ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Ανάπτυξη εφαρμογής ανίχνευσης σεισμού σε έξυπνα κινητά τηλέφωνα με λειτουργικό Android

Development of an Earthquake Detection Application in Smart Phones with Android OS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Του

Ίντ Απόστολου-Ουαλίντ

Βόλος, Ιούνιος 2012

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Ανάπτυξη εφαρμογής ανίχνευσης σεισμού σε έξυπνα κινητά τηλέφωνα με λειτουργικό Android

Development of an Earthquake Detection Application in Smart Phones with Android OS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ίντ Απόστολου-Ουαλίντ

Επιβλέποντες :

Τσομπανοπούλου Παναγιώτα
Επίκουρη Καθηγήτρια Π.Θ.

Μποζάνης Παναγιώτης
Αναπληρωτής Καθηγητής Π.Θ.

Εγκρίθηκε από την διμελή εξεταστική επιτροπή την ΗΜΕΡΟΜΗΝΙΑ ΕΞΕΤΑΣΗΣ

(Υπογραφή)

….....

Τσομπανοπούλου Παναγιώτα

Επίκουρη Καθηγήτρια Π.Θ.

(Υπογραφή)

.....

Μποζάνης Παναγιώτης

Αναπληρωτής Καθηγητής Π.Θ.

Βόλος, Ιούνιος 2012

(Υπογραφή)

.....

Ιντ Απόστολος-Ουαλίντ

Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την κα Παναγιώτα Τσομπανοπούλου επίκουρη καθηγήτρια για την υποστήριξη της και το χρόνο που αφιέρωσε στη διπλωματική αυτή εργασία.

Επίσης θα ήθελα να ευχαριστήσω τους καθηγητές μου κο Χρήστο Αντωνόπουλο αναπληρωτή καθηγητή και Ιωάννη Κατσαβουνίδη επίκουρο καθηγητή, που με μύησαν στον τρόπο σκέψης του μηχανικού.

Στην προσπάθεια αυτή μου στάθηκε η οικογένεια μου, οι φίλοι μου και η ομάδα μου, τους οποίους ευχαριστώ θερμά.

```
for(i=0; i<all; i++)
        printf("Thanks: %s\n",friends_family_array[i]);
```

Περίληψη

Η πρόωρη και ταχεία πρόβλεψη φυσικών καταστροφών όπως σεισμοί, και τσουνάμι είναι ένα απο τα ερευνητικά ζητήματα των τελευταίων ετών. Η εύκολη ενσωμάτωση αισθητήρων στα σπίτια και στα έξυπνα κινητά τηλέφωνα έφερε αυτή τη ιδέα ένα βήμα πιο κοντά στην υλοποίηση. Επειδή προειδοποιήσεις τέτοιου τύπου απασχολούν όλους σε μια κοινωνία, η συμμετοχή και υποστήριξη των κατοίκων είναι κρίσιμη και πολλές φορές επιβεβλημένη.

Σε αυτή την εργασία, παρουσιάζουμε μια εφαρμογή ανίχνευσης σεισμών, η οποία στηρίζεται στην ανίχνευση σεισμών τοπικά μέσω ενός έξυπνου κιτητού τηλεφώνου. Κάθε κινητό που διαθέτει την εφαρμογή είναι σε θέση να ανιχνέυσει ένα σεισμό, χώρις να είναι μέρος ενός δικτύου αισθητήρων.

Κάθε φορά που ένας σεισμός λαμβάνει χώρα η παράσταση της μέγιστης επιτάχυνσης του εδάφους έχει μια συγκεκριμένη μορφή που εξαρτάται και από το είδος του σεισμού. Γνωρίζοντας αυτές τις μορφές μπορούμε να αποφανθούμε αν ανά πάσα χρονική στιγμή γίνεται σεισμός ή οχι μέσω μιας σύγκρισης με τη τρέχουσα επιτάχυνση. Η παραπάνω ιδέα ήταν αυτή στην οποία βασιστήκαμε για να υλοποιήσουμε την εφαρμογή μας η οποία όταν αντιληφθεί ότι γίνεται σεισμός ενημερώνει τον χρήστη.

Βέβαια η καθημερινή χρήση του κινητού τηλεφώνου σε διάφορες καθημερινές δραστηριότητες όπως το περπάτημα, η οδήγηση, η γυμναστική, ή και κάθε άλλη συνηθισμένη καθημερινή δραστηριότητα, μπορεί να μας παράγει μορφές επιτάχυνσης που μοιάζουν αρκετά πολύ με σεισμικές μορφές, κάτι το οποίο θα μπορούσε να ξεκινήσει έναν λανθασμένο συναγερμό. Στην υλοποίηση της εφαρμογής μας προσπαθήσαμε να ελαχιστοποιήσουμε τα σενάρια λάθους συναγερμού, προσέχοντας όμως τα σενάρια έλλειψης συναγερμού σε περιπτώσεις σεισμών, είναι ελάχιστα αν όχι ανύπαρκτα μιας και τα αυτά είναι ο λόγος ύπαρξης της εφαρμογής.

Η υλοποίηση της εφαρμογής ανίχνευσης σεισμών στηρίχτηκε στη ιδέα των λεγόμενων Συνόλων Εκπαίδευσης και αποφασίζει για την ύπαρξη σεισμικών δονήσεων τοπικά, χωρίς την ύπαρξη διακομιστών ή και δικτύωσης με άλλες συσκευές. Επιπλέον, παρουσιάζουμε αλγορίθμους διαχωρισμού παραστάσεων κανονικής χρήσης, με αυτές των σεισμικών παραστάσεων.

Τα αποτελέσματα των πειραμάτων που επιχειρήσαμε σε σενάρια πραγματικού χρόνου, ήταν ικανοποιητικά σε μεγάλο βαθμό, αφού η εφαρμογή ήταν σε θέση να απαντήσει σωστά σε καθημερινά σενάρια χρήσης, και σε μοντέλα κινούμενου τραπεζιού.

Abstract

Early and rapid detection of natural catastrophes caused by earthquakes, tsunamis, fires, and such events, is a topic under study for many years. New abilities of equipping typical citizens with common sensors, e.g., attached on smart phones, made this life saving issue closer to application. Since these alerts involve everyone in a community, their contribution would be critical and on most cases guaranteed.

In this thesis, we provide an implementation of an earthquake detection application, based on an individual sensor, placed an Android smart phone. The implementation can be extended to be a part of a community-based sensor network.

When abnormal seismic event is presented, the PGA (Peak Ground Acceleration) values generates a specific pattern. Therefore using an Accelerometer sensor attached to each Android smart phone, we can calculate the acceleration pattern and decide to alert when earthquake patterns are detected. This can be critical on conditions where the user is not aware of the danger on the moment, and can be a life saving alert.

On the other hand, the daily use of the device on different activities, like sports, walking, driving, or any other daily activity, can create an earthquake pattern, which is a false-positive alert. We tried to eliminate those false-positive alerts by building a daily training data, while keeping the true-negative (undetected earthquakes) scenarios rare if not impossible.

We present an implementation of an application that uses training sets model, to decide locally –without using any server or networking- if an earthquake is happening or not. We also present algorithms to achieve this differentiation between normal daily use of the common sensors patterns, and the seismic abnormal activities patterns.

Finally, we experimented this application on real life scenarios, where we had good detection results, on both daily and shaking table experiments, using the basic training sets we created.

Contents

1 Introduction

1.1 Earthquake Detector Application (ED)

The Earthquake detector, is an application that was developed to detect real-time earthquakes activities, and alert the user, to take safety steps and decrease the effects of such an event.

In this thesis, we present the implementation of an application (ED) that uses training sets and user controllers (that eventually will work with each other) to keep the users prepared and ready for any kind of earthquakes. In future the application can cooperate with others to make a community system for early earthquake alerts.

This application is a light-weight Android application, optimized on both memory and CPU performance, with minimum SDK of 2.1 (level 7 SDK), and can be easily installed and used on any Android smart phone. It also provide a user UI, and capabilities such as exchanging training sets, or download any uploaded training sets of other more experienced users from all over the world.

Our Main target is to alert users for the danger of an earthquake that has already started, so that they can be able to take any important measurements, and save their lives, especially users that are not aware of the danger due to unconsciousness, sleeping, or even old people with awareness problems.

1.2 Thesis Structure

In the 2nd Chapter related works are presented and compared to Earthquake Detector. Chapter 3 discusses about the required theoretical background. In Chapter 4 the system design and architecture is described as well as a high level class explanation. Chapter 5 discusses the implementation details and presents the platforms, the software development, as well as the file formats that are used. In Chapter 6 a manual of Earthquake Detector is presented. Finally, Chapter 7 presents our conclusions and directions for future work, while Chapter 8 contains the bibliography.

2 Related Works

In this chapter we will quickly present some related works, when searching for earthquake detection or earthquake physics, Internet provide a lot of searching results, this means that there is a great work and some serious studies for prediction the next big earthquake, or a geological instability, most of the theoretical side studies, especially on earthquake physics side, suggests that there isn't a specific algorithm to predict such events with sureness, on the other hand, computer science suggests a series of algorithms that can achieve the target using different methods, each with its own cost.

We will discuss two papers [1] and [2]in the next sub-chapters.

2.1 The Rapid Detection of Rare Geospatial Events: Earthquake Warning applications [2]

Michael Olson, Annie Liu, and Matthew Faulkner, presented on their paper algorithms and simulations of detecting rare geospatial events. They applied their theory to an application that warns for rare events such as earthquakes. After analyzing streams of data they collected of a lot of sensors, on the form of mobile and tablets sensors, or even special-purpose stationary sensors.

They also presented algorithms to detect such events on a cloud computing servers, by exploiting the scalability of cloud computers while working within the limits of state synchronizations across different servers in the cloud.

They also analyzed the impact of sensors and population distribution on such applications, by assuming that the population and sensor distributions are similar.

2.2 The next big one: Detecting Earthquakes and other Rare events from community-based Sensors[1]

Matthew Faulkner, Michael Olson, Rishi Chandy, and Jonathan Krause, discussed in their paper the possibility of predicting the next big earthquake, with an early warning of it, they also discussed the difficulties of modeling and characterizing rare events such as earthquakes, and how those problems can be solved.

They present a principled approach towards detecting rare events that learns sensor-specific decision thresholds online, in a distributed way. That maximizes anomaly detection performance at a fusion center, under constraints on the false alarm rate and number of messages per sensor.

They also present their community-based sensor network CSN, a community sensing system with the goal of rapidly detecting earthquakes using cell phone accelerometers, consumer USB devices and cloud computing based sensor fusion.

Finally an Android client implementation for community-based sensor network, with early warning application is also discussed, this implementation had been studied very carefully and we used some important points on our implementation.

3 *Theoretical Background* [6]

Earthquake engineers and scientists, believe that early earthquake detection is a promising and interesting topic for research. Serious studies on the earthquake prediction had been completed, implementing algorithms that will predict with no big sureness when the next event will occur and where.

The reason this early prediction is a very complicated and hard issue, is the physics of the earthquake itself. Each earthquake is very different for other earthquakes. These differences are based on the depth of the earthquake epicenter, on the way each earthquake occurs, on the geological points around the center of the earthquake, and on the amplitude of the earthquake. These very important parameters produce very different earthquake patterns, that may be close to each other or totally different, these patterns can also look very similar to some non-earthquake earth activities, (e.g. Volcano activities). This makes earthquake prediction even harder and more complicated.

Computer scientists came up with some really interesting ideas of early warning applications, especially when citizens of common communities are equipped with sensors that can collect very important information. These community-based sensor networks can early notify for an upcoming event. The main idea is when some sensors detect a seismic abnormality, they send these information to a cloud server. Which then notifies the citizens of near geographical areas for the possibility of this upcoming seismic event. So the difference between the early warning and the prediction of an earthquake, is that the first one is a warning of an already existing earthquake that just began to some parts of the community. The earthquake prediction foresees an event before it even starts, usually long time prior to its appearance.

For the success of the community sensor network, sensors built on smart phones (e.g. the accelerometer on the Android phones) must be able to detect and notify the server for when events begging. These smart phones have to be trained to recognize the current acceleration patterns, with some certainty before notifying the server for any event. This is not an easy task, since daily movements produce an acceleration patterns, that can be similar to seismic patterns.

The target of our android earthquake detector, is to minimize the negative-true, and the positive-false scenarios. A negative-true scenarios is when an earthquake is happening but the application can't detect it, while a positive-false one is when no earthquake happening, but the application detect some abnormality. We used the AI (Artificial Intelligence) training sets method, where the application can learn by a real scenarios, and keep these information for future reference. We took advantage of the user interaction to stabilize the application behavior when this is needed.

We made some primary experiments measuring the acceleration of the device while placed on a shaking table, and on a daily use with the smart phone in a back-pocket. The results were good enough, where the application can provide better behavior and decision accuracy, with more earthquake real PGA (Peak Ground Acceleration) values, or a real earthquake experiences during its training period.

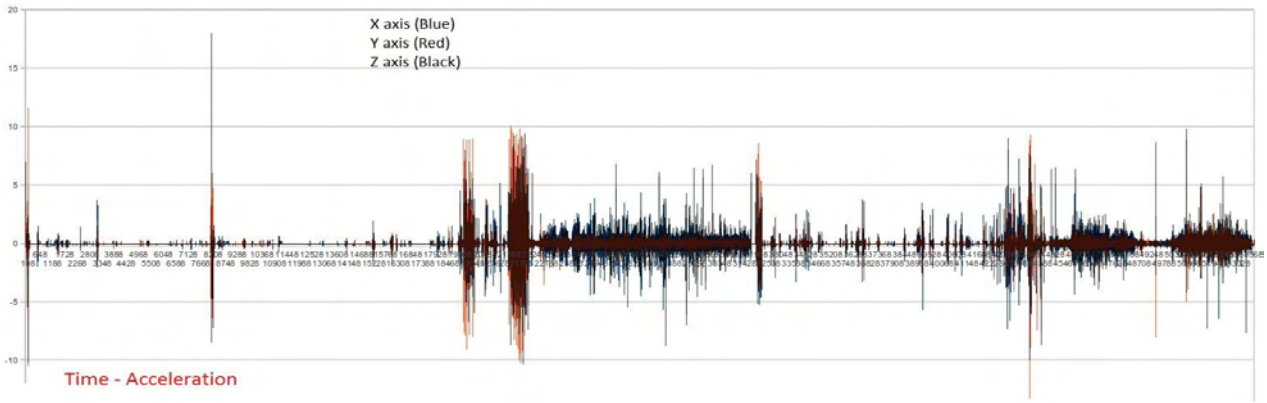


Figure 1: Graphical pattern of acceleration of one hour of daily use with the smart phone in the back-pocket.

4 System Design

This chapter analyzes the system's design and software architecture.

4.1 Architecture

In this chapter, we will analyze the most important points that draw the earthquake detector architecture.

4.1.1 Events and Knowledge base

Developing a software that can detect earthquake activities, is a non-trivial matter, especially when using cheap sensors on a smart phone. Having no high level data mining algorithms and tools, or cloud computers as servers, the software have to be as simple as possible, but complicated enough so that any smart phone can **decide** under any circumstances, based only on the knowledge of the current acceleration values, and/or any previous local saved knowledge.

To accomplish our target, we used a simple idea of making the application understand the current motion pattern, using mathematical constants of the current motion vectors, such as variance, mean value, and Fourier series coefficients. Those patterns are compared to a local knowledge base, and the application decides if the particular motion is closer to be a normal motion, or a seismic activity.

As we mentioned, this application is a local host earthquake detector, which means that there is no central server, or any kind of network, and decisions are token by the same smart phone that measured the current motion pattern. This makes some

knowledge bases untrusted. For example, our built-in knowledge base contains motion patterns while walking, running, driving, getting up and down on stairs, and some simulated earthquake activities. Therefore, if some user decided to dance while running the application, it is very possible to recreate a pattern of a "known" seismic activity. To eliminate such problems, users are responsible of training their own application on special movements. This training will save new normal user activities like dancing, on its own knowledge base, so that the next time the user dances using his smart phone, there will be no problem at all.

We will refer to the knowledge base as the **training set** in the rest of this thesis. The training set is a set of data or events that are saved in a specific pattern, and is used to help the application decide and produce results.

4.1.2 *Filtering the Gravity Acceleration*

Physicists assume that due to earth's gravity, there is always a force **g** (g=9.8 m/s^2), effecting any object's acceleration, even if this object is stable.

When the accelerometer perform a measurement, the gravity acceleration constant will appear on one of the 3 axis (X,Y, or Z), based on the devices orientation on the moment. This problem causes the application not to have a stable point of reference. To create this point, all accelerometer measurements have to pass through a high-pass filter, which will cut off any gravity acceleration effects, and provide the application with the stable point of reference it needed.

Figure 2 and 3 shows how the gravity filter changes the graphical chart of the acceleration values when the device is placed on a stable surface (table). All 3 axis have a mean value of almost zero when sampling the acceleration values on stable table, when the Z device axis is vertical on the table.

4.1.3 *Decision algorithm*

Deciding whether an event is a seismic abnormality or not, is based on the applications knowledge bases. When reading an event, the application compare the particular events pattern with these in the knowledge bases. It will results in two closest known patterns, one seismic pattern, and normal one. Next, the application

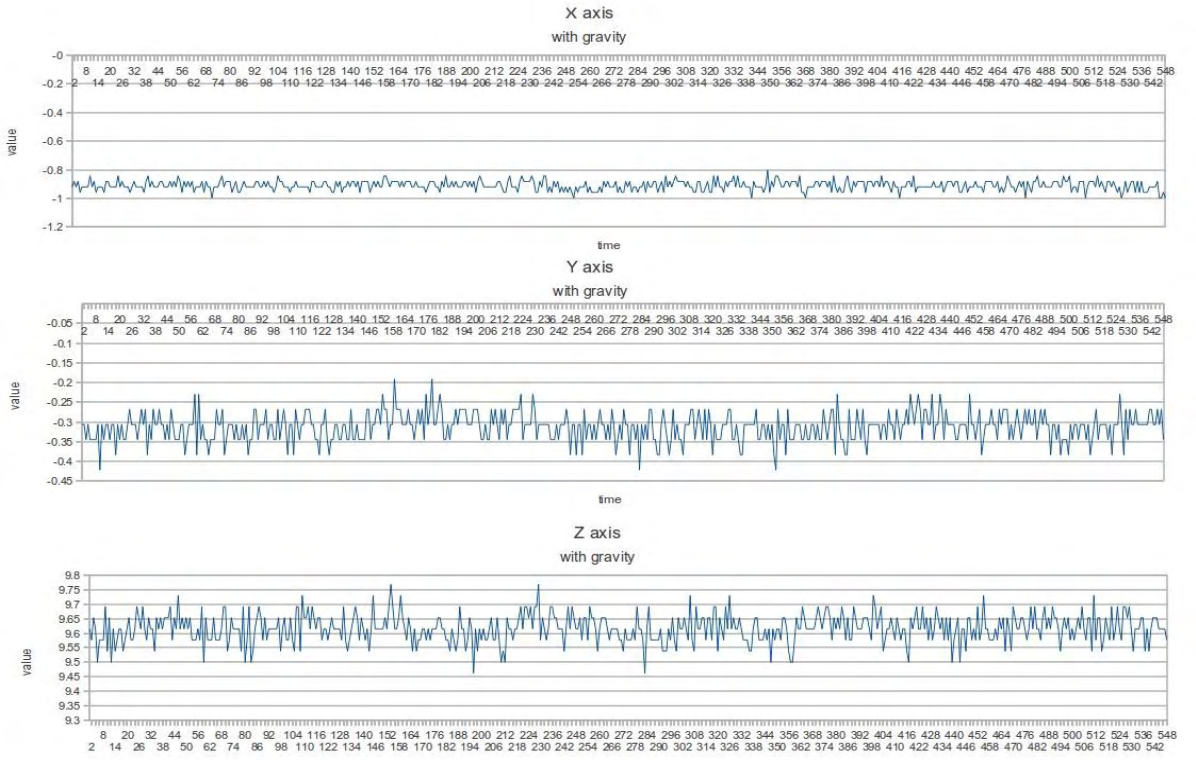


Figure 2 Acceleration values before filtering the gravity values.

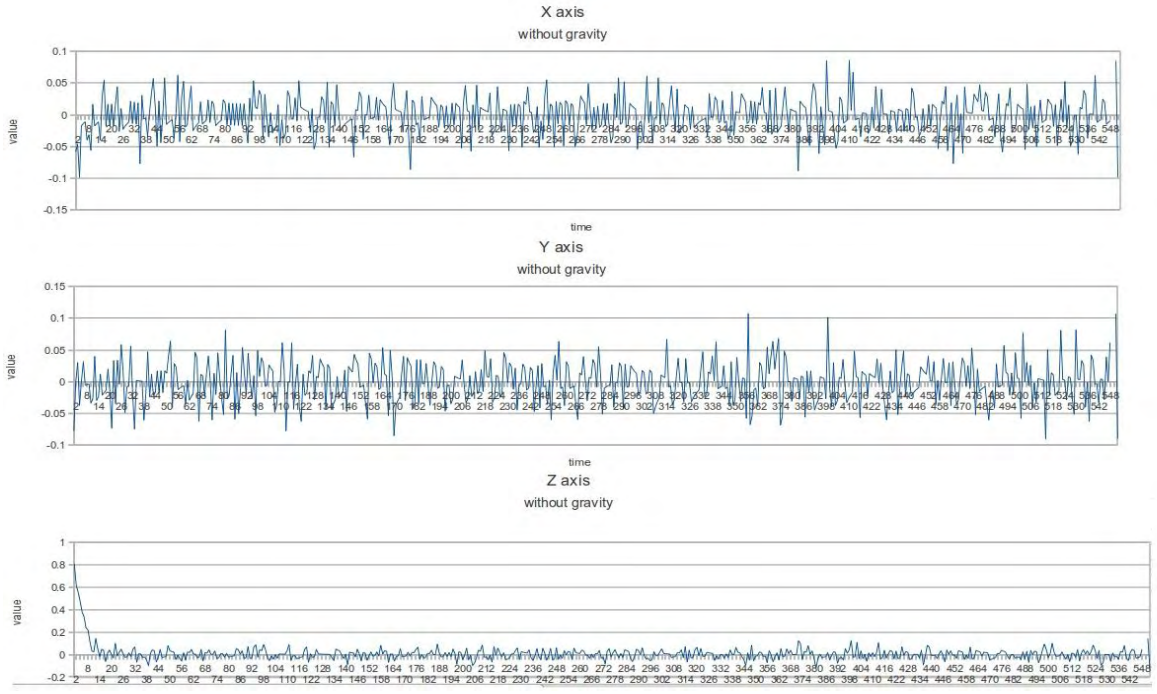


Figure 3: Acceleration values after gravity correction.

tries to detect more pattern similarity, and eventually decide whether it's a seismic pattern or not.

This algorithm do not guarantee a 100% correct decision. Based on the current knowledge base, and the quality of the training, this percentage can grow. Also, earthquakes have very different patterns with each other, and some of those patterns are very similar to some daily activities, where even human being with closed eyes, cannot be sure if they are experiencing an earthquake or not, e.g. bulldozers motion working is really close to an earthquake motion.

4.2 Description of Classes

This application's implementation had four major classes described as follows:

- *EarthquakeActivity class:* [3]

This is the main class of the project, which controls the flow of the application, as well as handling the UI (User Interface) functionalities. It also receives the data calculated by the Accelerometer, starts the alarm wherever it's necessary, and finally keeps other classes parameters on track and updated.

Controlling the UI contains all listeners creating, such as the browse button, the sensitivity seek-bar, the radio-button group, and the accelerometer live values monitor, any user interactions are received by this class, and passed to other classes if needed.

Finally, when a specific number of Acceleration samples are received from the sensor, this class will pass those values through a high-pass filter, before passing those values to both EarthquakeMath and Detector classes.

- *EarthquakeMath class:*

This class contains and implements all the mathematical functions that the application needs, those functions are, compute the Fourier series coefficients (A_n, B_n), calculation of the Euclidean norm <<X,Y>>, compute a variance of an array of values, and finally getting the maximum absolute value of an array.

Calling those functions with a specific order, will eventually create a new array of 36 values (32 Fourier coefficients, 2 variance values, and 2 maximum absolute values), which will be used later by the Detector class, these values express the "event" characteristics. It is important that those values are saved by a specific format, this format will be explained later in section 5.1.

The idea of these functionalities over a samples array, is discussed in [2]. We doubled the length of the event vectors, by removing the linear projection they use on their paper. We believe our event vectors are more accurate, but we lose a little performance for processing such big vectors. This trade off we can accept, since the application is running smoothly and without having any performance issues.

- *Detector class:*

Detector class is the core of this application. This class is responsible to decide if the event vector produced by the EarthquakeMath class, is a seismic event or not -when the application is on detection mode- and to save the event vector in the proper training set when the application is on one of the training modes.

Deciding a seismic abnormality is a hard concept, we use the current training sets situation, this class calculates the correlation of the event vector, with the closest saved event in both training set (Normal and earthquake sets), if the correlation value of the normal event is less than the correlation of the earthquake event plus a threshold (that is determined by the sensitivity seek-bar "User defined"), this will mean that the current event is familiar and set to be a normal activity caused by the user, so the decision will be no seismic event, otherwise the class will decide a seismic activity, due to lake of knowledge of such an event in the training set, and finally notifies the main class "EarthquakeActivity class", to start the alert.

- *FileDialog class:* [5]

This is an independent class, which starts whenever the browse button is clicked. It's main job is to open a file dialog window, where the user can select or create a training set file. This file is a normal training set file (not seismic values), and returns the full path of the file selected to the main class, where this file will be handled.

- *libdata.so*

Static library libdata.so, is a library that provides a fast file functions. Those functions are the load first line of a specific training set file function, load a whole training set file function, and finally save a training set file. Those functions are important to be developed using the Java Native Interface "JNI" due to great performance difference on file I/O functionalities between JAVA, and C. Appendix A contains more information about JNI and NDK developing.

Some other classes are implemented, such as the WorkingMode class, and the SelectionMode class. Both classes keeps the constants of the application. Please refer to 5.1 for more information about those constants.

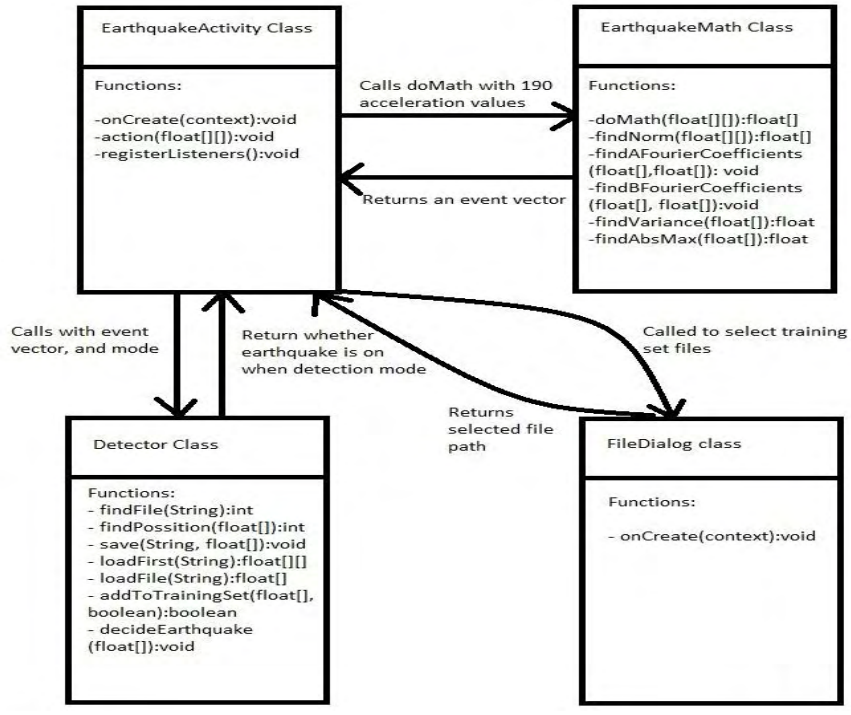


Figure 4 General data-flow of the four main classes.

5 Implementation

5.1 Implementation details

In this chapter, we will start explaining some basic algorithms and ideas we use on this project. Next we will discuss some implementation details of each class we described in the previous chapter, and finally, we will discuss the classes' interaction and data-flow model we used. Appendix B contains the real code of the application.

5.1.1 *Training Vectors / Events Vectors*

Training vectors have 36 float elements, and are saved in the training set files. These vectors describe some known motion patterns, that are used to compare with any current motion pattern (Event vector) and decide whether the latest is a seismic or a normal event.

We use the term "event vector" to refer to the current run-time motion pattern. After reading and processing the sensor (Accelerometer) values, these event vectors can either be used to detect a current earthquake activity, or be saved on a training set, and be used later as training vectors.

5.1.2 *File Format*

From now on, "files" mean the training sets files, either a normal training set, or an earthquake training sets. These files are always saved and handled with a specific,

stable and unchanged format, while the files are sorted, and chunked to 100 files of the same size (except for custom user training sets).

All of these files start with 2 integers, the first one expresses the total number of training vectors saved in the specific file, the second one is the width that these vectors use (normally 36 values/floats wide). The next line of these files contains the first training vector. This first vector of each file is always loaded to RAM, for faster selection of files where we have to read/write.

The file continues with the other training vectors, wrote each on a new line (each VECTOR_SIZE floats on a new line).

Finally, all file operations including saving/loading full files, or loading the first training vector are called using the libdata.so static library. Other parts of the application are not allowed to perform any such operation.

5.1.3 EarthquakeActivity class

Main parameters:

int **mode**: this parameter decides if the application is on an earthquake detection mode, normal training mode, or earthquake training mode.

float **samples**[190][3]: this float array saves the accelerometer sensor values as received.

Description:

As we mentioned previously, this is the main class of this project. This live loop class, starts with initializing the project parameters and UI listeners methods, activates the accelerometer sensor, and registers the sensor event listener. When the sensor count a 190 new acceleration values (event), it calls the action method, where this event will use the Earthquake math class, to get a new 36 values vector. Next the application decides to detect if this event is an earthquake, or save this event as a normal or earthquake training event, based on the mode parameter.

This list describes the UI listeners that are started on the application initialization phase:

- Seek bar onProgressChanged, this listener changes the detecting threshold, based on the seek bar progress current value, which can have values between 0-20.
- Radio group button onCheckedChange, this listener is responsible of changing the mode parameter, this parameter can have values of detection mode, normal training mode, and earthquake training mode, and used to decide what the application have to do with a new event.
- Button onClick, this listener will start a new file dialog object, where user can select the current training set to use as a normal event values.

Like the UI listeners, an accelerometer sensor listener is registered on the application initialization phase, this listener will activate each time the sensor changes it current acceleration values. When this happens, those values are passed through a high pass filter. This filter will calculate the gravity acceleration on each axis, and subtract this value from the real event values. The gravity value calculation on each axis uses this formula:

$$Gravity_x(t) = \left(a * Gravity_x(t-1)\right) + \left(\left[1-a\right] * Event_x(t)\right)$$

$$a = \frac{t}{t+dt}$$

where $a \sim 0.8$, and $Event_x(t)$ is the measurement of the current event on the x axis of the device, $Gravity_x(t)$ and $Gravity_x(t-1)$ are the gravity values on the same axis at current and previous time point.

The event values that are greater than 0.141747 will eventually pass the filter, this value is the absolute maximum value of the accelerometer's sensor noise (gravity acceleration and other noises like the sensors error), so that those values are pure and has no noise added to them. When the high pass filter is over, those values are saved on the samples[][] array, and printed on the application window.

Cutting values with values less than 0.141747 is totally safe, earthquake engineers suggests that PGA less than ~ 0.15g = $1.5\,m/s^2$, are unnoticeable by people. Therefore, we can cut those values without worrying of not firing an alarm for such an events.

Finally, when 190 values are saved on the array, the action method is called. This method sends this array to the earthquake math object, and receives a 36 new event vector values. Next it uses the mode parameter to decide which detector class method to call. If detection mode is checked, then the action method will call the DecideEarthquake method, and will fire an alarm if this method return true. This alarm will stop on the next false return of the DecideEarthquake method. If detection mode is not checked, this event vector will be saved on the normal training set files, or on the earthquake training set files depending on the current training mode checked.

5.1.4 EarthquakeMath Class

Input:

float **samples[190][3]**: this input array, which represents the sensor values as received and passed through the high pass filter.

Output:

float **output[36]**: an array of 36 floats which are the event values after processing.

Description:

This class is called using EarthquakeMath public method doMath, where a series of math methods are then performed, those math methods will eventually produce a 36 floats vector with a specific format.

The doMath method starts by finding the X,Y Euclidean norm, the acceleration values of the X axis are an 190 values vector, which are saved on samples[i][0], i=0,1,2, …, 189. The acceleration values of the Y axis are also an 190 values vector, and those values are saved on samples[i][1], i=0,1,2, …, 189. The Euclidean norm of <<X,Y>> is:

$$\| X,Y \|_i = \sqrt{X_i^{\,2} + Y_i^{\,2}}\ ; \qquad \text{for i=0,1,2, …, 189}$$

This method “normXY” will return a new vector of 190 values, Next, the doMath method will call the Fourier series method, which will calculate the first 8 coefficients “harmonics” (A_n, B_n) values of the Fourier series, on the <<X,Y>> vector. Those

values will be saved on the output vector on positions 0-15. We remind that the Fourier series coefficients are: [4]

$$A_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos nx.dx \quad , \; n \geq 1$$

$$B_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin nx.dx \; , \; n \geq 0$$

$$\text{and, } \; A_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} xdx$$

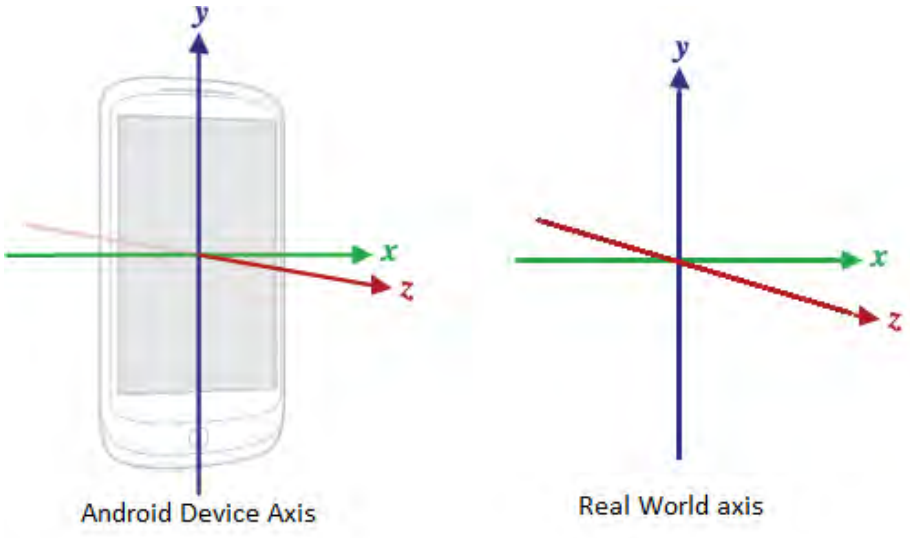
Next, the doMath will call the find variance method; a float will be returned and saved on position 16 of the output vector. The variance is calculated using this formula:

$$Variance = \frac{1}{\mu} * \sqrt{\sum << \mathrm{X,Y} >>_i - \mu} \, ,$$

for i=0,1,…,189 and μ = mean value of <<X,Y>> vector.

Finally, maximum absolute value of the <<X,Y>> vector is found, and saved on position 17 of the output vector.

Fourier series, variance and absolute maximum value of the acceleration values of the Z axis, which are saved on the samples[i][2], i=0,1,2,…,189, are calculated. The 16 harmonic values of the Fourier series of the Z vector, are saved on the output vector on positions 18-33, then variance of the Z vector is saved on position 34, finally the maximum absolute value of Z is saved on position 35.



The reason we picked to use the Euclidean norm of the X,Y axis, and the Z independently, is that a device that is on the detection mode will be facing up (Z axis is vertical on the device) on most cases, which gives the Z acceleration values more importance over the X and Y axis. This means that the application has to be accurate when handling the Z axis acceleration values. Finally, having a smaller output vector is recommended, due to limitation of searching time on the training set, as will be better explained on the detector class implementation details.

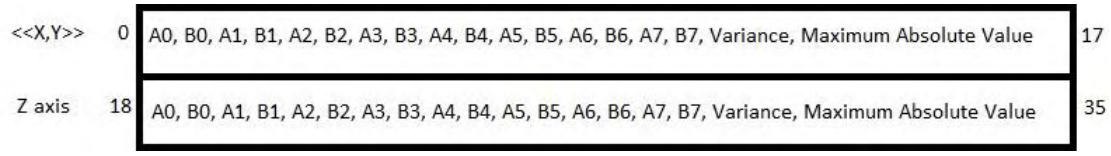


Figure 5 The training/event vectors alignment as saved in the training set

This mathematical module used to process the acceleration samples, to produce such event vectors, is discussed on [2] under the Android application they discuss.

5.1.5 Detector Class

Main parameters:

float[100][36] traininSetBases: this parameter keeps the first vector of each training set file.

int[100] trainingSetLengths: this parameter keeps the length of each training set file, depending on its number.

float[] trainingSet: this vector is used to load a file into, the size of the vector depends on the number of vectors in the file to load.

Description:

The detector class is the core of the application, it decides whether an event is actually a seismic abnormality or not, or saves the current event vector in the training set when training mode is enabled.

For minimal memory consuming and best possible performance, a detector object, will load the first vector of the 100 training set files, and keeps these values on the trainingSetBases, also it keeps the length of these files on the trainingSetLengths, this

will minimize the memory use by a factor of almost 100, and due to the sorted training set files, a two level training set tree scheme is created, and this will minimize the searching time also.

When a detector class method is called, such as the decision method, or the save method, the algorithm always starts by selecting the correct training set file, this search starts by reading the sorted trainingSetBases, and stops when a "bigger" vector is found, when this happens, the previous file will be loaded.

This will be better explained in the next example. Let's assume that we have a trainingSetBases of 4 positions, each of 4 integer values, and that those values are:

trainingSetBases[0] = "0, 4, -2, 5"

trainingSetBases[1] = "4, 4, 0, 6"

trainingSetBases[2] = "4, 5, 2, 8"

trainingSetBases[3] = "6, 3, 9,-1"

Those values are sorted properly, the sorting algorithm always checks the first value first, and sort based on this value, if the first value of both vectors (training set vector, event vector) are equal, the second values are tested. If the values on second position are equal, then the third value is examined, etc …

So, if we assume that those trainingSetBases values are the first vectors of 4 training set files, and that the current event vector we are searching is:

EventVector = "5, 3, 2, 1"

Then the algorithm will run and pass the first position of trainingSetBases. Since 0 is less than 5, also it will pass the second and third positions because 4 is less than 5, the search will stop when examining the fourth position of the trainingSetBases, because 6 is greater than 5, and the file that will be loaded is the file that contains all the values between:

trainingSetBases[2] = "4, 5, 2, 8"

trainingSetBases[3] = "6, 3, 9,-1"

This is the file number 2, or the third file.

On this phase, the trainingSet will contain the file vectors, the search will restart, but this time it will use the trainingSet values, not the trainingSetBases. Using the same

exact algorithm, the detector class will find the most similar vectors to the current event vector. Next, a correlation value is calculated to the closest two vectors (the first smaller, and the first greater vectors). This will produce two correlation values, and the algorithm will pick the smallest one, the correlation between two vectors is calculated using this formula:

$$Corr(X,Y) = \sqrt{\sum_i X_i^{\,2} - Y_i^{\,2}} \quad , \quad \text{for i=0,1,2, …, 36.}$$

The same thing will happen to find the closest earthquake vector from the earthquake training set, and a final correlation value is produced.

To make the final decision, the algorithm assumes that if the normal correlation value (correlation to the closest normal training set vector), is smaller than 3 times the earthquake correlation value, the decision will be False, otherwise it will be True, where the false value represents a normal activity, and the true value representing an earthquake activity.

Finally, if the application is running in training mode, then the detector class will only locate the exact position of the current event vector, in the proper training set depending on the training mode (earthquake or normal training), and add it in the proper file.

Please notice that when a file is already loaded in the RAM, there is no need to reload the same data if the next event happens to be in this same file. Loading a new file, will only happen when the needed information are saved on another file (different from the already loaded one). This improved the application performance, because most of the times, the event vectors are very similar to each other when sampling time is near, which means that those event vectors are probably in the same training set file.

5.1.6 Libdata.so

Description:

Libdata.so, is a fast file reading and saving library, that is implemented to help performance while training set files loading or saving.

The problem with the JAVA file input/output streams, is that they are extremely slow in compare with the native C code, which made the file loading by the detector also slow; this meant a non-effective earthquake detecting.

This library contains 3 functions, described as follows:

– int loadFirst(float[] output, String path);

This function reads the first training vector from a file located at “path”, and saves them on the JAVA VM memory with address “output”, and returns the number of vectors that are saved in this file.

– void loadFile(float[] output, int size, String path)

This function will read “size” training vectors saved in file located at “path”, and write them on the JAVA VM memory with address “output”.

– void saveFile(float[] input, int size, String path)

This function will save “size” training vectors wrote on the JAVA VM memory with address “input”, and write them in a file located at “path”.

Those 3 functions are implemented using the C programming language, and built using the Android-NDK, in combination with the GCC, as a static library, and communicate with the JAVA Android code, using the JNI (Java Native Interface), more about JNI and NDK on Appendix A.

5.1.7 *Constant classes*

Two classes are used to keep constant values, the first on is the WorkingMode class, which contains those constants, and used by Detector, EarthquakeMath, and EarthquakeActivity classes:

- int DETECTION, this is a static value that refers to the mode with value 0, that means the application is on a detection mode.
- int NORMAL_TRAINING, this is a static value that refers to the mode with value 1, this means the application is on a normal training mode.
- int EARTHQUAKE_TRAINING, this is a static value that refers to the mode with value 2, this means the application is on an earthquake training mode.

- int SAMPLES_SIZE, this static value refers to the width of the event/training vectors that is set to 36.
- int WINDOW_SIZE, this static value refers to the number of acceleration values needed before calling the earthquake math, and perform detection or saving operation.
- int HARMONICS, this static value refers to the number of Fourier coefficients to collect, this value is set to 8, which means that the Fourier series function will find the $A_0 - A_7$,and $B_0 - B_7$.

The other class is the SelectionMode class, which contains two static values used by the FileDialog class, those values are:

- int MODE_CREATE, this static value with value of 0, refers to the new file creation mode needed by the file dialog.
- int MODE_OPEN, this static value with value of 1, refers to the open file mode while selecting a file.

5.1.8 Other helping tools (non-Android)

Fourier application, this application is a C script, which reads a file with accelerometer values saved, and uses those values to produce as many training vectors as possible, those training vectors uses the same format as the vectors on the Earthquake math class.

Sort_trainingset application, this application is also a C script, that sorts vector files of the previous format, sorting the training set is important to help performance on the android device, and is based on the first value first (checking greater values from right to left, or position 0 to 36).

Erase_multiples application, this application if there are vectors of the same 36 values on a training set file, and erases any vectors with this attribute, keeping one of those multiple vectors.

Separator application, to help performance on the memory side, this script reads a training set file with the known format, and cuts it to 100 files of the same size; those files are also sorted and do not include any repeated vector values.

5.2 ***Platforms and Development Software***

This chapter analyses platforms and development software used by this application.

5.2.1 *Platforms*

Earthquake detector application was hosted on Android smart mobile phone, those smart phones uses ARM microprocessor, and Android middle-ware and operating system.

To debug and test this project, we used Samsung galaxy "Spica" GT-I5700, with:

- Android OS v2.1 (Eclair) update 1
- ARM11 processor, 800MHz, 32KB (16K data cache, 16K instruction cache) L1 level cache, 32bit machine word width, and ARMv6 instruction set.
- 256 MB of SDRAM.
- Accelerometer sensor.

Android smart phones with equal or higher system specification are valid platforms, and can run this application with no troubles.

5.2.2 *Development Software*

We used the eclipse Indigo v3.7.1, with the ADT (Android Development Tools) for eclipse Plugin installed, to write and run the projects code, using JAVA compiler with compliance level 1.5.

Also, we used the Android NDK-R8 to compile and build static library that this project used.

Finally, the GNU Compiler Collection (gcc) version 4.6.3, is used to compile the additional tools such as the Fourier, Separator, … etc, applications, as well as building the static library with combination with the Android NDK.

6 User Guide

In this Chapter we are going to present a user manual, explaining how UI interacts with the user.

6.1 Application's Installing

Installing the application can be done using one of the following ways:

- Downloading the EarthquakeActivity.apk, and locating it using a file manager, then open the APK file using an application installer application (can be found on Android Market).
- Using the adb tool that is attached to the eclipse ADT Plugin, using a super user mode on Linux, user must run the following command: ./adb install PATH_TO_APK
- Installing Earthquake Detector through Android Market. (future work)

6.2 Application's Launching/Using

Running the application is very simple. After installing the application, user can find a shortcut of the application on his main menu, touching the application shortcut "icon" will automatically start the application.

User is responsible of selecting the sensitivity of the application, to do that, just drag the seek bar left or right (high sensitivity, low sensitivity), the application will change it status depending on the current position of the seek bar.

User also can pick between detecting mode, or training mode, simply by checking the mode radio-button. It's highly recommended to use training only when the application is firing a false alert on special conditions. Special conditions are situations where the application always fires a false alert, such as playing soccer, or dancing, and to use the training mode as rare as possible, and depend more on the sensitivity seek-bar to eliminate false alerts.

Finally, users can exchange training sets with each others. One user can simply copy the training set found in the memory card of each user on this path:

/sdcard/earthquake

Users with worse training set have to copy the whole folder, to their SD card, if for some reason the training set files names are not “SortedCleanVectors” then the user have to use the browse button to select the new training set files, where selecting the folder is enough.

Figure 6 and 7, shows a screen-shot of the application main interface, and the file dialog interface.

6.3 Recommended Use

This application can run under any circumstances, but it's recommended to use it with a stable mobile phone. Using and depending on the application during a normal day activities, is not a very good strategy. The reason is that when walking or driving for example, firing an earthquake alert is not the same important as firing an alarm while sleeping, where the user is unconscious. Our target is “firing an alarm on a life saving situations”, where the user will not be aware of the danger without application's help.

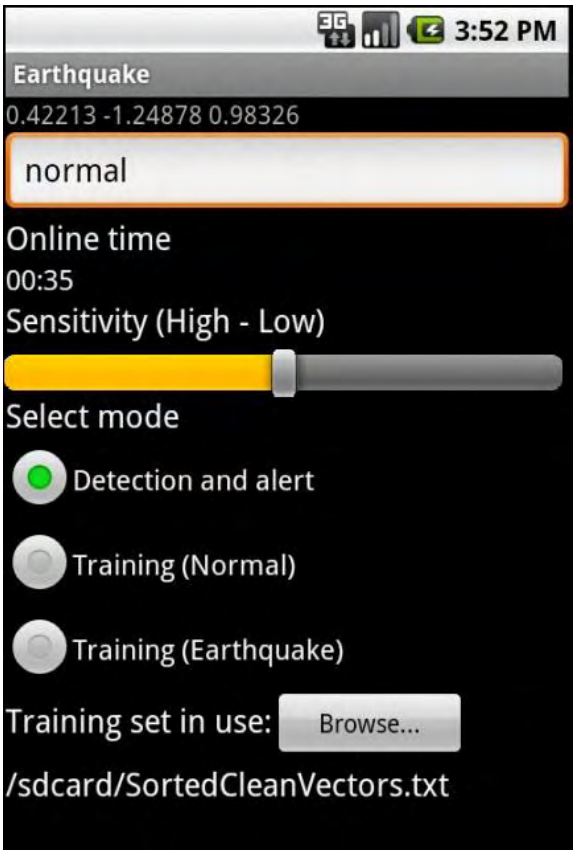


Figure 6 The main application interface

Figure 7 The file dialog interface

7 Conclusion

In this final section, the Earthquake Detector functionalities are restated and potential future work is discussed.

7.1 Conclusion

Implementing a single point earthquake detection application is possible, under the condition of minimal user interference, such as picking the sensitivity level or getting a better training sets, this project is a valid earthquake detection application, simply because it can detect even a very tiny earthquake patterns, when sensitivity level is high, using a stable device while the user is unconscious, which is our target in the first place.

It is highly recommended to get better earthquake training sets. Gathering this data from active seismic areas will help the application decide with higher accuracy, based on known seismic patterns and not unknown daily patterns. These data are very hard to locate and process, due to absence of an earthquake predicting techniques, and the rareness of such events. To collect such data, researchers use earthquake simulator machines, that will produce simulated earthquake patterns with a specific amplitude, depth, power and time duration.

7.2 Future Work

We can suggest those points as a future work to continue this project:

- Implementation of a cloud server that will save and handle seismic events, as sent of different Android clients, and early notify for some seismic upcoming events.
- Searching for better training sets, especially earthquakes patterns, this can be done by cooperation with seismologists, and conversion of normal PGA values to training vectors, that can be saved and used by the application.
- Implementing of a real-time training set updating algorithm, which will make all application clients have the same newest data base, or even all geographical near application clients with the same data base.
- Remove the sensitivity seek-bar, by assuming sensitivity level based on the application main behavior, using the current earthquake knowledge from the Internet, for example, if the application is detecting earthquakes but the real situation as received from the Internet is a normal day with no earthquakes, then the sensitivity level have to lower.
- Building a new UI, with maps and charts of the current acceleration values, and save those charts for future reference if the user wishes to.

8 Appendix A:

Android Developing

Programming on Android platforms, is not very different of JAVA programming. In fact, Android platforms uses JAVA as a main programming language. In this appendix we will present the basics of programming on Android platforms, and we assume that the reader has the basic programming skills on object oriented programming languages, such as JAVA, C++, as well as basic knowledge of Linux OS.

8.1 Where to start

Creating an application for android smart phones/tablets, needs the eclipse IDE installed with the ADT Plugin enabled, to do so, user must download the eclipse IDE 3.6 or greater, you can find the eclipse IDE on this link:

http://www.eclipse.org/downloads/

Next, unpack or extract the tar file, to run the eclipse IDE, use your terminal to navigate to the eclipse root directory, and execute:

sudo ./eclipse

This will start the eclipse SDK, after that, select your working directory, and there you are, ready to use the eclipse.

To install the ADT Plugin, you have to follow the instructions provided by the android developers site, on this link:

http://developer.android.com/sdk/eclipse-adt.html

When successfully download and install the ADT Plugin, a folder named android-sdks, will appear next to the eclipse root folder, this folder contains the adb application, which is a very useful tool as we will explain later.

After finishing the ADT installation, developer must create a new emulator. This can easily be done, by clicking on the window drop-menu on the main menu toolbar of the eclipse IDE, and then selecting the AVD manager. This window have options for creating, editing, and starting the virtual emulator that you created, selecting new AVD will create a new emulator with the SDK that you will select, with an SD card of the size you select, and any other devices you wish to attach to this emulator, such as the sound playback support, and many others.

Congratulations, you are now ready to start the Hello world application.

8.2 Hello World application

To write your first hello world application on Android, please follow this link, which explains analytically and step-by-step how to write an android Hello World application:

http://developer.android.com/resources/tutorials/hello-world.html

This link also have some images of how to create and edit an AVD.

Writing an Android application is not a difficult issue at all. In contrary, the eclipse IDE helps you to find any class or method of any object by just typing the dot after an object name, this live list of all the object methods is really helpful, since the java API is very large and hard to remember. A list of suggestions will always help you to select the suitable method or object, based on a help manual that will appear next to any method you select, which makes programming an Android JAVA very easy and fast.

8.3 Using a real device

Well, the hello world is running fine on the emulator, to try this application on a real Android device, you have to be sure that you have an Android device with SDK that equals the project minimal SDK, or greater.

To run the application on the device, USB data cable must be attached, after this, navigate on your device to settings-> applications-> Unknown Sources, make sure that this option is ticked, also make sure that the USB debugging is also ticked, this option is on settings-> applications->development->USB Debugging.

When everything is ready on the devices side, you have to make sure that your operating system recognize your attached device, to do so, enter the super user mode on your terminal:

sudo su

Then, navigate to your android-sdks folder:

cd PATH_TO_ROOT/android-sdks/platform-tools/

Here the adb tool is installed, the adb tool is your application to manage your device, so if you execute:

./adb devices

You shall see your device serial number, if only question marks are printed, then the adb server is not ready to see your device, and you have to restart it using the following commands:

./adb kill-server

./adb star-server

When you see your devices serial number, you are ready to run the application on your Android device, just by pressing run on the eclipse IDE.

8.4 Android NDK

The Android NDK is a companion tool to the Android SDK that lets you build performance-critical portions of your apps in native code. It provides headers and libraries that allow you to build activities, handle user input, use hardware sensors, access application resources, and more, when programming in C or C++. If you write native code, your applications are still packaged into an .apk file and they still run inside of a virtual machine on the device. The fundamental Android application model does not change.

To download the Android NDK follow this link:

http://developer.android.com/sdk/ndk/index.html

8.5 JNI "Java Native Interface"

The java native interface is an interface that allows your C,C++ code to communicate with the Java Virtual machine, and have access to data arrays and parameters, as well as using object methods, and write back any important data.

When writing a JNI code, some important header files have to be included to C code, such as the jni.h, which have a very important macros implementation.

Declaring a JNI function, is a little different from the classic C function declaring, the function has to follow this scheme:

```
JNIEXPORT "Return type" JNICALL
Java_"Obsolete_Class_path"_FunctionName(JNIEnv *env, jobject
obj, …);
```

Where the Return type is void, int … etc, and the obsolete class path is the application package name, separated with _ , followed by the class name that will eventually call this JNI code.

To get more information about the JNI coding, please refer to:

http://java.sun.com/docs/books/jni/

To build a static library, using the JNI code, you will have to create a folder named jni on the application root directory, and write your C code files inside, create a file named "Android.mk", and then execute the ndk-build application on your application ROOT directory, the ndk-build application is located on:

NDK_FOLDER/ndk-build

Finally, to call the native methods from your JAVA application code, you have to declare the functions as follows:

Public "Return type" FunctionName(…);

And to load the static library created by the NDK, just use the following command:

```
Static{
        System.loadLibrary("LibName");
}
```

DON'T add the .so prefix to the LibName.

For example, let's assume we have a class named JNIClass, that is saved in path: ProjectRoot/src/com/example/jni/JNIClass.java

And that this class will load the static library "JNILib.so", this will affect the C files, so that every native function header must look like this function example:

```
JNIEXPORT void JNICALL
Java_com_example_jni_JNIClass_FunctionExample(JNIEnv      *env,
jobject obj, …);
```

On the other hand, the JNIClass.java file, must contain:

System.LoadLibrary("JNILib");

Public native void FunctionExample(); //The JNIEnv, and jobject are never declared

More information about writing and building the Android.mk can be found here:

http://mobile.tutsplus.com/tutorials/android/ndk-tutorial/

Finally, use help by searching for more JNI functions and tutorials from the Internet, a lot of projects and forums discusses any kind of problems and hints to make your JNI look better and more efficient.

9 Appendix B: Code

In this appendix, we will present the code as used by the Earthquake Detector application.

9.1 Earthquake Activity class

```
package com.earthquack.detector;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.content.res.Configuration;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.media.Ringtone;
import android.media.RingtoneManager;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.Chronometer;
```

```
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.SeekBar;
import android.widget.TextView;


public class EarthequackActivity extends Activity {
        private TextView result;
        private EditText text;
        private Button browse ;
        private TextView fileNameView;
        private String fileName;
        private RadioGroup GroupMode;
        private RadioButton detection, training, trainingEar;
        private SeekBar sensBar;

        public int mode, added=0, warmmingup=0;
        private SensorManager sensorManager;
        private Sensor sensor;
        private float x, y, z;

        private Detector DetectorObj;
        private EarthquakeMath maths;

        private float[] gravity;

        private int samplesNumber = 0;
        private float[][] samples;
        private boolean useAlarm = false, isWarm = false;
        private Ringtone ring;
```

```
        public Ringtone r;
    int REQUEST_SAVE = 0;
    int REQUEST_LOAD = 1;

        private float NOISE_THRESHOLD = (float) 0.141747;           //Experimental noise
maximum

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);

                gravity = new float[3];

                text = (EditText) findViewById(R.id.TheText);
                Chronometer c = (Chronometer) findViewById(R.id.chronometer);
                fileNameView = (TextView) findViewById(R.id.FileName);
                mode = WorkingMode.DETECTION;

                registerListeners();

                samples = new float[190][3];
                DetectorObj = new Detector();
                maths = new EarthquakeMath();

                sensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
                sensor =
sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER).get(0);
                c.start();
```

```
            result = (TextView) findViewById(R.id.result);
            result.setText("no results yet");
        }

        public synchronized void onActivityResult(final int requestCode,
        int resultCode, final Intent data) {

        if (resultCode == Activity.RESULT_OK) {
            fileName = data.getStringExtra(FileDialog.RESULT_PATH);

            fileNameView.setText(fileName);
            DetectorObj.filePathBase = fileName+'/';
            int st = DetectorObj.init();
            if(st == -1){
                text.setText("Invalid folder");
                fileNameView.setText("Invalid folder, please select trainingset");
            }
            else{
                text.setText("Using New path");
            }

        } else if (resultCode == Activity.RESULT_CANCELED) {
            Log.e("ReturnFileDialog","canceled");
        }

    }

        public void registerListeners(){

            GroupMode = (RadioGroup) findViewById(R.id.ModeGroup);
            browse = (Button) findViewById(R.id.browseButton);
```

```
                detection = (RadioButton) findViewById(R.id.DetectionMode);
                training = (RadioButton) findViewById(R.id.TrainingNormal);
                trainingEar = (RadioButton) findViewById(R.id.TrainingEarthquake);
                sensBar = (SeekBar) findViewById(R.id.sensitivity);
                sensBar.setMax(20);
        sensBar.setProgress(10);
        sensBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {

                    public void onStopTrackingTouch(SeekBar arg0) {

                    }


                    public void onStartTrackingTouch(SeekBar arg0) {

                    }


                    public void onProgressChanged(SeekBar arg0, int arg1, boolean arg2) {
                        float progress = (float)((float)arg1 / 2);
                        DetectorObj.DECISSION_THRESHOLD = progress;
                    }
                });
                GroupMode.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener(){

                //      @Override
                        public void onCheckedChanged(RadioGroup group, int checkedId) {
                            if(detection.isChecked()){
                                mode = WorkingMode.DETECTION;
                            }
```

```
                        if(training.isChecked()){
                                mode = WorkingMode.NORMAL_TRAINING;
                        }
                        if(trainingEar.isChecked()){
                                mode =
WorkingMode.EARTHQUAKE_TRAINING;
                        }
                }
        });

        browse.setOnClickListener(new Button.OnClickListener(){
         public void onClick(View v) {
                 Intent intent = new Intent(getBaseContext(), FileDialog.class);
          intent.putExtra(FileDialog.START_PATH, "/sdcard");

          //can user select directories or not
          intent.putExtra(FileDialog.CAN_SELECT_DIR, true);
          intent.putExtra(FileDialog.SELECTION_MODE,
SelectionMode.MODE_OPEN);

          startActivityForResult(intent, REQUEST_LOAD);
         }
    });
  }


  private void action(){
        boolean decide=false;
        float[] eventVector = new float[(WorkingMode.HARMONICS * 2) + 4];
```

```
switch(mode){
case (WorkingMode.DETECTION):{
        eventVector = maths.doMath(samples);

        decide = DetectorObj.decideEarthquake(eventVector);
        if(decide){
                text.setText("EARTHQUACK");
                if(useAlarm)
                        alertStart();
        }
        else{
                if(useAlarm)
                        alertStop();

                text.setText("normal");
        }
        added = 0;
        break;
}
case (WorkingMode.NORMAL_TRAINING):{
        eventVector = maths.doMath(samples);
        boolean saved = DetectorObj.addToTrainingSet(eventVector, mode);
        if(saved){
                added++;
                text.setText("saved !" + Integer.toString(added));
        }
        else{
                text.setText("not saved");
        }
        break;
}
```

```
		case (WorkingMode.EARTHQUAKE_TRAINING):{
			eventVector = maths.doMath(samples);
			boolean saved = DetectorObj.addToTrainingSet(eventVector, mode);
			if(saved){
				added++;
				text.setText("Saved "+Integer.toString(added));
			}
			break;
		}
		}
	}


	private void refreshDisplay() {
		String outString = String.format("%f %f %f\n", x, y, z);

		result.setText(outString);
		if(isWarm){
			samples[samplesNumber][0] = x;
			samples[samplesNumber][1] = y;
			samples[samplesNumber][2] = z;
			samplesNumber++;

			if(samplesNumber == WorkingMode.WINDOW_SIZE){
				action();
				samplesNumber = 0;
			}
		}
		else{
			warmmingup ++;
```

```
                        if (warmmingup >= 10){
                                isWarm = true;
                        }
                }
        }

        private void alertStart(){
        // use a handler, only UI thread can access that
                Uri alert =
RingtoneManager.getDefaultUri(RingtoneManager.TYPE_ALARM);
                if(alert == null){
              // alert is null, using backup
              alert=
RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
              if(alert == null){
                alert=
RingtoneManager.getDefaultUri(RingtoneManager.TYPE_RINGTONE);
              }
            }
                ring = RingtoneManager.getRingtone(getApplicationContext(), alert);
                if(ring == null){
                        Log.e("Alert","Null alert");
                }
                else{
                        ring.play();
                }
                try{
                        Thread.sleep(2000);

                }catch(InterruptedException e){
                        e.printStackTrace();
```

```
        }
    }

    private void alertStop(){
        if(ring != null && ring.isPlaying()){
            ring.stop();
        }
    }

    private SensorEventListener accelerationListener = new SensorEventListener() {
        //@Override
        public void onAccuracyChanged(Sensor sensor, int acc) {
        }

        public void GravityHighPassFilter(SensorEvent event){
            // alpha is calculated as t / (t + dT)
        // with t, the low-pass filter's time-constant
        // and dT, the event delivery rate

            final float alpha = (float) 0.8;

        gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
        gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
        gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

        x = event.values[0] - gravity[0];
        if(Math.abs(x) < NOISE_THRESHOLD)
            x = (float) 0.0;

        y = event.values[1] - gravity[1];
        if(Math.abs(y) < NOISE_THRESHOLD)
```

```
            y = (float) 0.0;

        z = event.values[2] - gravity[2];
        if(Math.abs(z) < NOISE_THRESHOLD)
            z = (float) 0.0;
          }

          //@Override
          public void onSensorChanged(SensorEvent event) {
                  GravityHighPassFilter(event);
                  refreshDisplay();
          }

};

@Override
public void onConfigurationChanged (Configuration newConfig){
          Log.e("New Config","Orientation Changed");
          super.onConfigurationChanged(newConfig);
          setContentView(R.layout.main);
}

@Override
public void onDestroy(){
          System.exit(1);
}

@Override
protected void onResume() {
          super.onResume();
          sensorManager.registerListener(accelerationListener, sensor,
```

```
                                SensorManager.SENSOR_DELAY_GAME);
        }

        @Override
        protected void onStop() {
                sensorManager.unregisterListener(accelerationListener);
                super.onStop();
        }
}
```

9.2 Earthquake Math class

```
package com.earthquack.detector;

public class EarthquakeMath {
        public EarthquakeMath(){
        }
        public float[] doMath(float[][] samples){
                float[] XYNorm;
                float[] XYFourier = new float[WorkingMode.HARMONICS];
                float[] XYFourierB = new float[WorkingMode.HARMONICS];
                float[] eventVector = new float[(WorkingMode.HARMONICS * 4) + 4];

                float var, abs;
                int pos = 0;
                XYNorm = findXYNorm(samples);

                findACoefficients(XYNorm, XYFourier);
                findBCoefficients(XYNorm, XYFourierB);
```

```
for(int i = 0, x = 0; i < WorkingMode.HARMONICS * 2; i+=2, x++){
        eventVector[i] = XYFourier[x];
        eventVector[i+1] = XYFourierB[x];
}
pos += XYFourier.length + XYFourierB.length;

var = findVariance(XYNorm);
eventVector[pos] = var;
pos++;

abs = findAbsoluteMaximum(XYNorm);
eventVector[pos] = abs;
pos++;

float[] zVector = new float[WorkingMode.WINDOW_SIZE];
float[] Zfourier = new float[WorkingMode.HARMONICS];
float[] ZfourierB = new float[WorkingMode.HARMONICS];

for(int i=0; i<WorkingMode.WINDOW_SIZE; i++){
        zVector[i] = samples[i][2];
}

findACoefficients(zVector, Zfourier);
findBCoefficients(zVector, ZfourierB);

for(int i = 0, x = 0; i < WorkingMode.HARMONICS * 2; i+=2, x++){
        eventVector[pos + i]     = Zfourier[x];
        eventVector[pos + i + 1] = ZfourierB[x];
}

pos+= Zfourier.length + ZfourierB.length;
```

```
                var = findVariance(zVector);
                eventVector[pos] = var;
                pos++;

                abs = findAbsoluteMaximum(zVector);
                eventVector[pos] = abs;
                pos++;

                // Event Vector is ready to be checked ...
                return(eventVector);
        }

        private float[] findXYNorm(float[][] samples){
                float[] output = new float[WorkingMode.WINDOW_SIZE];

           int i;

           for(i=0; i<WorkingMode.WINDOW_SIZE; i++){
                output[i] = (float)(Math.sqrt((float)Math.pow(samples[i][0] -
(float)samples[i][1], 2.0)));
           }

                return output;
        }

        private float[] findACoefficients(float[] input, float[] output){
           int i, t;
           float sum;
           final int period = input.length;
```

```
            sum = (float) 0.0;

            for(i=0; i < period; i++)
                sum += input[i];

            output[0] = sum / period;

            for(i=1; i < WorkingMode.HARMONICS; i++){
                sum = (float) 0.0;
                    for(t=0; t < period; t++){
                       sum += input[t] * (float)Math.cos((double)(Math.PI * (float)i *
(float)t / (float)period));
                    }
                    output[i] = sum / period;
            }

            return(output);
        }

        private float[] findBCoefficients(float[] input, float[] output){
            int i, t;
            float sum;
            final int period = input.length;

            sum = (float) 0.0;

            for(i=0; i < WorkingMode.HARMONICS; i++){
                sum = (float) 0.0;
                    for(t=0; t < period; t++){
                       sum += input[t] * (float)Math.sin((double)(Math.PI * (float)i *
(float)t / (float)period));
```

```
                    }
                    output[i] = sum / period;
            }

            return(output);
        }

        private float findVariance(float[] input){
                float var = (float) 0.0;
            float sum, av;
            int i;

            sum = (float)0.0;
            for(i=0; i<WorkingMode.WINDOW_SIZE; i++){
                sum+=input[i];
            }
            av = sum/WorkingMode.WINDOW_SIZE;                    //mean
value of input array

            sum = (float)0.0;
            for(i=0; i<WorkingMode.WINDOW_SIZE; i++){
                sum += Math.pow((float)(input[i] - av),2.0);
            }
            var = sum/WorkingMode.WINDOW_SIZE;

                return var;
        }

        private float findAbsoluteMaximum(float[] input){
                float max = (float) 0.0;
                int i;
```

```
            for(i=0; i<WorkingMode.WINDOW_SIZE; i++){
                if(Math.abs(input[i]) > max){
                    max = Math.abs(input[i]);
                }
            }
                return  max;
        }
}
```

9.3 Detector class

```
package com.earthquack.detector;



public class Detector {
        public String filePathBase = "/sdcard/earthquake/";
        public float DECISSION_THRESHOLD = 1;
        private float[] trainingSet, earthquakeSet;
        private float[][] trainingSetBases;
        private int[] trainingSetLengths;
        private int lastFileNumber = -1;

        private int totalSamples, earthquakeTotal;
        private int samplesSize = WorkingMode.SAMPLES_SIZE;

        static{
                System.loadLibrary("data");
        }
```

```
public native void loadFile(float[] output, int size, String path);
public native int loadFirst(float[] output, String path);
public native void saveFile(float[] input, int size, String path);

public Detector(){
	String path = filePathBase+"SortedCleanVectors";
	int i;
	float[] output = new float[36];

	trainingSetBases = new float[100][36];
	trainingSetLengths = new int[100];

	for(i=0; i<100; i++){
		String pathNow = path;
		if(i<10){
			pathNow = path + "0";
		}
		pathNow = pathNow + Integer.toString(i);
		trainingSetLengths[i] = loadFirst(output, pathNow);

		System.arraycopy(output, 0, trainingSetBases[i], 0, samplesSize);
	}

	earthquakeTotal = loadFirst(output, filePathBase+"earthquakeset.txt");
	earthquakeSet = new float[earthquakeTotal * samplesSize];
	loadFile(earthquakeSet, earthquakeTotal, filePathBase+"earthquakeset.txt");
}

public int init(){
	String path = filePathBase+"SortedCleanVectors";
	int i;
```

```
        float[] output = new float[36];


        for(i=0; i<100; i++){
                String pathNow = path;
                if(i<10){
                        pathNow = path + "0";
                }
                pathNow = pathNow + Integer.toString(i);
                trainingSetLengths[i] = loadFirst(output, pathNow);
                if(trainingSetLengths[i] == -1){
                        break;
                }
                System.arraycopy(output, 0, trainingSetBases[i], 0, samplesSize);
        }
        if(i < 100){
                return(-1);                                     //error
        }
        return(1);


}
private int findPosition(float[] input, float[] trainingSetSel, int totalSamplesSel){
        int i,j;
        for(i=0; i < totalSamplesSel; i++){
                j=0;
                while(j<samplesSize){
                        if(input[j] == trainingSetSel[(i * samplesSize) + j]){
                                j++;
                                continue;
                        }
                        else if(input[j] != trainingSetSel[(i * samplesSize) + j]){
                                break;
```

```
                }
            }
        if(j == samplesSize || input[j] < trainingSetSel[(i * samplesSize) + j]){
                break;
            }
            if(input[j] > trainingSetSel[(i * samplesSize) + j]){
                continue;
            }
        }
        return(i);
    }

    private int findFile(float[] input){
        int i, j;
        for(i=0; i<100; i++){
            j=0;
            while(j < samplesSize){
                if(input[j] == trainingSetBases[i][j]){
                    j++;
                    continue;
                }
                else if(input[j] != trainingSetBases[i][j]){
                    break;
                }
            }
            if(j == samplesSize || input[j] < trainingSetBases[i][j]){
                break;
            }
            if(input[j] > trainingSetBases[i][j]){
                continue;
            }
```

```
            }

            return(i - 1);
        }

        private void saveTemp(float[] input, int size, int fileNumber){
            String fileName = "SortedCleanVectors";

            if(fileNumber < 10){
                fileName = filePathBase+"SortedCleanVectors0";
            }
            fileName = fileName + Integer.toString(fileNumber);

            saveFile(input, size, fileName);
        }

        private void loadFile(int fileNumber){
            String fileName = filePathBase+"SortedCleanVectors";

            if(fileNumber < 10){
                fileName = filePathBase+"SortedCleanVectors0";
            }

            fileName = fileName + Integer.toString(fileNumber);
            int size = trainingSetLengths[fileNumber];

            trainingSet = new float[size * samplesSize];
            loadFile(trainingSet, size, fileName);
            totalSamples = size;
        }
```

```
public boolean addToTrainingSet(float[] input, int mode){
	int coordI, fileNumber;

	if(mode == WorkingMode.NORMAL_TRAINING){
		float[] temp = new float[(totalSamples + 1) * samplesSize];
		fileNumber = findFile(input);
		if(fileNumber == -1){
			fileNumber = 0;
		}
		if(fileNumber != lastFileNumber){
			loadFile(fileNumber);
			lastFileNumber = fileNumber;
		}
		coordI = findPosition(input, trainingSet, totalSamples);
		int i;
		for(i=0; i<samplesSize; i++){
			if(trainingSet[(coordI * samplesSize) + i] != input[i]){
				break;
			}
		}
		if(i == samplesSize){
			return false;
		}
		for(i=0; i<samplesSize; i++){
			if(trainingSet[((coordI - 1) * samplesSize) + i] != input[i]){
				break;
			}
		}
		if(i == samplesSize){
			return false;
		}
```

```
                System.arraycopy(trainingSet, 0, temp, 0, (coordI) * samplesSize);
                System.arraycopy(input, 0, temp, ((coordI) * samplesSize), samplesSize);
                System.arraycopy(trainingSet, coordI * samplesSize, temp, (coordI + 1) *
samplesSize, (totalSamples - coordI) * samplesSize);

                        saveTemp(temp, trainingSetLengths[fileNumber] + 1, fileNumber);
                        trainingSet = new float[(totalSamples +1) * samplesSize];

                System.arraycopy(temp, 0, trainingSet, 0, (totalSamples +1) * samplesSize);
                        lastFileNumber = -1;

                        return(true);
                }
                else{           //add to earthquake training set
                        float[] temp = new float[(earthquakeTotal + 1) * samplesSize];
                        coordI = findPosition(input, earthquakeSet, earthquakeTotal);

                System.arraycopy(earthquakeSet, 0, temp, 0, (coordI) * samplesSize);
                System.arraycopy(input, 0, temp, ((coordI) * samplesSize), samplesSize);
                System.arraycopy(earthquakeSet, coordI * samplesSize, temp, (coordI + 1) *
samplesSize, (earthquakeTotal - coordI) * samplesSize);

                        saveFile(temp, earthquakeTotal+1,
filePathBase+"earthquakeset.txt");
                        earthquakeTotal++;
                        earthquakeSet = new float[(earthquakeTotal + 1) * samplesSize];

                        System.arraycopy(temp, 0, earthquakeSet, 0, (earthquakeTotal) *
samplesSize);
                        return(true);
                }
```

```
	}

	public boolean decideEarthquake(float[] input){
		boolean isEarthquack = false;
		int j = 0;
		int coordI, fileNumber;
		float disIminusOne = 0, disI = 0;
		float result, earthquakeResult;

		fileNumber = findFile(input);
		if(fileNumber == -1){
			fileNumber = 0;
		}
		if(fileNumber != lastFileNumber){
			loadFile(fileNumber);
			lastFileNumber = fileNumber;
		}
		else{
			//already loaded file
		}

		/*at this point i,j has the coordinates of where this input should enter (nearest pointes are tra.[i-1] and tra.[i])... */
		/* find root((x1-x2)^2 + (y1-y2)^2 + ...) of each */
		coordI = findPosition(input, trainingSet, totalSamples);

		if(coordI < totalSamples){
			for(j=0; j<samplesSize; j++){
				disI += (float)Math.pow((float)(input[j] - trainingSet[(coordI * samplesSize)+j]), 2.0);
			}
```

```
                disI = (float)Math.sqrt(disI);
                if(disI == 0){
                        disIminusOne = 1000;
                }
        }

        if(coordI > 0 && disI != 0){
                for(j=0; j<samplesSize; j++){
                        disIminusOne += (float)Math.pow((float)(input[j] -
trainingSet[((coordI - 1) * samplesSize)+j]), 2.0);
                }
                if(disIminusOne == 0){
                        disI = 1000;            //match
                }
        }
        disIminusOne = (float)Math.sqrt(disIminusOne);

        if(disIminusOne < disI){
                result = disIminusOne;
        }
        else{
                result = disI;
        }

        /* LOOKING UP EARTHQUAKE TRAINING SET TO COMPARE */
        coordI = findPosition(input, earthquakeSet, earthquakeTotal);

        if(coordI < earthquakeTotal){
                for(j=0; j<samplesSize; j++){
                        disI += (float)Math.pow((float)(input[j] -
earthquakeSet[(coordI * samplesSize)+j]), 2.0);
```

```
                }
                disI = (float)Math.sqrt(disI);
                if(disI == 0){
                        disIminusOne = 1000; //match
                }
        }

        if(coordI > 0 && disI != 0){
                for(j=0; j<samplesSize; j++){
                        disIminusOne += (float)Math.pow((float)(input[j] -
earthquakeSet[((coordI - 1) * samplesSize)+j]), 2.0);
                }
                if(disIminusOne == 0){
                        disI = 1000; //match
                }
        }
        disIminusOne = (float)Math.sqrt(disIminusOne);

        if(disIminusOne < disI){
                earthquakeResult = disIminusOne;
        }
        else{
                earthquakeResult = disI;
        }

        result = result /3;

        if(result  < earthquakeResult + DECISSION_THRESHOLD){
                isEarthquack = false;
        }
        else{
```

```
                    isEarthquack = true;
                }
                return isEarthquack;
            }
}
```

9.4 Libdata.so

```
#include <assert.h>
#include <jni.h>
#include <string.h>
#include <android/log.h>
#include <stdio.h>

/*for android logs*/
#define LOG_TAG "DataJNI"
#define LOG_LEVEL 10
#define LOGI(level, ...) if (level <= LOG_LEVEL) {__android_log_print(ANDROID_LOG_INFO, LOG_TAG, __VA_ARGS__);}
#define LOGE(level, ...) if (level <= LOG_LEVEL) {__android_log_print(ANDROID_LOG_ERROR, LOG_TAG, __VA_ARGS__);}

#define SamplesSize 36

JNIEXPORT int JNICALL
Java_com_earthquack_detector_Detector_loadFirst(JNIEnv * env, jobject obj, jfloatArray output, jstring jpath)
{
   int trash, i, j;
   int samplesNumber=0;
   jfloat *outbuffer = (*env)->GetFloatArrayElements(env, output, NULL);
   const jbyte *str;
```

```
    str = (*env)->GetStringUTFChars(env, jpath, NULL);
    FILE *f;

    if(!(f=fopen(str, "r"))){
        LOGE(1,"File %s cannot load\n",str);
        return(-1);
    }
    (*env)->ReleaseStringUTFChars(env, jpath, str);

    fscanf(f, "%d", &samplesNumber);
    fscanf(f, "%d", &trash);

    for(i=0; i<SamplesSize; i++)
        fscanf(f,"%f ",&outbuffer[i]);

    fclose(f);
    return(samplesNumber);
}

JNIEXPORT void JNICALL
Java_com_earthquack_detector_Detector_loadFile(JNIEnv * env, jobject  obj, jfloatArray
output, jint size, jstring jpath)
{
    jclass          cls = (*env)->GetObjectClass(env, obj);
    FILE *f;
    int i,j;
    jfloat *outbuffer = (*env)->GetFloatArrayElements(env, output, NULL);
    int ss = (int)size;
    const jbyte *str;
    str = (*env)->GetStringUTFChars(env, jpath, NULL);
```

```
    if(!(f=fopen(str, "r"))){
        LOGE(1,"File %s cannot load\n",str);
        return;
    }
    (*env)->ReleaseStringUTFChars(env, jpath, str);
    fscanf(f,"%d", &ss);
    fscanf(f,"%d", &ss);

    for(i=0; i<size; i++){
        for(j=0; j<SamplesSize; j++){
            fscanf(f,"%f ",&outbuffer[(i*SamplesSize)+j]);
        }

    }
    fclose(f);
}

JNIEXPORT void JNICALL
Java_com_earthquack_detector_Detector_saveFile(JNIEnv * env, jobject obj, jfloatArray input, jint size, jstring jpath)
{
    jclass          cls = (*env)->GetObjectClass(env, obj);
    FILE *f;
    int i,j;
    jfloat *inbuffer = (*env)->GetFloatArrayElements(env, input, NULL);
    int ss = (int)size;
    const jbyte *str;
    str = (*env)->GetStringUTFChars(env, jpath, NULL);

    if(!(f=fopen(str, "w"))){
        LOGE(1,"File %s cannot open\n",str);
```

```
        return;
    }

    (*env)->ReleaseStringUTFChars(env, jpath, str);

    fprintf(f, "%d %d\n",size, SamplesSize);

    for(i=0; i<size; i++){
        for(j=0; j<SamplesSize; j++){
            fprintf(f, "%f ",inbuffer[(i*SamplesSize)+j]);
        }
        fprintf(f,"\n");
    }
    fclose(f);
}
```

9.5 Constant classes

- WorkingMode.java:

```
package com.earthquack.detector;

public class WorkingMode {
      public static final int DETECTION = 0;
      public static final int NORMAL_TRAINING = 1;
      public static final int EARTHQUAKE_TRAINING = 2;
      public static final int SAMPLES_SIZE = 36;
      public static final int WINDOW_SIZE = 190;
      public static final int HARMONICS = 8;
}
```

- SelectionMode.java

```
package com.earthquack.detector;
public class SelectionMode {
      public static final int MODE_CREATE = 0;
      public static final int MODE_OPEN = 1;
}
```

10 Bibliography

[1] The next big one: Detecting Earthquakes and other Rare events from community-based Sensors, by Matthew Faulkner, Michael Olson, Rishi Chandy, and Jonathan Krause, California Institute of technology.
[2] The Rapid Detection of Rare Geospatial Events: Earthquake Warning applications, by Michael Olson, Annie Liu, and Matthew Faulkner, California Institute of technology.
[3] Fourier Series, Wikpedia, http://en.wikipedia.org/wiki/Fourier_series
[4] Android Sensors Tutorial, by Lars Vogel, http://www.vogella.com/articles/AndroidSensor/article.html
[5] Google project, Android file dialog, http://code.google.com/p/android-file-dialog/
[6] The Physics of earthquakes, by Hiroo Kanamori and Emily E Brodsky, Department of Earth & Space Sciences, University of California, Los Angeles, Los Angeles.