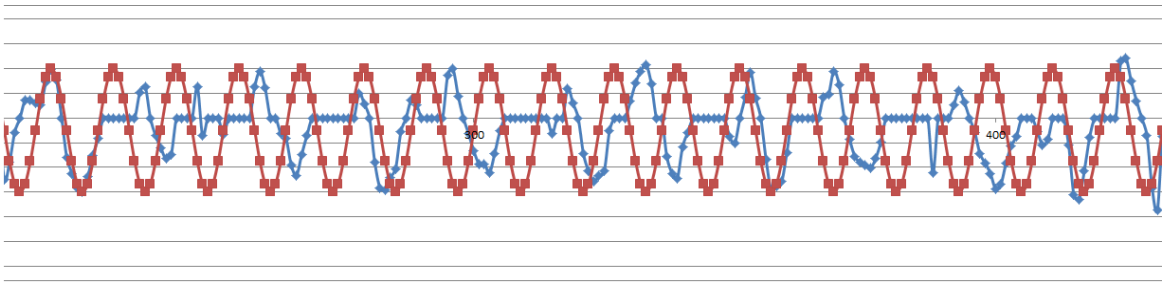

Figure 71: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 3 Part 2

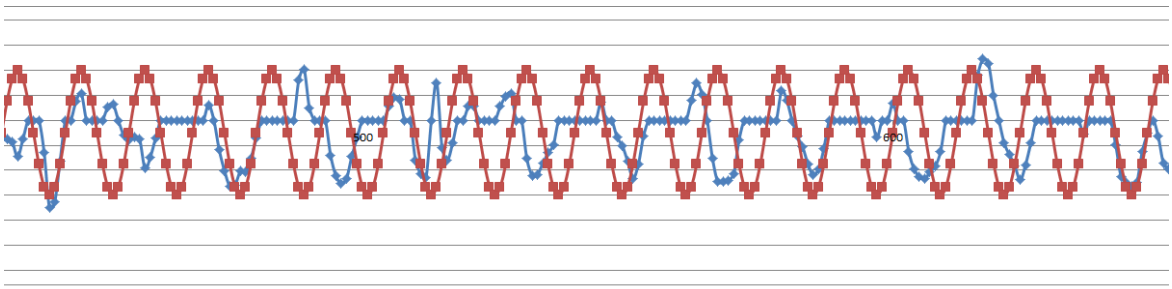

Figure 72: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 3 Part 3

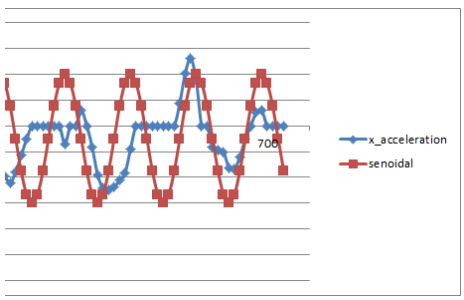

Figure 73: Acceleration Axis X vs Sinusoid 0.189 Hz Sample 3 Part 4

Acceleration of the accelerometer on the axis Y compared with a sinusoid of 0.063 Hz.

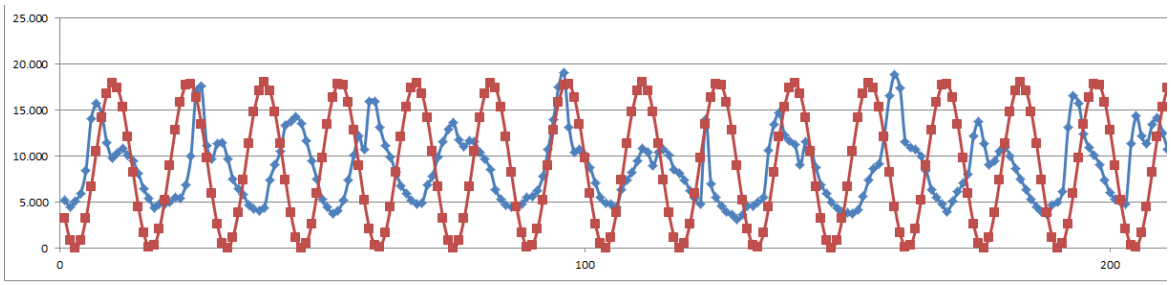

Figure 74: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 3 Part 1

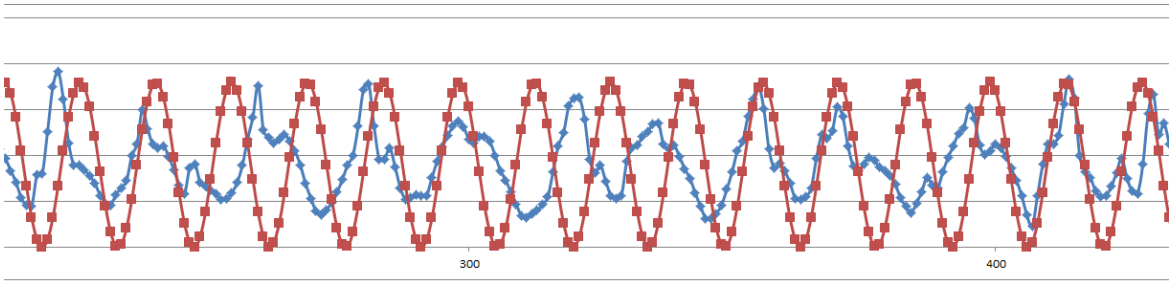

Figure 75: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 3 Part 2

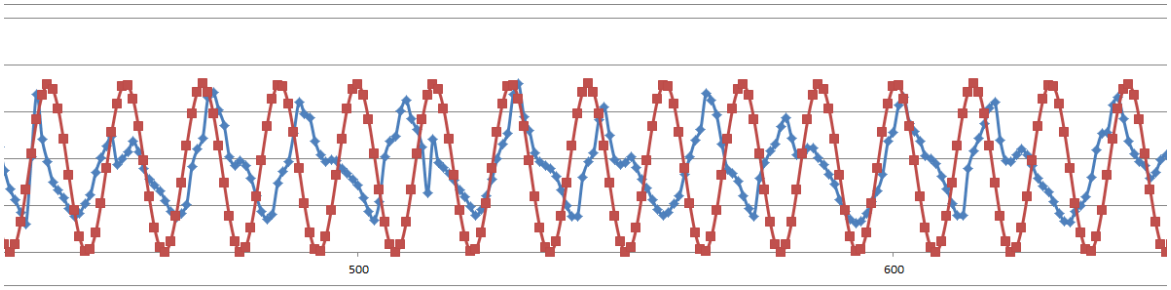

Figure 76: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 3 Part 3

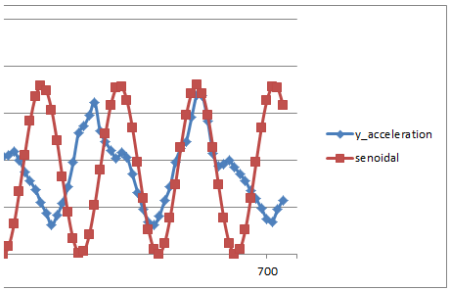

Figure 77: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 3 Part 4

Acceleration of the accelerometer on the axis Z, it is too aleatory.

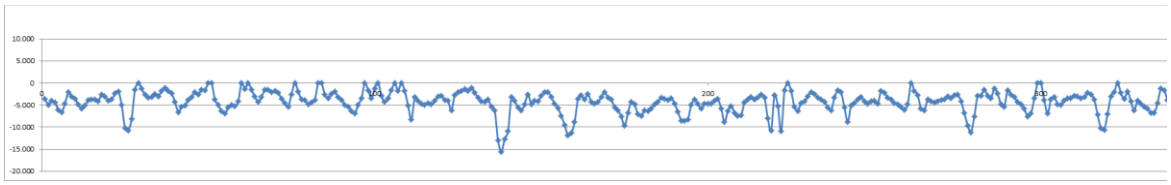

Figure 78: Acceleration Axis Z Sample 3 Part 1

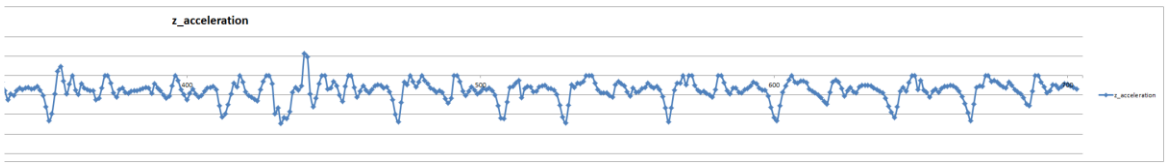

Figure 79: Acceleration Axis Z Sample 3 Part 2

5.1.4. Sample 4

April 11, 2015 at 18:25.

In the latter case, the 0.189 Hz sine wave is not well adapted to the acceleration in the X axis, however it looks like a 0.063 Hz frequency sinusoid as the Y axis.

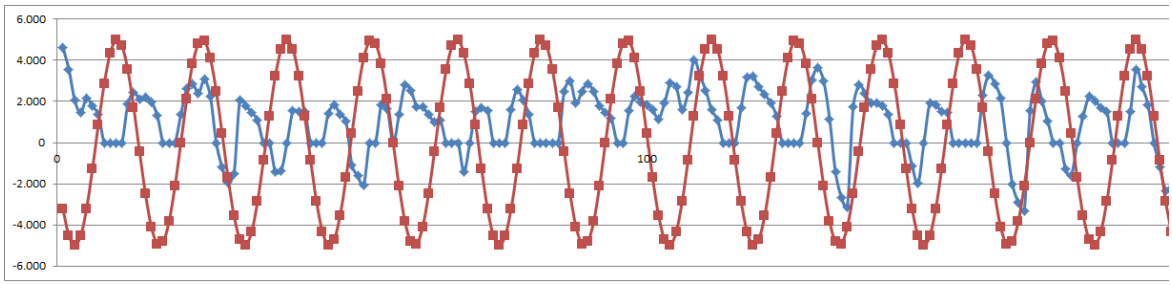

Figure 80: Acceleration Axis X vs Sinusoid 0.063 Hz Sample 4 Part 1

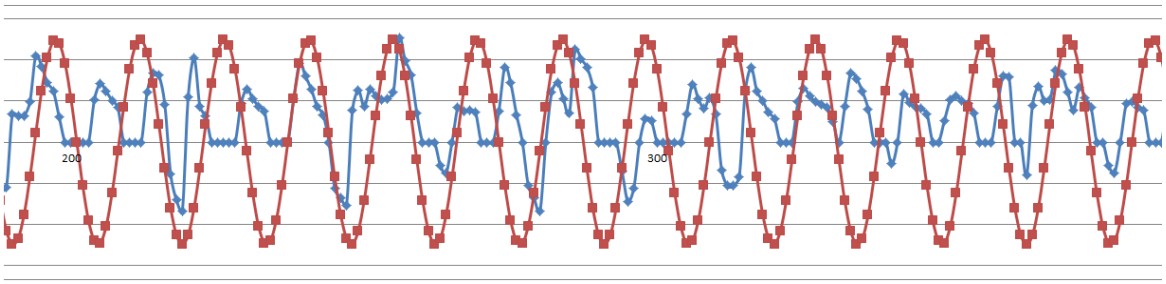

Figure 81: Acceleration Axis X vs Sinusoid 0.063 Hz Sample 4 Part 2

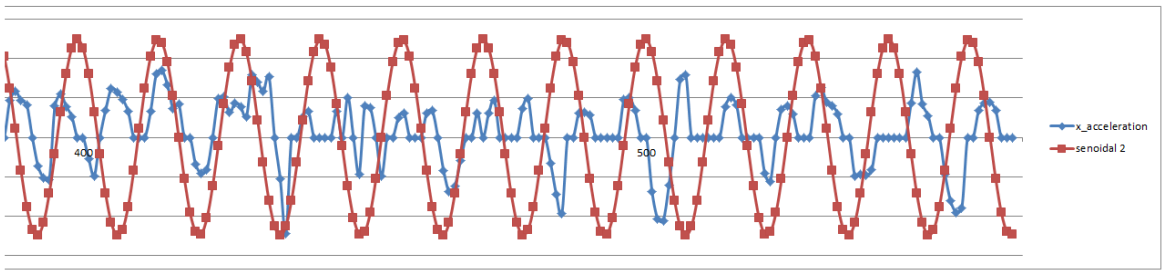

Figure 82: Acceleration Axis X vs Sinusoid 0.063 Hz Sample 4 Part 3

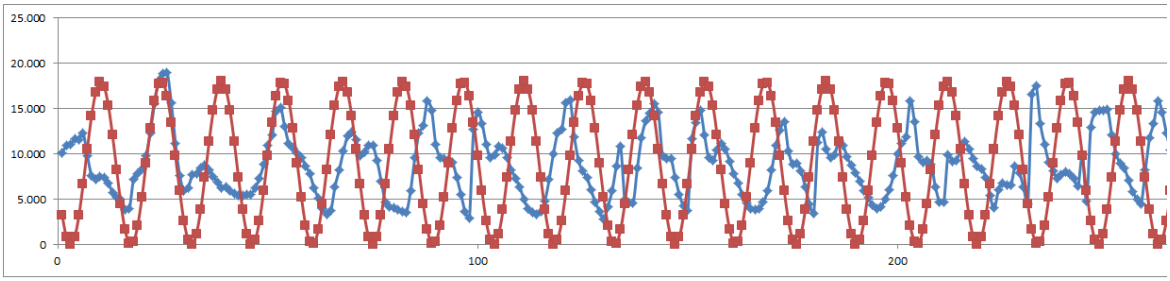

Figure 83: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 4 Part 1

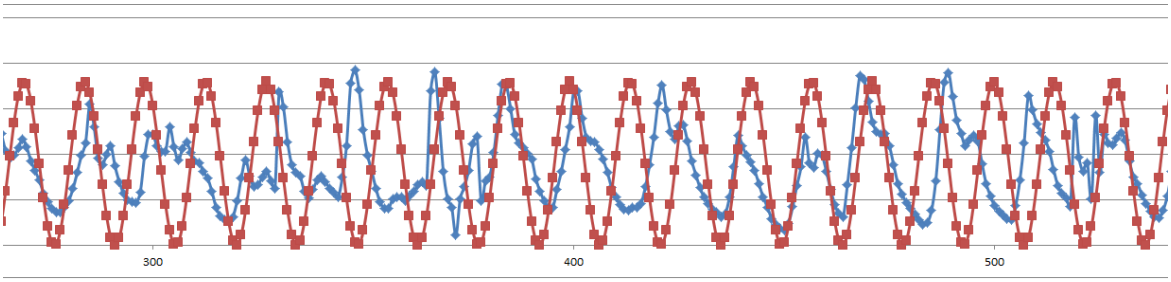

Figure 84: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 4 Part 2

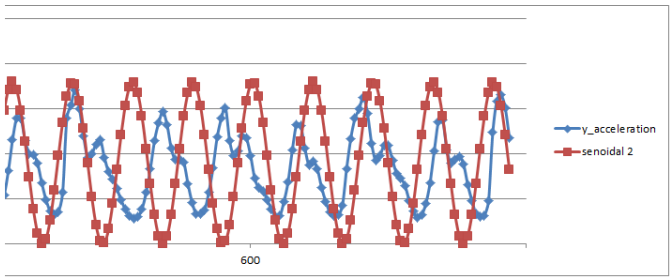

Figure 85: Acceleration Axis Y vs Sinusoid 0.063 Hz Sample 4 Part 3

Acceleration of the accelerometer on the axis Z, it is too aleatory.

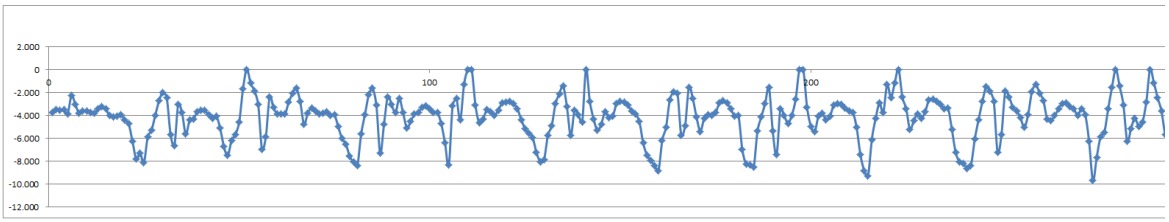

Figure 86: Acceleration Axis Z Sample 4 Part 1

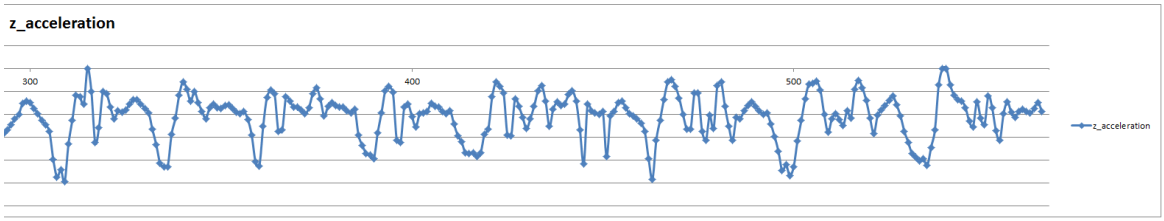

Figure 87: Acceleration Axis Z Sample 4 Part 2

5.1.5. Conclusions

The results comply with what we expected. We found a sine wave that fits quite well to the acceleration of the axis X and another sine wave that adapts quite well to the axis Y. On the other hand, we have seen that the Z axis is more random that the others. These results may vary depending on the position of the mobile but we always have three axes along which we can be helpful to identify the graphics that come from underground.

However these waves depend on many variables such as distance between stations, number of users, anomalies on public transport, etc. So they never are able to adapt exactly to any sine wave perfectly.

5.2. Train

In this case, we have sought a sine wave and try to adjust it with all cases. In the case of train signals coming from the accelerometer at axis X and Y are similar and the Z axis again becomes more random.

The sine wave that best fits with our samples is a sinusoidal of frequency 0.051 Hz.

5.2.1. Sample 1

March 16, 2015 at 17:56.

Acceleration of the accelerometer on the axis X and Y compared with a sinusoid of 0.051 Hz.

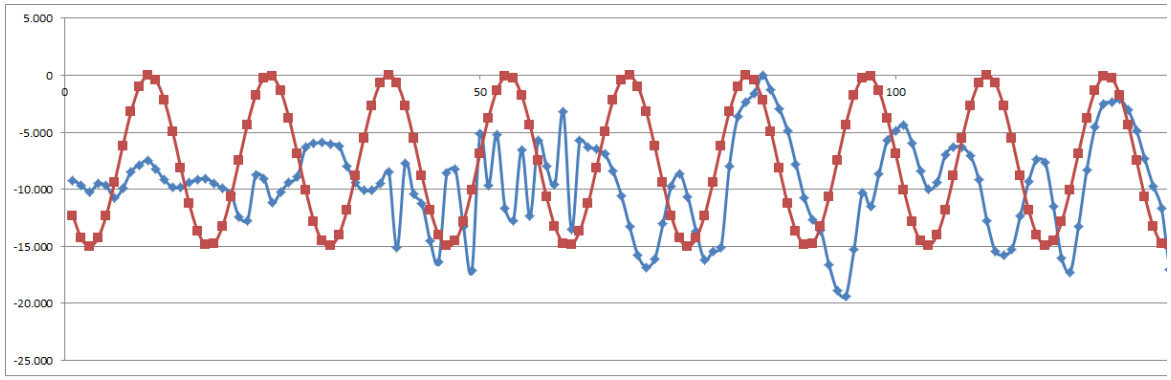

Figure 88: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 1 Part 1

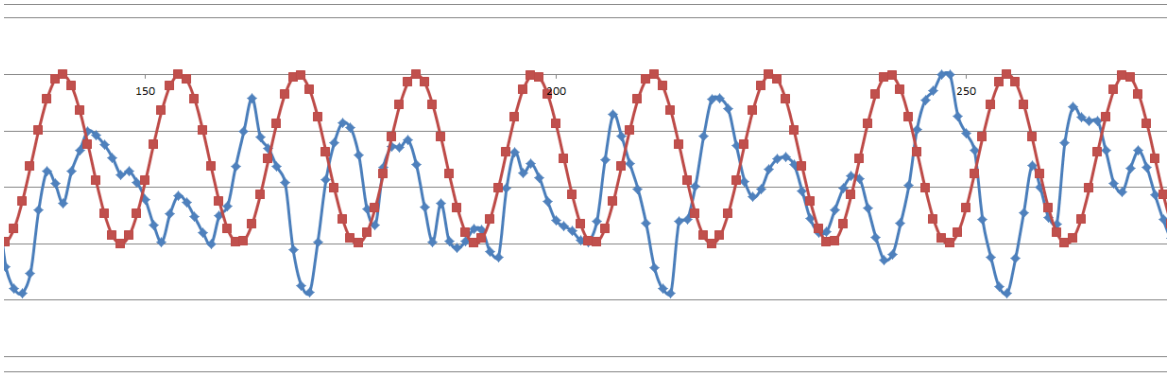

Figure 89: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 1 Part 2

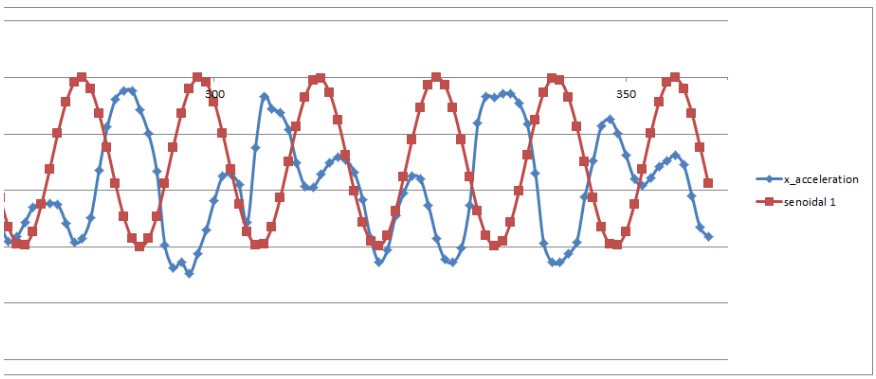

Figure 90: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 1 Part 3

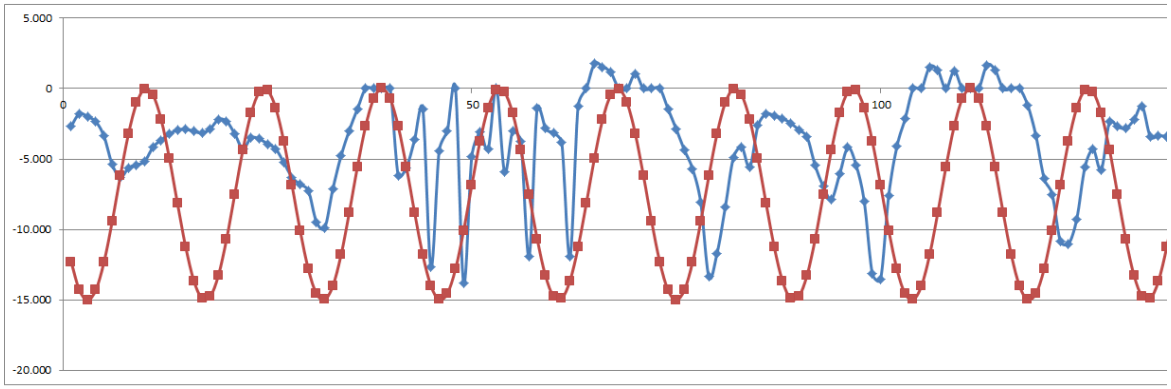

Figure 91: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 1 Part 1

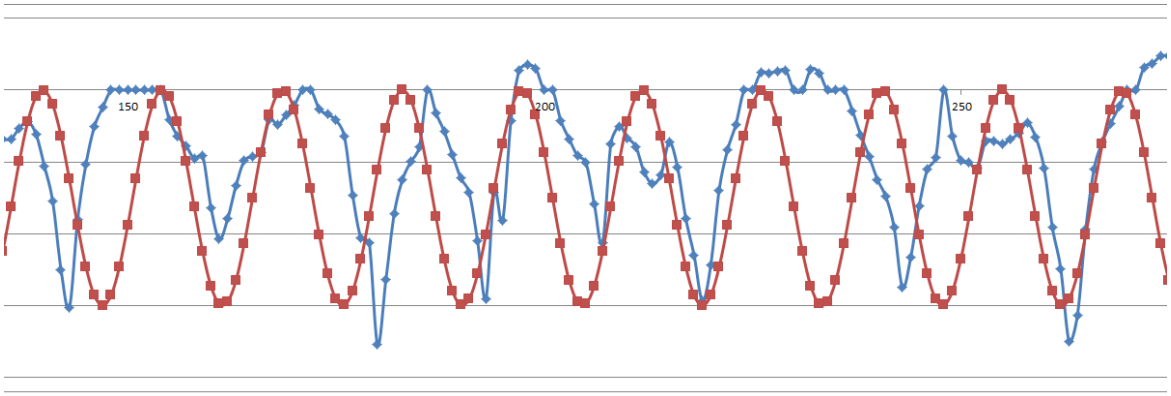

Figure 92: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 1 Part 2

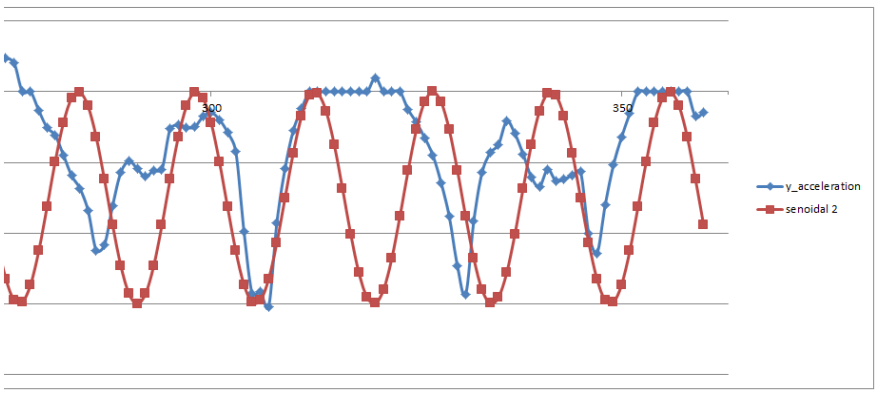

Figure 93: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 1 Part 3

Acceleration of the accelerometer on the axis Z, it is too aleatory.

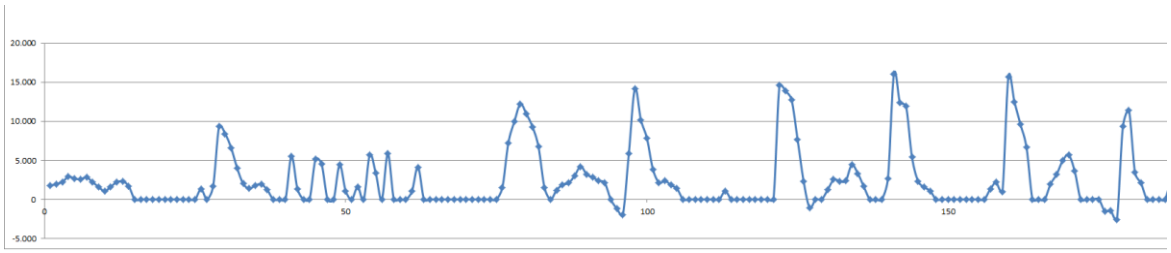

Figure 94: Acceleration Axis Z Sample 1 Part 1

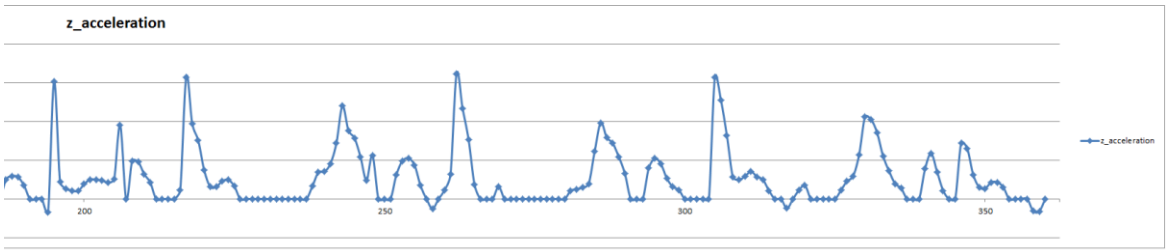

Figure 95: Acceleration Axis Z Sample 1 Part 2

5.2.2. Sample 2

March 16, 2015 at 16:53.

Acceleration of the accelerometer on the axis X and Y compared with a sinusoid of 0.051 Hz.

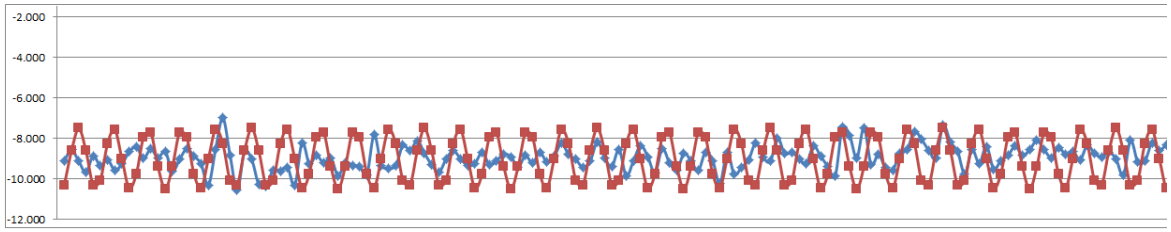

Figure 96: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 2 Part 1

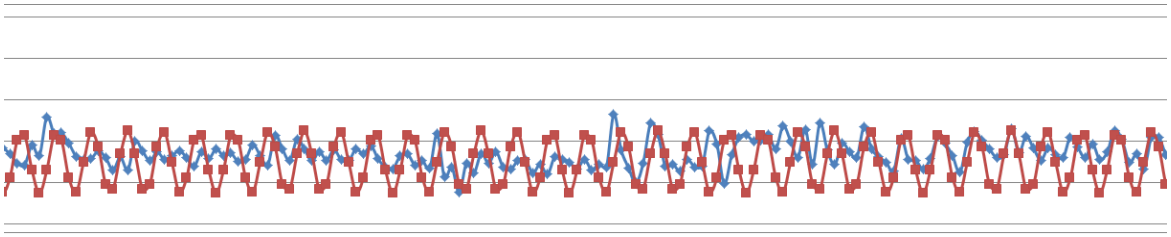

Figure 97: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 2 Part 2

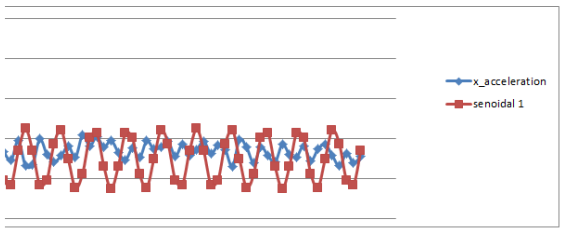

Figure 98: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 2 Part 3

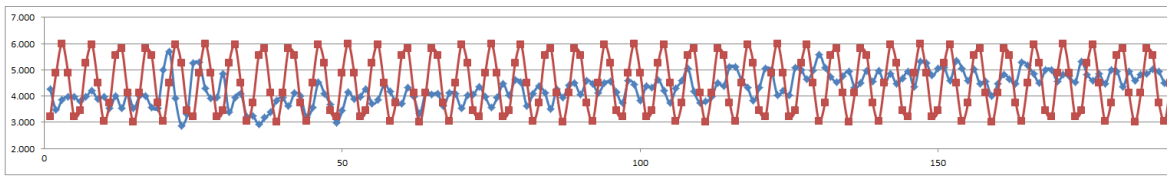

Figure 99: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 2 Part 1

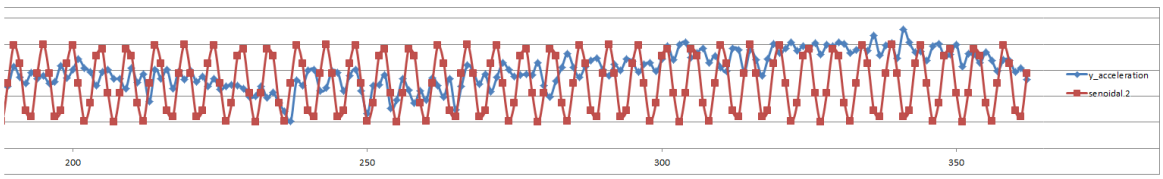

Figure 100: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 2 Part 2

Acceleration of the accelerometer on the axis Z, it is too aleatory.

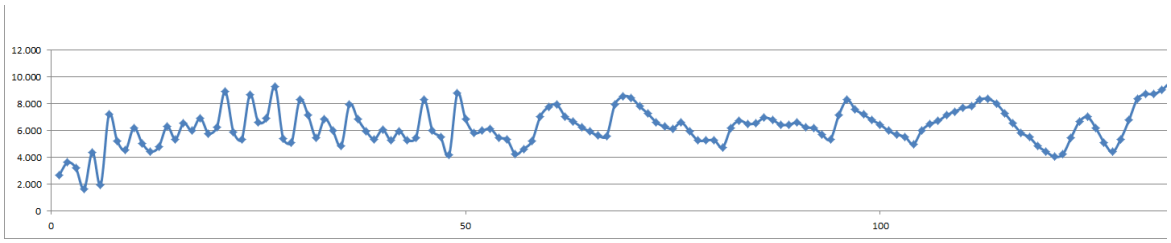

Figure 101: Acceleration Axis Z Sample 2 Part 1

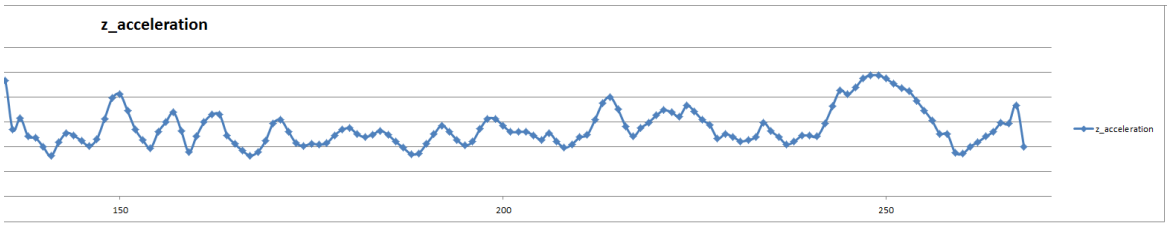

Figure 102: Acceleration Axis Z Sample 2 Part 2

5.2.3. Sample 3

March 16, 2015 at 17:00.

Acceleration of the accelerometer on the axis X and Y compared with a sinusoid of 0.051 Hz.

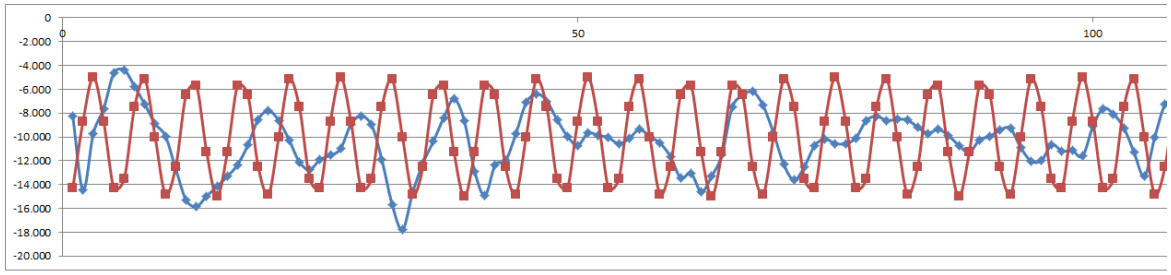

Figure 103: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 3 Part 1

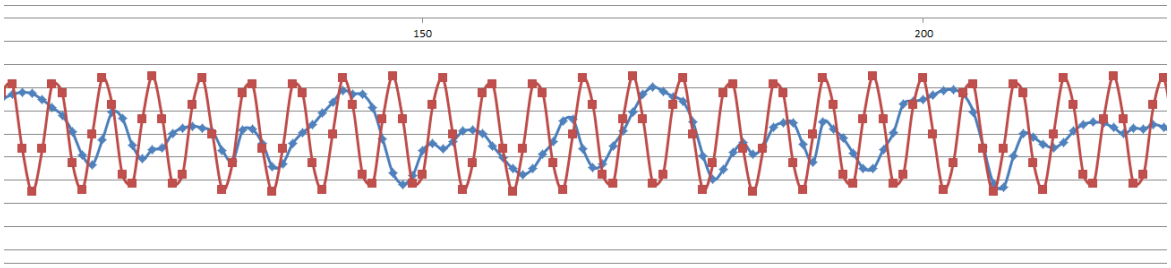

Figure 104: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 3 Part 2

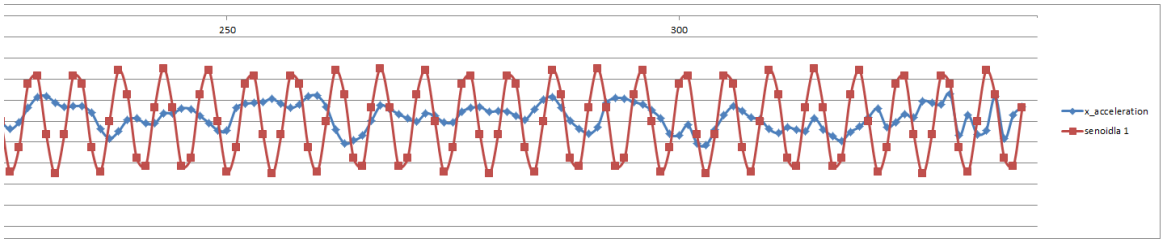

Figure 105: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 3 Part 3

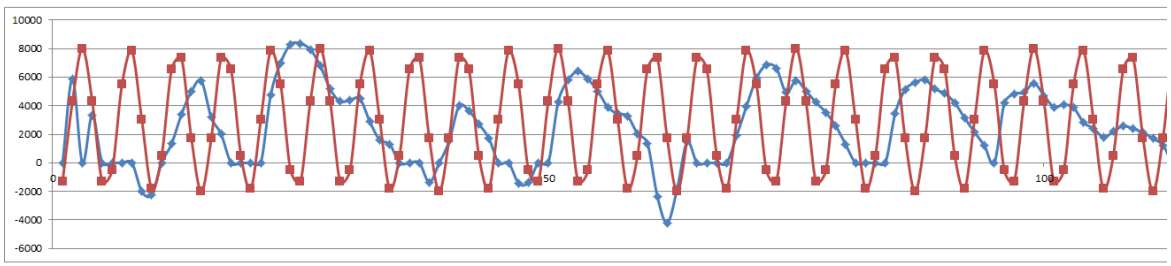

Figure 106: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 3 Part 1

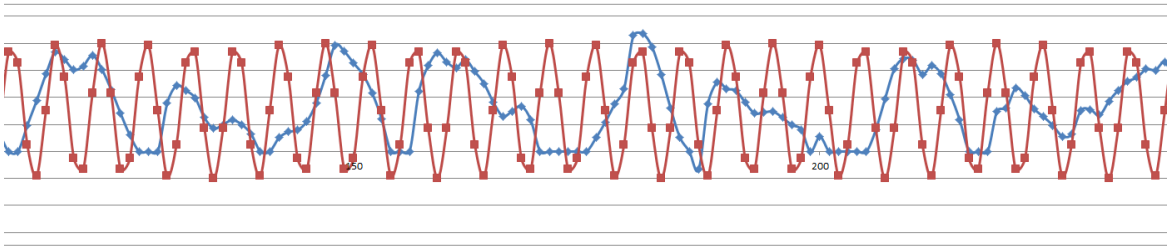

Figure 107: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 3 Part 2

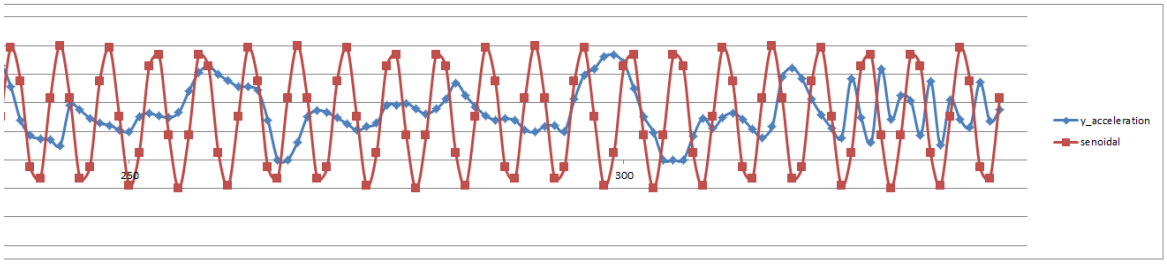

Figure 108: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 3 Part 3

Acceleration of the accelerometer on the axis Z, it is too aleatory.

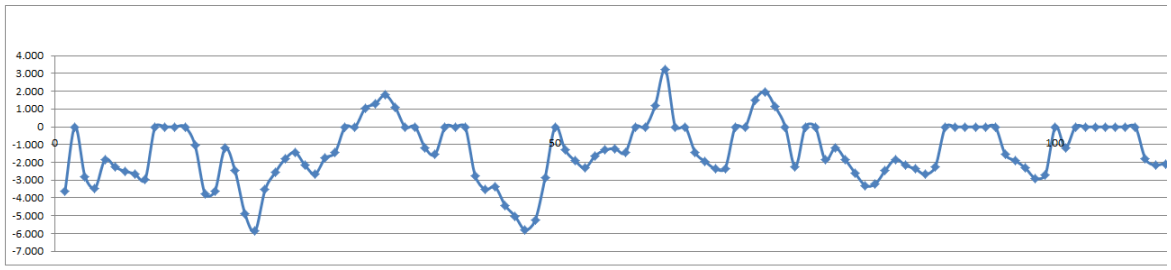

Figure 109: Acceleration Axis Z Sample 3 Part 1

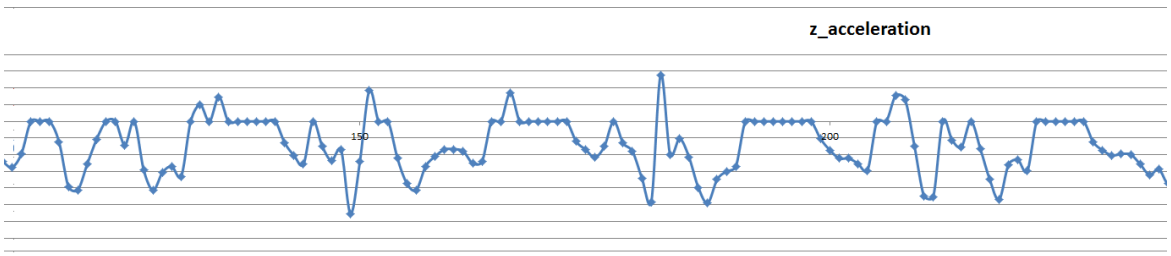

Figure 110: Acceleration Axis Z Sample 3 Part 2

5.2.4. Sample 4

March 16, 2015 at 19:08.

Acceleration of the accelerometer on the axis X and Y compared with a sinusoid of 0.051 Hz.

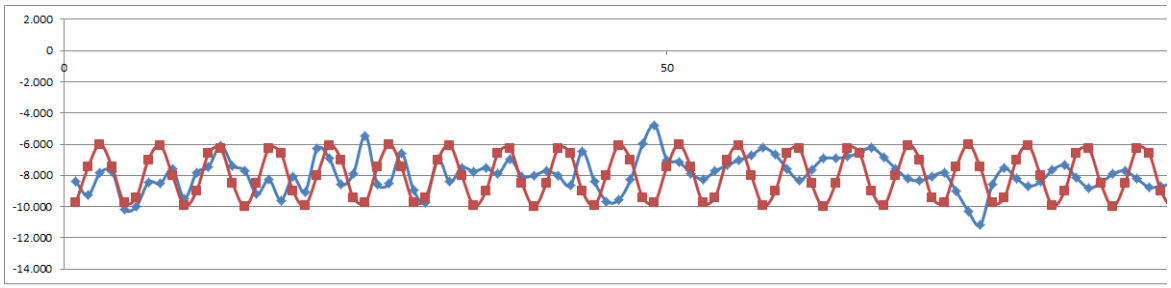

Figure 111: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 4 Part 1

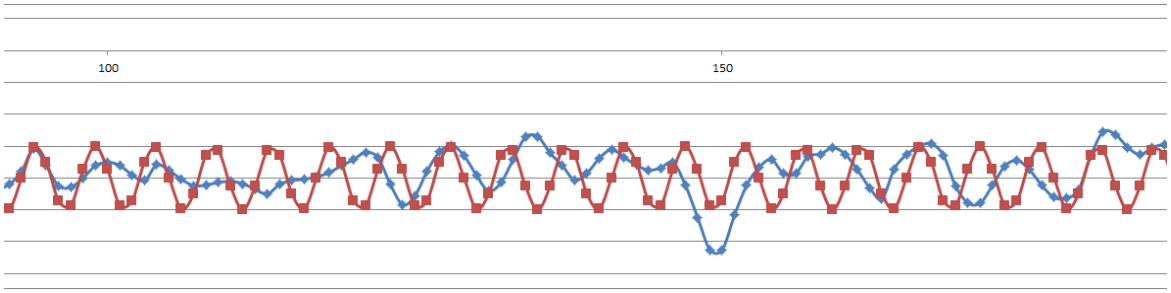

Figure 112: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 4 Part 2

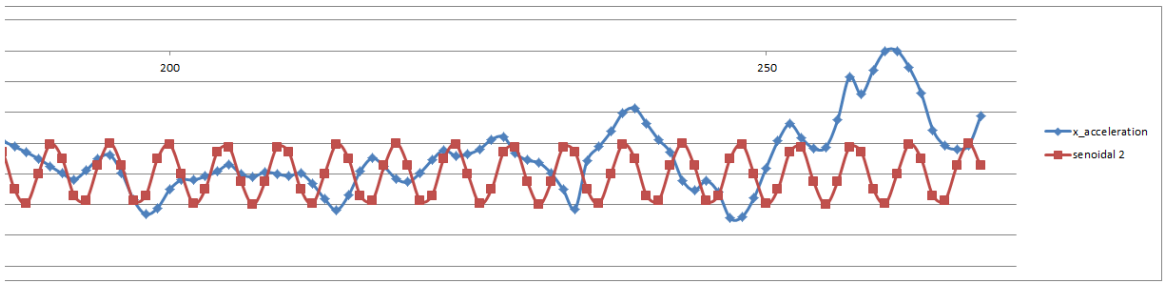

Figure 113: Acceleration Axis X vs Sinusoid 0.051 Hz Sample 4 Part 3

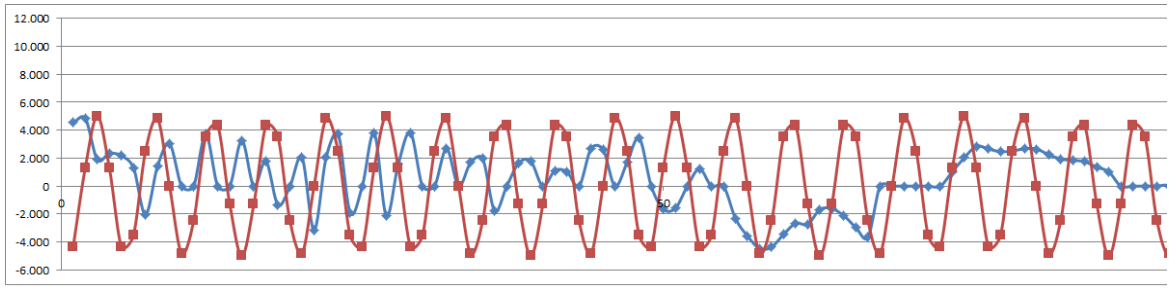

Figure 114: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 4 Part 1

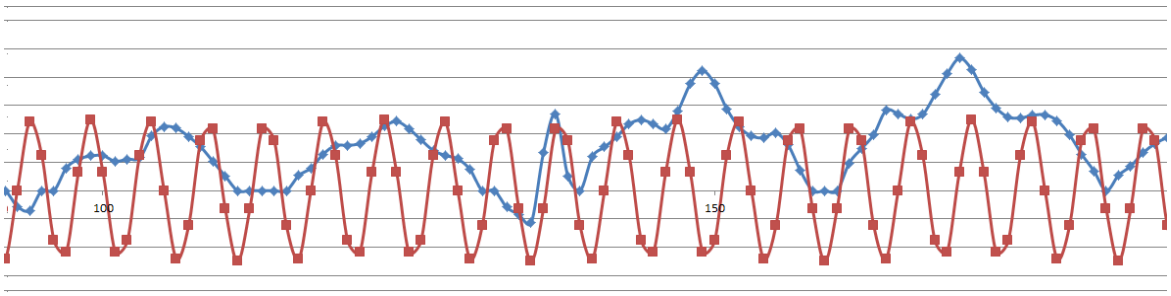

Figure 115: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 4 Part 2

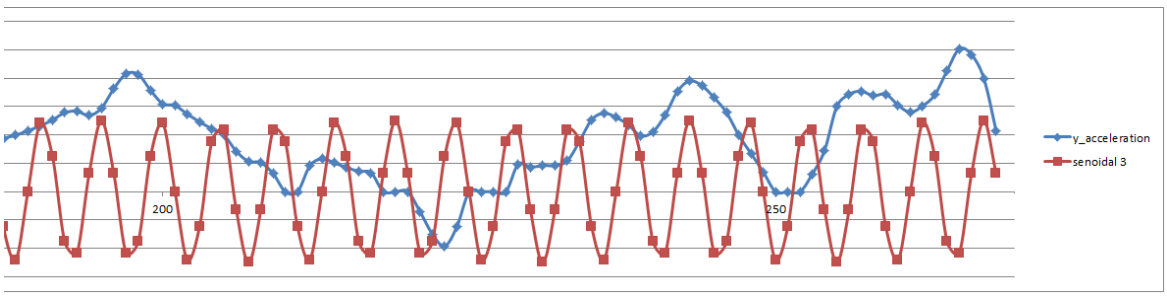

Figure 116: Acceleration Axis Y vs Sinusoid 0.051 Hz Sample 4 Part 3

Acceleration of the accelerometer on the axis Z, it is too aleatory.

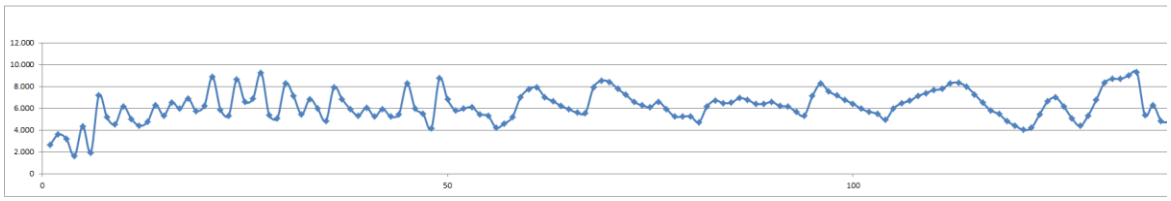

Figure 117: Acceleration Axis Z Sample 4 Part 1

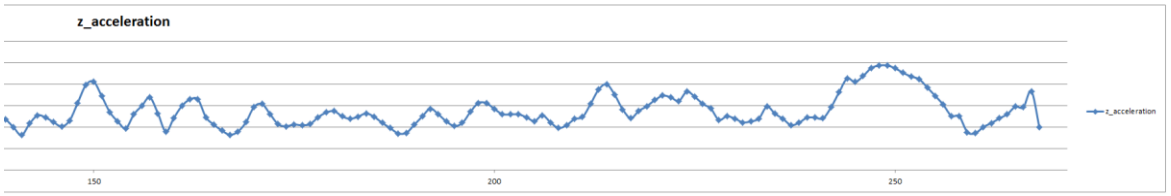

Figure 118: Acceleration Axis Z Sample 4 Part 2

5.2.5. Conclusions

The graphics are more random at the train than initially thought. We can see that the sine wave signal is adapted quite well to the signal in some parts but in other far enough. The shape of these signals is not like a sinusoidal, only on a few parts so in this case it would be more difficult to determine the shape of the signal from the accelerometer.

And again we see that the waveform received in the Z axis is very random, therefore we have not compared it with any signal.

5.3. Conclusions of the Analysis

This analysis is very important for future work because it will help to identify the type of transportation system being used by the user. Therefore, it will help to save battery because the use of the GPS signal will be reduced. This is really important especially in the subway where we will not have GPS signal.

By comparing the signal collected with sinusoids, we can distinguish when the user is using train or subway.

Type of transport	Frequency on axis X	Frequency on axis Y
Subway	0.189 Hz	0.063 Hz
Train	0.051 Hz	0.051 Hz

Table 3: Transport Samples vs Sinusoid

6. Conclusions and future development:

6.1. Conclusions

The two new modules incorporated give more value to the application for the user because they allow him to have two new tools to help him in his everyday life.

- The accident detector helps the user to quickly notify an accident, this gives added safety to the user because accidents happen when you least expect it.
- The pedometer can help the user to lead a healthier life, he can keep records of his steps, see how many kilometers he has walked and how many calories have been burned.

6.2. Future Work

These new modules can allow us to add many new features.

6.2.1. Crash Detector

- It would be interesting to send a message to the server when the application detects the sudden change caused by an impact and wait for a second confirmation message, either emergency or false alarm. Thus, if the device is damaged the server can detect that something has gone wrong.
- On the server, we could analyse the areas of the city with more frequency of accidents.
- Talk with other emergency services such as those in Alpify to work with them.
- If the user presses the emergency button, MobilitApp could give him instructions on first aid and depending on the vehicle used these instructions could vary, it would help to the user to stay calm and know what to do in an accident (metro, bus, car, etc.)

6.2.2. Step Counter

- Add targets or goals of daily steps, kilometers or calories and notify the user when they have been fulfilled or not fulfilled at the end of the day.
- Keeping track of the days that have taken more steps.
- Generate graphs of steps, distance and calories distributed by days or weeks.
- Save the records of steps and share them on social networks.
- Calculate the CO2 saved by walking instead of using motorized transport.

7. Annexes

7.1. Annex A - Setting a server up with Raspberry Pi

For starting up the server, I have used a paper that wrote Gerrard Marrugat, we could find it in his Final Degree Thesis [10].

The document details the steps and all configurations to be made. We had a problem with the domain to register the Mobilitapp web because the web used (http://www.noip.com/) recently changed and free domains available were changed. However, you could use the old version of the web and return to the domain that had earlier mind to preserve the web (http://mobilitapp.noip.me/).

7.2. Annex B – Web Page

MobilitApp has its own website, so adding new features this also has to be reflected in the web.

We have added a small description and some screenshots to see the functionality.

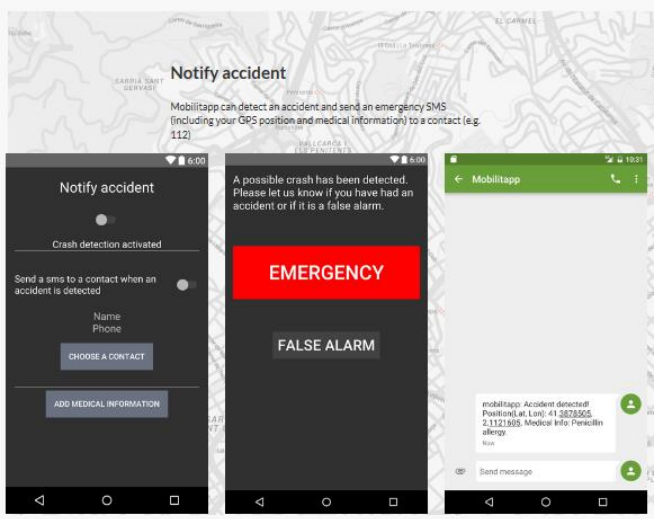

Figure 119: Notify accident Web Page

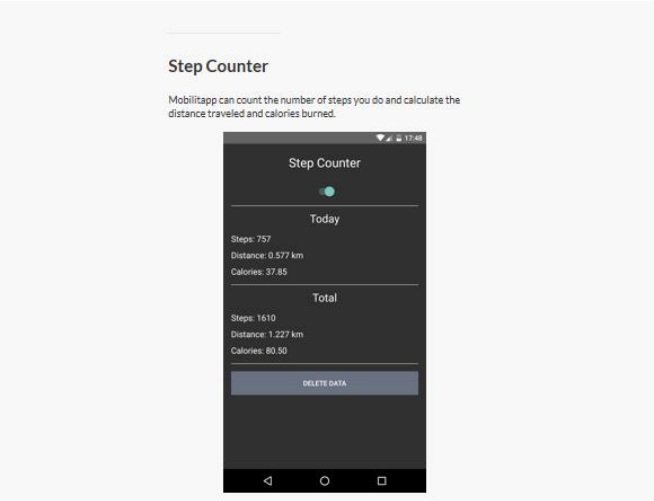

Figure 120: Step Counter Web Page

By incorporating the information of the new features on the website, it has followed the style that we had in order to have uniformity.

Bibliography:

[1] Marc Llahona, Sergi Casanova, Ángel Torres, Gerard Marrugat. MobilitApp, Android application to obtain mobility data of the citizens in the metropolitan area of Barcelona.
http://www-entel.upc.edu/monica.aguilar/simulators.html

[2] Android Developers. Motion Sensors.
https://developer.android.com/guide/topics/sensors/sensors_motion.html?hl=es

[3] Android Developers. Android Manifest.
https://developer.android.com/guide/topics/manifest/manifest-intro.html?hl=es

[4] SOLID principles.
https://es.wikipedia.org/wiki/SOLID

[5] SharedPreferences.
https://developer.android.com/reference/android/content/SharedPreferences.html

[6] Services.
https://developer.android.com/guide/components/services.html

[7] Activity
https://developer.android.com/reference/android/app/Activity.html

[8] Thread
https://developer.android.com/reference/java/lang/Thread.html

[9] Decimal degrees
https://en.wikipedia.org/wiki/Decimal_degrees

[10] Final Degree Thesis. "Improvement of algorithms to identify transportation modes for MobilitApp, an Android Application to anonymously track citizens in Barcelona". Authored by Gerard Marrugat Torregrosa. (July 2015).
http://arxiv.org/abs/1605.05342

[11] Android Studio IDE
https://developer.android.com/studio/index.html?hl=es-419

[12] Java
https://en.wikipedia.org/wiki/Java_(programming_language)

[13] Excel
https://en.wikipedia.org/wiki/Microsoft_Excel