**Partial Discharge Classification using AI Techniques**

Final Report for CS39440 Major Project

*Author*: James Slater ([jas38@aber.ac.uk](mailto:jas38@aber.ac.uk))

*Supervisor*: Dr Neil MacParthalain ([ncm@aber.ac.uk](mailto:ncm@aber.ac.uk))

6th March 2012

Version 1.0 (Draft)

This report is submitted as partial fulfilment of a BSc degree in  
Artificial Intelligence and Robotics (GH7P)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

**Declaration of originality**

In signing below, I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the sections on unfair practice in the Students’ Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
* I understand and agree to abide by the University’s regulations governing these issues.

Signature ……………………………………………. (James Slater)

Date …………………………………………………

**Consent to share this work**

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature ……………………………………………. (James Slater)

Date …………………………………………………

**Acknowledgements**

I am grateful to…

I’d like to thank…

**Abstract**

The aim for this project is to create an Android App, which can classify data from a test instrument, which detects Partial discharge in power distribution networks. The detection instrument will be provided by EA Technology (the industry partner for this project).  Partial discharge signatures are used in order to detect the deterioration of the insulation properties relating to cabling, switching, transformer or other components of an electrical network. Partial discharge has the effect of causing a progressive deterioration of insulating materials, which will eventually lead to complete electrical breakdown. The effects of partial discharge within high voltage cables and equipment can be very serious, ultimately leading to complete failure and very high cost of replacement. The early detection of insulation breakdown in networks therefore offers a great number of advantages.

The App proposed in this project is designed for engineers to use for high voltage asset (cabling and infrastructure network) management; the instrument will provide data to the App in order to determine if an asset (cable, transformer, switching network, etc.) is deteriorating by examining the noise signature. If time permits, additional functionality will be added such that raw audio signal can be sent to the Android device, and processed in real-time, otherwise it will use saved data. AI techniques will be used to develop a means of processing and classifying the demodulated ultrasonic (audio) waveforms.

The conclusion is the App will classify different noise signatures and be able to discriminate between Partial Discharge, background noise or other interference (i.e. transformer noise).

**Contents**

1. Background, Analysis & Process 7

1.1. The problem 7

1.1.1. Partial Discharge Basics 7

1.1.2. Ways of detecting Partial discharge 7

1.1.3. Android Device capabilities and limitations 8

1.1.4. Features extraction 8

1.1.5. Artificial Intelligence Classifiers 9

1.2. Analysis 10

1.3. Process 12

2. Design 13

2.1. Overall Architecture 13

2.2. Detailed Design 13

2.2.1. Structural View 15

2.2.2. Implementation View 15

2.2.3. Behavioural View 16

2.3. User Interface Design 17

2.4. Other Relevant Sections 19

3. Implementation 21

3.1. Final Application 21

3.1.1. Recording 21

3.1.2. Classification 21

3.2. Spike work 24

3.2.1. Understanding Android 4.4+ 24

3.2.2. Loading WAV Audio files in Android 25

3.2.3. Feature extraction 25

3.3. Issues that arose 26

4. Testing 28

4.1. Overall Approach to Testing 28

4.2. Automated Testing 28

4.2.1. Unit Tests 28

4.2.2. User Interface Testing 28

4.2.3. Stress Testing 28

4.3. Integration Testing 28

4.4. User Testing 28

5. Critical Evaluation 29

6. Appendices 30

A. Third-Party Code and Libraries 30

6.1. Project Specification 30

6.1.1. Android App 30

6.2. UML Diagrams 30

6.2.1. Class Diagrams 30

6.3. UML Creator Visual-Paradigm 33

http://www.visual-paradigm.com/ 33

6.4. Feature Spike work 33

6.5. jAudio Features Analysis 33

6.6. Testing Tables 33

6.6.1. Load the application. 33

6.6.2. Main Menu Screen 33

6.6.3. Record screen 34

6.6.4. Save As Screens 34

6.6.5. File Chooser Screen 35

6.6.6. View Data screen 35

6.7. Final Screen Shots 37

B. Code Samples 37

6.8. Graph View 37

6.9. jAudio 2.0 37

6.10. Libsndfile 37

6.11. Recording WAV format audio in Java 37

6.12. YAAFE 38

6.13. Stack Overflow 38

7. Annotated Bibliography 39

# Background, Analysis & Process

This project will be undertaken in conjunction with a company called EA Technology, who work in the electrical industry offering services and instruments to help manage the state of the High/Medium voltage assets. Working at EA Technology over my year in industry provided me with the motivation and background knowledge for this project.

## The problem

Partial discharge is one of the leading causes of failures in electrical assets in a power system. An electrical asset can be a thing like cables, transformers or switching networks. As an example, a transformer converts between different voltage levels and is one of the most critical aspects of the system. Many transformers today are in service past their designed life cycle, and keeping them in service makes economical sense considering the cost of repairs and replacements. A transformer failure can result in a small explosion damaging surrounding transformers, a loss of power to a portion of the population and ensue a fine to the electrical provider. Consequently it is essential to be able to detect the condition of an asset and to plan its service life.

Several methods can be used for monitoring changes in the electrical asset; an extensive list of available methods is available in [12]. I will be focusing on using the ultrasonic sound released by partial discharge as my data as a way of classifying the state of an asset.

### Partial Discharge Basics

The Partial Discharge User Group defined partial discharge as follows, “Partial discharge is an electrical discharge or spark that bridges a small portion of the insulation between two conducting electrodes. Partial discharge activity can occur at any point in the insulation system, where the electric field strength exceeds the breakdown strength of that portion of the insulating material”[1]

Partial discharge has a deteriorating effect and over time it reduces the lifetime of an insulation system. Partial discharge, either on the surface or inside an electrical insulation, incurs high-energy electrons or ions to deteriorate the insulation material. This may result in chemical decomposition of the insulation material. [18]

Partial discharge is a sign that the electrical asset has begun to wear and may fail, hence it is very important to be able to detect and monitor the condition of electrical assets to better understand the wear and assess if an asset will need replacing.

While looking into different ways of researching partial discharge, I came across a paper published by Cigre, which has looked at the many forms of partial discharge via an oscilloscope [14].  I will keep this research in mind as I work through my own work, however as I am using ultrasonic data an oscilloscope will not be of use to me. Having said this, the research provides a better understanding of how each sub group of partial discharge happens and their effects on electrical assets.

### Ways of detecting Partial discharge

**Electrical Detection**

Two types of sensors are usually used for partial discharge measurement in transformers: capacitive and inductive coupling sensors. Capacitive sensors work by measuring changes in an electrical property, called capacitance. Capacitance describes how two conductive objects with a space between them respond to a voltage difference applied to them. Inductive sensors measure inductance of an electromagnetic field in the close surroundings of the sensing surface of the electrical asset. Inductive sensors are common measuring devices for rapidly varying currents for example in transformers.[16]

**Acoustic detection**

The acoustic wave, audible or not, is due to the expansion of gases near the partial discharge, which propagates as a pressure wave. The main frequency used for acoustic detection is between 10 kHz to 1000 kHz and usually ultrasonic partial discharge detectors are tuned at 40 kHz. Acoustic detection has the advantage of being able to localise the partial discharge source. It calculates the time difference for ultrasonics to reach multiple successive sensors within the transformer tank, thus pinpointing the location.

**Chemical detection**

Detection of chemical by-products produced by partial discharge activity is one of the simplest methods for partial discharge detection. When the electrical charge exceeds the field strength of air a spark is created that arcs across a void in the insulating material, which is an example of partial discharge. This breaks down the surrounding insulator into it’s chemical components, which can then be detected chemically. The two primary chemical tests employed by power companies today are dissolved gas analysis (DGA) and high performance liquid chromatography (HPLC).[17]

This type of detection is very well established, immune against noise and relatively easy to measure. However it cannot say much about the type of defect, location and intensity of partial discharge, and it also involved taking the asset offline.

### Android Device capabilities and limitations

Android is the chosen operating system of the customer EA Technology. Android is available worldwide on a wide range of devices, hence accessible to all of EA Technology’s customers.

Android was established in 2008, and the last 7 years have seen much iteration of Android versions. Each version was named after a confectionary item, for example the current version 5.0 has the name Lollipop.

It is possible to make an App backwards compatible, and after some conversation with EA Technology it was decided to making all devices running 4.4+ onwards compatible with the app.

The minimum specifications for running Android 4.4 is 512 RAM, see [TODO Ref the Doc] for the full list of hardware requirements. I will be designing and testing the app on a Samsung S4, which has a specification as follows [20]: -

* Screen - 5.0 inches
* Processor - Quad-core 1.6 GHz Cortex-A15 & quad-core 1.2 GHz Cortex-A7
* Sensors - Accelerometer, gyro, proximity, compass, barometer, temperature, humidity, gesture
* Memory 2 GB RAM, 8GB Internal storage
* Battery - Li-Ion 2600 mAh battery

### Features extraction

All Artificial Intelligence classification techniques are dependent on the selection of features available. Partial discharge has three main data criteria:

* Phased-Resolved data
* Time-Resolved data
* Data without Phase/Time information

Phase-Resolved data is the acoustic data captured with relation to the AC test voltage. The test voltage is considered to be constant and the phase angle is split up to plot any partial discharge activity within the divided phase angle. Partial discharge is most likely to occur as the voltage builds, in other words when the phase is approaching a peak.

Time-Resolved data displays the true shapes of the individual partial discharge pulses. The test voltage is treated as a constant, as seen in Phase-Resolved data. This data pattern has attractive advantages, since there are some direct relationships between the physics of the defect and the shape of the partial discharge signal.

Data without Phase/Time information captures a typical partial discharge pulse where time and phase data are not recorded as a reference.

**Statistical methods for feature extraction**

Each WAV audio file can be represented as an array of values between 1 and -1, therefore many statistical functions can be applied to study the audio file.

For example: Let N be the number of values in the array.

The average of all the values can show a "measure of central tendency", i.e. a statistic that represents a typical value from the data set.

The variance measures the spread of the data set. A variance of zero indicates that all the values are identical.

Skewness is a measure of the symmetry of the data, which can distinguish noise in an audio file.

These statistics are a small sample of features, which can be collated into a vector to summarise an instance of the audio file, therefore allowing the possibility of comparison of multiple instances.

**Signal processing tools**

Signal processors can be used to extract features from the audio data, such as Fourier transform, wavelet transforms and Haar transforms.

A fast Fourier transform on the audio signal can show the relationship between the test AC voltage and the partial discharge activity. Fourier transforms only consider the frequency and not time domain, which is a disadvantage because \*\*\*\*\*\*. To overcome this, wavelet transforms can be used however at present, there is limited research into this area.

### Artificial Intelligence Classifiers

**Distance/Decision function classifiers**

These classifiers works on the data from other already classified and uses the information as a reference. The distance each data feature is from the features from the already classified features and the distance they are apart can be the decision of the classification and new data.

Examples of these classifiers are Minimum Distance, Manhattan Distance, and Nearest Neighbor.

Minimum distance classifier is the distance defined as an index of similarity so that the minimum distance is identical to the maximum similarity.

Manhattan Distance classifier, if an attribute is numeric, then the local distance function can be defined as the absolute difference of the values. Therefore the global distance is computed as the sum of these local distances, and then we refer to it as the Manhattan distance. Weighted sums and weighted averages are also possible.

Nearest Neighbour classifier, works by identifying the nearest neighbours to a query example and using those neighbours to determine the class of the query. For example the classifier assigns an instance to the class most heavily represented among it neighbours. This is based on the idea that the closer it is to another instance the more likely it is to part of the same class.

**Statistical Classifiers**

This approach is to find statistical features to separate the data and to classify the results. Statistical learning theory models this as a function estimation problem.

There are a couple of choices to use a classifier for statistical data, for example Bayes Classifier and support vector machines.

Bayes Classifier is to predict the values of features for members of that class. Examples are grouped in classes because they have common values for the features.

Support vector machines, is to find the optimal hyper plane for linear separable data. It can extend to patterns that are not linearly separable by transformations of original data.

**Neural Networks**

Artificial Neural networks have been very successful in pattern recognition and classification problems. The basic advantage of ANN over other classifiers is its ability to learn from examples. Knowledge in the training set is extracted and stored in the connection weights and neuron biases during the learning phase [23]. However once an ANN has been taught it cannot learn new instances and thus having to re teach a new ANN.

## Analysis

The problem is to use the saved or recorded WAV audio files, to classify if the asset the instrument have measured is showing signs of PD. There are many options to chose from for which AI technique to use, weighing up the pro and cons of each and the fact from my meeting with EA Technology the chance of having new data that will need to be classified further than just a binary classification (is or isn’t partial discharge).

To do this the feature extraction I feel is the key, from the background research done and talking with my supervisor we felt that a statistical feature extraction would be provide the best outcome. Maybe not at the first but having the ability to build upon the system and add complexity to be able to better classify the data. For example if by just checking the features we extract we could get a 80% correct classifier but then by adding on some complexity such as a fuzzy logic classifier we could make a 90+% correct classifier.

From my research there are many forms of PD and as many forms of noise that can effect the results within ultrasonic detection. Such as corona has a different thumbprint to surface discharge. This is shown in the Cigre paper and from other papers the acoustic thumbprint also differs.

Noise can play a major factor in capturing acoustic data, such as wind or other sources of sound ultrasonic or not in our audible bandwidth.

After a meeting with the domain experts at EA Technology they confirmed that they could hear a difference is the play back of the recorded difference and pointed out what they were listening out for.

In this meeting they went through how the data was captured and how they processed the raw data. The instrument uses a band pass filter to a 40KHz heterodyne process to convert the raw audio to a digital signal to save as a WAV audio file. I can use this information to process the WAV file to convert it back to the original raw data this may provide more information than the compressed data saved.

From the background reading I knew the main aspect of the work will be analysing the WAV file and being able to pick out relevant features. To do this I did some spike work into how to read in the audio file using Java. I found just normal playback was very simple in Java, however when trying to load a WAV file and convert the binary data to a double array within the Android environment was very tricky.

After the meeting with EA Technology they use a library called libsndfile, this is a C/C++ library that reads audio files of which WAV is one of the formats. As the WAV files can many different forms from sample rate to how many channels were recorded (Up to 4), finding a library that can manage this will be very beneficial as it will be fully tried and tested and have a community behind it for debugging and help on any problems.

I had to do some more spike work into getting native code running within the Android environment, there were some very good tutorials and I was able to make a small sample app, passing in string and integers from the Java classes to C files, doing the basics that I will need for the project. From this spike work using this libsndfile library is possible and makes reading the audio files much more efficient, tried and tested.

After the meeting with EA Technology I took some time trying some statistical analysis on the audio files against the classification each has to try and separate the partial discharge from the non-partial discharge. I found Skewness might be the major factor as this shows how noisy the data is.

I feel a Neural network will not be right for the job as it is more a black box and wont be beneficial later on in the project as to add a new classification you would need a new neural network to be taught. Using a Bayes network also is not right as to make a true statistical model of each electrical asset with the information needed would not be available and thus only be able to half implement the Bayes system.

An off the cuff idea was to add an image recognition approach to the plot that the audio files creates and finding similarities. However the processing power needed for this would be very high, and not viable across the Android mobile range.

The customer EA Technology and I discussed the goals of the project and what would both want to see as the final outcome. EA Technology were very interested in the feature extraction and classifier a lot more so than the Android device, and offered resources so that I was not limited to the power of the device such as a server to process the features on.

In 6.1 Project Specification there are the aspects of the project that need to be delivered.

This section is what I will be comparing and evaluating myself self come the end of the project.

The main aspects I can see from the specification are:

* Feature extraction
* A classifier to classify new data
  + Split the data provided into training/testing sets, so we can test the classifier
* This deployed on an android device running 4.4+
  + The app will record and save the audio
  + Load and classify the data
  + Classify real-time data
* Unit test and test tables to show the app fully tested

From this specification provided I was able to make it clear that the time left I would not be able to have all these goals achieved, and while I understood that the main area they are interested in is the classifier, I made this my priority as well. I was able to say that out of the main points of the specification the real-time classification will be the hardest and would be best as a sub goal only if time permits.

## Process

EA Technology use waterfall model to keep track of and help deliver the projects they use. With this knowledge I created a Gantt chart (SEE APPENDIX) to help them and myself plan what will be needed and when certain data or information will need to be ready by.

While having all the dates set for EA Technology will help them prepare the documents and data needed for myself to do the project. It also keeps myself on track knowing what needs to be done by when, for the programming aspect I want to use a template from extreme program (XP). XP offers a lot on how to deliver a project on time and while certain practices are not relevant to me like pair programming others is like iterative planning and constant testing.

Waterfall model is a sequential template where from the requirements you make the design etc. I can use this for my timings in the project, such as for my background reading, on to then needing the data to find what features to extract. Having this as an overall plan is generally what I am used to and fell most confortable with.

For the overall picture some criteria will have to be met/understood before I could move on for example there is no point in me having the data is I had no plan or idea of how to pick out the features. However once I am starting to program the Android app I know I estimate it will take a few weeks but being able to have iterations to show the customer and receive feedback will be invaluable.

This is where XP will come in using its iterative approach and constant testing practices, and allows for an ever-changing requirement specification.

In XP it uses stories created by the customer with the developers, to make use of this I am working off the points in the specification, for example:

* Recording, the engineer will need to be able to record and save the audio data either via the built in mic or plugging in a mic via the headphone jack.

Using this as a story I will be able to plan iterations of the software I am going to create, and get feedback at each stage. The main advantage for this is getting the user interface right for the end users, as my knowledge of the situations is limited having a feedback loop will be needed. I believe a hybrid of the two models will be a success.

# Design

## Overall Architecture

Android is the operating system the application will be built in. The main language used for Android apps is Java. Android is also capable if supports native C/C++ code compiling it for the operating system and using it as an external library.

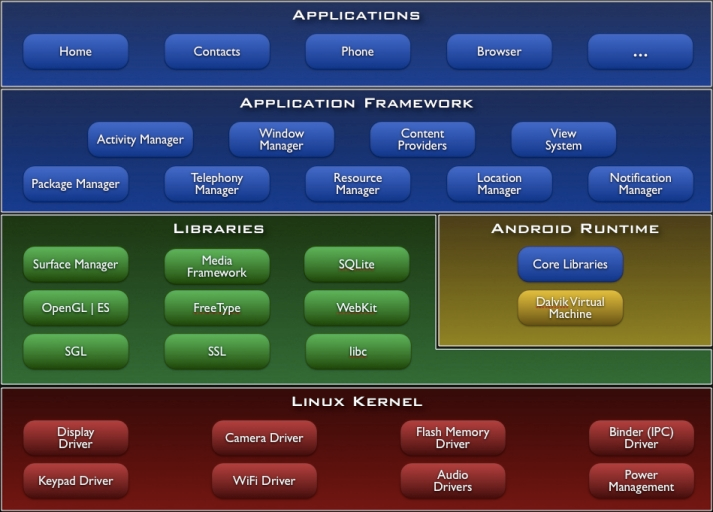


Figure -- [24]

Figure 1 is showing the overview of the Android operating system, the application I will create will be in the Applications section. The app will be using a lot of the Application framework and my own C library for the reading of WAV files. This library is compiled using the NDK and saved as part of the app but is loaded as a library from the Libraries section.

Google the creators of Android have created their own IDE call Android Studio, this is where I will be creating, building and debugging the app. Android Studio does not come with the native compiler and will be downloaded and deployed within the structure of the IDE so that it will be able to compile all code at once. There were other choices for IDE such as Eclipse or (ASK MADGE), I didn’t decide to use this IDE’s as I have had experience using Android Studio and knew how to quickly and efficiently create an app.

## Detailed Design

Detailed design is the in’s and outs of the system using unified mark-up language (UML) to present and describe the structural, behavioural and the user interface designs.

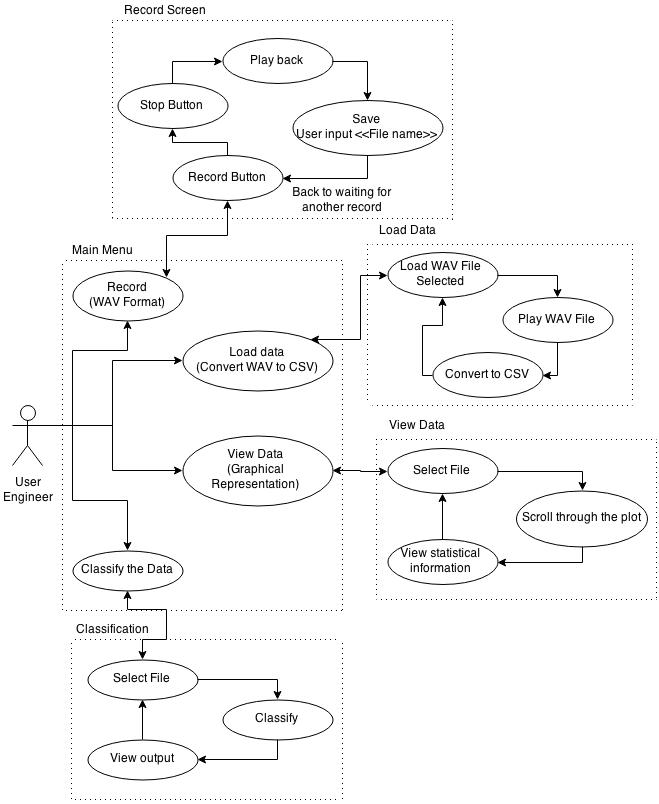
**Use Case Diagram** 

Figure -- Use case displaying how an engineer will be using the App

The use case above (Figure 2) is showing how an engineer will be using the App. The app will have 4 main sections, Record, Load/convert, view the data and classify, while the main aspect of the app is to classify the audio data coming in the other sections will be needed.

If the user wants to record he/she need to make sure the EA Instrument is plugged in and ready, once the recording has started and the user clicks stop they will need to input the file name. The location of the file is automatic as the new regulations in android means an app can only have access to its own folder. Once the file has been saved, the app will be waiting for the next record.

Load/convert the data, I have kept this separate as to show the different functions and for testing. This screen will search for the files in the app folder, allow the user to scroll through them, hear the recorded audio to make sure that it is the correct one and convert to a CSV format. The conversion will move and create the files to another location with in a new folder.

View Data, allows the user to view the converted data is a graphical format and see the statistical data created for each audio file. If the audio file has already been classified this will also be shown here.

Classify data will have the same select file system as the previous two screens and be able to process the data and return with an confirmation of it the audio file is showing signs of partial discharge or not.

### Structural View

The structural view describes the static structure of a system. The focus on this section is the structure not the behaviour of the application. Class diagrams on the main aspects of the system with a overview description on the passing of data and interactions of each class is what you will find here.

**Class Diagram**

See 6.2.1 Class Diagram for the full class diagram. These diagrams have been made via Visual-Paradigm see section 6.3.

The class diagram is illustrating a static view of the overall system, I have split the class diagram into two sections the main system and the features that get extracted. I did this as the main system is more the user interface and how it interacts with the background and the features are from jAudio, and have been altered to fit into the project.

The main system is depicting the user interface as each activity extends the action bar that is apart of Androids core system. This allows myself to have a menu bar for settings and other important options for the user. The class MainActivity is the first class to be opened when the application is started. This has two methods that are linked with the buttons with the corresponding xml file for the display. These two methods will call the two other screens, either FileChooser or Record.

FileChooser function will be where the user selects the audio file that has been saved; this will pass the selected file name as a String to the LoadData class.

The LoadData class having the files name to load will start a thread to load the data and to initialise a new AudioStruct. This is in a new thread as it will have to potential to take some time to complete and this also keeps the UI responsive for the user. The AudioStruct is passed to the classifier that holds all the known average values for both partial discharge and non-partial discharge. This is where will find the closest matching class for the new audio data and allow the user to get the information whenever needed.

Recording class allows the user to record an audio file; to do this it calls the ExtAudioRecording class see section 6.11 for more detail on this class. This class will take the input from the mic and save it to a temp file called ‘tmp.wav’, this allows the user to playback the recording before deciding if it needs to be saved

Once the audio has been recorded the user can call SaveAs, and this will as the user for the reference as a text input. The text input will have to be checked to stop the user entering anything harmful, in Android you can limit what characters are entered within the xml file this will be limited to the following characters – “abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789\_-”.

### Implementation View

**Component Diagram**

???

### Behavioural View

Behavioural view is how the application or system changes state from external influence. In this section you will find sequence diagrams showing the users interactions from starting the application to recording, saving and viewing and classifying the audio data.

**Sequence Diagram**

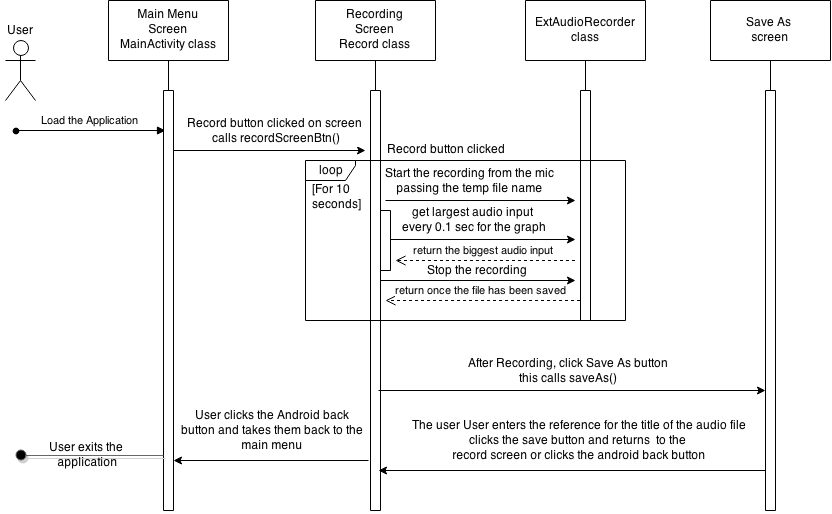


Figure - A sequence diagram showing the user recording new audio data

Figure 3 showing the sequence diagram for a user to start the application, capture new audio and save the file with a reference of their choosing. Once the user has started a recording there is a count down on the screen, also there is a feedback of the volume levels detected by the mic, so the user can see something is being recorded.

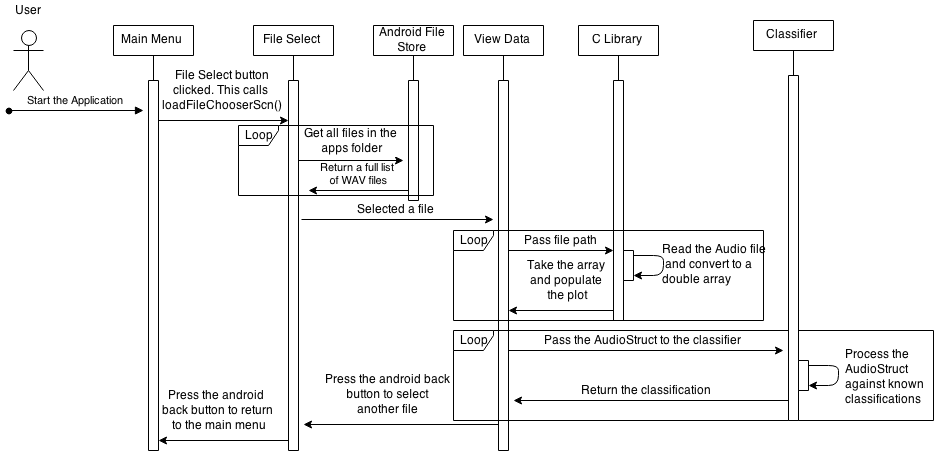


Figure - A sequence diagram of the user viewing the data and classification

Figure 4 showing the user selecting a file to view its data and classification. To be able to select an audio file it must loop through finding the audio files in the folder. If there are more than a screen full (This varies depending on the orientation) the user will be able to scroll through to select them. The selected file gets then passed to the C Library to read in and convert to a double array, all the information from the audio file is stored in an AudioStruct and passed to the classifier. The classifier holds already classified instances and finds the best match, to return the classification to show to the user.

For both sequence diagram Figure 3 and Figure 4 we use the default Android back-button because, in the Android best practises it states that the user should allow that the only back button is the device not and so there is no need in creating my own. This allows for a universal flow across all devices to keep with the users expectations.

## User Interface Design

**Mock Screen shots**

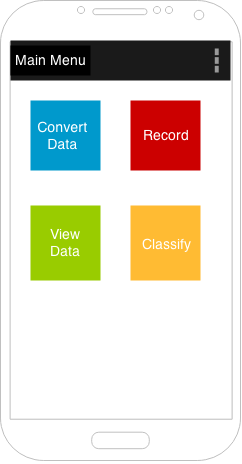


Figure - Mock screen shot of the main menu

Figure 5 showing the mock main menu of the app, had to be clean and simple as the end user will be an engineer will be in the field. The engineer will be in full health and safety gear one of which will be gloves this is the reasoning for the big clear colour coded buttons.

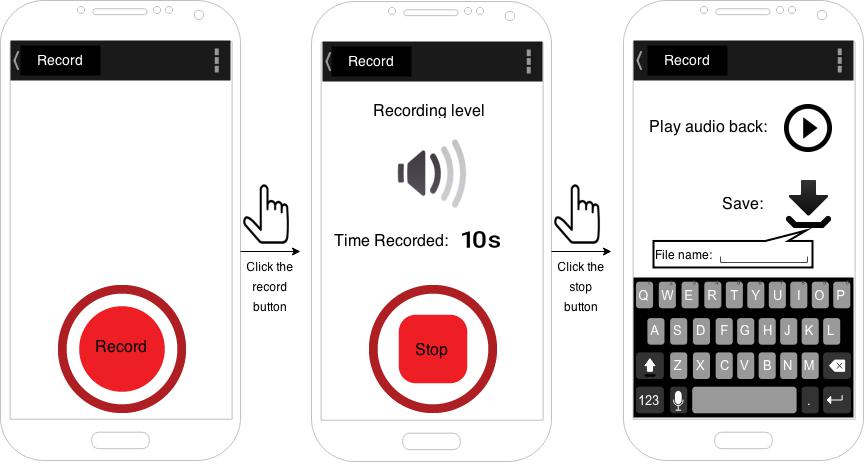


Figure - Mock screen of the recording process

Figure 6 shows the flow of screens to record the audio. The record button is as big as it can be on the android screen. By having the speaker once the user has clicked record it allows a feedback to the user of the volume level of the recording.

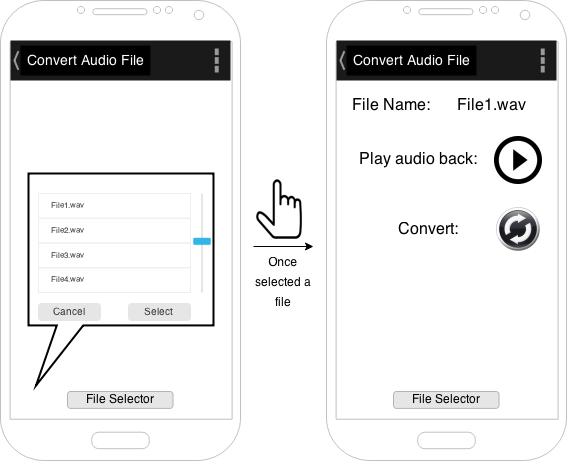


Figure – Mock Convert Audio file to data Screen

Figure 7 is showing how the user can convert the data from the audio WAV format to a CSV file. The file selector screen will be the same across all screens and will provide the user with the list of files save on the device.

Once the user has selected the file it takes them to the convert screen that allows the user to playback the audio file or converts it.

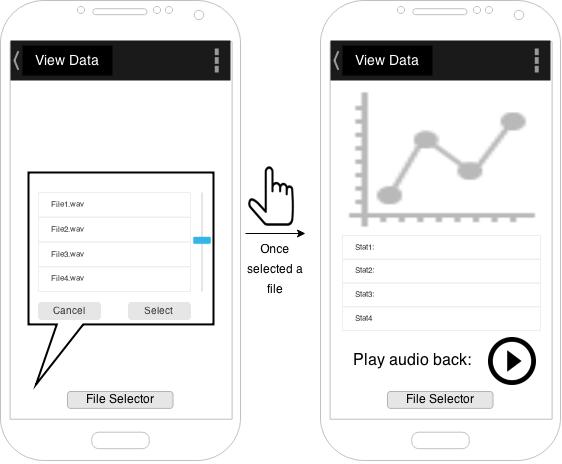


Figure - View data screen

Figure 8 is how to show the data once converted. The select file window will only show the converted data (file ending in .csv). It will show the plot of the data, the features extracted and allow the user to playback the audio and select a new file to load.

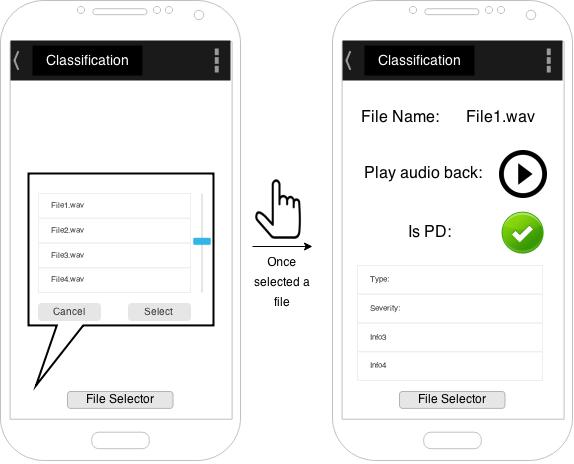


Figure - Classification screen

Figure 9 is the classification of the audio file selected. Again it will show the user the list of file that is available to classify,

## Other Relevant Sections

**Data storing**

The audio will be saved in the applications own folder created by the Android operating system. This is for security so no other applications can read anything outside its own folder.

SQLite is the database implemented inside the Android operating system and if I ever need to save any information other than the audio it will be saved with an SQLite database.

**Data structures**

When loading in an audio file there is a lot of information to be stored for each file and so I felt it best to keep all together in a structure.

The structure will hold all the audio data in a double array format, sample rate, window size for the features, file name and features extracted from its data. All of these variables will need getters and some will have setters. By doing this I have saved myself from having these as rouge variables across multiple classes. It brings all the relevant information into a single point of reference and encompasses the object orientation way of Java.

# Implementation

In this section will be describing how the final system was implemented and the issues encountered along the way. The approach for this section is a top down; it will show the final system, and the issues that arose and the spike work.

## Final Application

The final application came through many iterations and feedback from the customer. From the original design the final outcome on the user interface is very different and so was the classification.

The initial mock UI was received very warmly with the customer, although they had some feedback and some alterations to make. The Main Menu was spot on the buttons had to be clear well separated and large enough for a user with gloves to be able to use.

### Recording

The recording screen needed some alterations after some discussions on what type of feedback and how the interactions from start to finish for recording new data. The recording function saved the file after each ten second record prompting the user to enter the reference/ title of the file, after showing the customer this we came to the decision that the user may not want every recording saved. To overcome this situation the recording would be saved as a temporary file called ‘tmp.wav’ which allows the user to play back the audio and then decide if he/she wants to save it. Having the customer’s feedback and quick response to questions, allowed for this function to be concluded on time even with the difficulties encountered.

The feedback for the user on what is being recorded originally was a speaker with 3 levels of volume as the outcome depending on the volume level inputted. After working through some iterations for the user to chose from, we came to a decision that a plot of the incoming audio would be best rather than the speaker image. The main reasons for this was that the engineer could potentially also see Partial Discharge as well as listen to it. See section 6.7 for the final screen shots of the recording screens.

Recording an audio file one of the two components of the application has been fully implemented. Implementing this function was a lot harder than first thought while working off the planned Gantt chart; it was complete within the time frame set. Recording while very successful came across some difficulties, for example the format needed was Waveform Audio Format (WAV), this format can be saved in many different sample rates and have up to 4 channels. My knowledge on how to handle this within the Android framework was lacking and after some spike work I found a tutorial and an open source library in java to use. The library is called ExtAudioRecorder, it was very simple to use, as from the library I only needed the recording/save class.

ExtAudioRecorder class needed to be edited to fit in with the current project; the edits that were needed were only small and simple ones. For example the recording class gets passed in what file format to save the audio as, this currently for the scope of this project as we only require WAV however the function was only disabled and if needed all the code is still there ready to be used.

### Classification

Being delayed with the data until the 13th March because of the CDA was a major setback and having lost 7 weeks because of this, 5 of these weeks was planned to have access to the data. Considering this some of the planned work will not be started or even fully implemented. The first step was to prioritise what could be achieved in the remaining weeks.

While doing a spike run looking at how to read in the audio files and create some summarising statistical values, now having the data to test further the results were interesting. The statistic value Skewness of the audio files after doing a Fast Fourier Transform (FFT), which showed the frequency of the audio file. The outcome was that if noise was present then the FFT plot was be constant through out while if partial discharge was present then the plot would be larger down at the lower frequencies. See section [TODO] for the plots and descriptions. While these results looked promising each result was taking far to long to conclude.

With constant communication with my supervisor he had suggested that we try and find a feature extract tool to help progression, I found jAudio, which allowed myself to read in the audio files and select which features to extract. 9 features were chosen and the average and standard deviation for each was calculated, ending up with 18 features in total.

jAudio works by having a window size of the array I chose 512. The window size had to be a power of 2, this is because of the FFT, and if the final window did not match the required size it will fill it in with zeros.

The 9 features are:

1. Compactness, this works by comparing each window to its neighbours, if they match the lower the number. This is much like Skewness the original idea as it is a very good indicator of noise. It also takes the magnitude spectrum of the neighbouring windows to better evaluate the difference.
2. Root Means Squared (RMS) is the measure of the power of the signal. This can be very helpful finding the partial discharge over noise within the audio file.
3. Spectral Centroid is the centre of mass of the power spectrum. Perceptually, it has a robust connection with the impression of "brightness" of a sound.
4. Spectral roll off point is the fraction of bins in the power spectrum at which 85% of the power is at lower frequencies. It is a measure of the right-Skewness of the power spectrum.
5. Spectral Variability is a measure of the variance of a signal's magnitude spectrum. To get this value you first need to work out the standard deviation of the magnitude spectrum, this shows the distance a set of numbers are spread out.
6. Strongest frequency via FFT max is the strongest frequency component of a signal, in Hz, found via finding the FFT bin with the highest power. This needs the FFT Bin Frequency Labels and the power spectrum.
7. Strongest frequency via spectral centroid is the strongest frequency component of a signal, in Hz, found via finding the spectral centroid. It needs both the spectral centroid and magnitude spectrum.
8. Strongest frequency via zero crossings is the strongest frequency component of a signal, in Hz, found via the number of zero-crossings. This requires the zero crossing value of the window.
9. Zero Crossing is the number of times the waveform changed sign. It can be an indication of frequency as well as noisiness.

Some of these features require some extra information suck as the power spectrum or magnitude spectrum.

* FFT, it takes the window as its real array input as the imaginary array input will be null, as we only need to focus on the real values. The FFT feature also used a Hanning window will be applied to the real input.
* FFT bin frequencies, the bin label, in Hz, of each power spectrum or magnitude spectrum bin. While by its self not very useful but very useful when using the power or magnitude spectrum.
* Power Spectrum is found by first calculating the FFT with a Hanning window. The magnitude spectrum value for each bin is found by first summing the squares of the real and imaginary components. The number of bins divides the result.
* Magnitude spectrum is found by first calculating the FFT with a Hanning window. The magnitude spectrum value for each bin is found by first summing squares of the real and imaginary components. The square root of this is then found and the number of bins divides the result.

Each feature is extracted for each window within the WAV file; from this array we can calculate the average value and the standard deviation, this makes up the 18 features.

With these features now extracted we split the data up into 60% training data and 40% testing data, so that the classifier does not fully over fit to the data to the training data. The 60% training data will be split into Partial Discharge and non-Partial Discharge, the aim was to put the values into an excel spread sheet to plot a graph and see if the is any separation between them, see Figure 10.

Figure -- The Average of features of partial discharge against non-partial discharge

From my own inspection I could separate the values and classify new audio data. The most obvious feature to do this was the compactness average. To help further analyse this data I took an average of the individual features to get smooth out any out lying data, when comparing one of the testing audio files it was still clear if it was Partial Discharge or not. See Figure 11 below.

Figure -- plotting an unclassified audio file against the known classification features

With the time left in the project I did not have time to fully understand each of these features and how they work. The way I was able to implement them all was using the source code of the jAudio, and extracting the functions and classes needed. The program was very complex and took a while to fully understand how the code works and to best way to implement it into the Android application. The features all worked off an feature abstract class which all other features extended, this was very easy to add in and I kept it all separate within its own package jAudio as this code was not mine. I had to edit them, as I did not need to pass a double array of extra features instead a single array. I also had to overload some methods in the features to be able to integrate into the Android project. The hardest bit was trying to order the feature extraction so that I had the correct value already worked out for a feature that needs it. For example the Strongest Frequency Via Spectral Centroid feature depends on the Spectral Centroid and Power Spectrum but the Spectral Centroid requires the Power Spectrum, so there is no point working out the Strongest Frequency Via Spectral Centroid first.

The classifier works by the having the average value for all features from the test data for partial discharge and non-partial discharge. All these values have been hard coded into the classifier. With these values it we find the minimum distance to each new audio feature. Then totals each feature to either partial discharge or no-partial discharge, as we have 18 features this could be a tie. To overcome this we calculate the certainty of each feature we do this by:

Having the partial and non-partial discharge value, we start by comparing the distance they are apart, then the distance from the about to be classified feature. With these distances we can say if the new value is higher than both partial and non-partial discharge values then it is 100% certain of the, or if the new feature lies directly between both values the certainty is 0%. See section for the pseudo code of this algorithm.

## Spike work

In this section it will cover the early spike work needed to create the final Android application.

### Understanding Android 4.4+

My experience has been in Android 4.2, so understanding the differences in the newer 4.4 + is needed. One of the main differences is the storage framework, where the application can only access its own folder. I found this at first to be a set back, however being able to call the Android built in path method to the internal storage always return to the applications folder.

This meant that really nothing is different and it was only small change to the code.

### Loading WAV Audio files in Android

I started by loading the audio data into Scilab, and saving the arrays as a CSV file. I then used the CSV files to be read in the Android application. By doing this I could create the features and the basics for the application, as I will be using the data as a double array.

The next step was to try and read the WAV file in Java to convert it to a double array. This was a lot harder than I would of expected it to be, playing the audio file on the device was very simple as Android already has the media player. After a meeting with EA Technology they use a C library to read in the WAV files called Libsndfile, this meant I had to add C support into the android app.

Following some simple tutorials I managed to get a very basic application working, this just passed information backwards and forwards between C and Java. It worked by making an extra folder called jni and building/compiling the code within jni with the NDK. To compile the C code I had to create a make file and run this in a terminal, the debugging feedback was very limited in this method as I found many times. Sometime the C code would compile and then the application would just stop running and give no indication on how it failed, this did slow me down. The make file called Android.mk holds the library name, the files needed to compile and a list of dependencies needed to create a library for the Java code al call. When building the native code there is a function that will build the C header file that the Java code calls, this meant that creating the functions was simple and all I have to do was call the right libsndfile functions from the created functions to run the code.

Libsndfile had a very large community and luckily I was not the first to try and use this library on Android. I was sent a link to another developer, who had put up their project code to use as an example of this, was released under the GNU lesser General Public License. Following what I had learnt from the other spike work into the using this library for reading the audio files in I found using the same read function worked.

Upon further investigation into if I needed to convert the data to help classify or manage the audio files better, I found that I did not need to. This was because reading in a CSV file with the array of values from the audio file took longer to process into doubles as this is first read in as a String type then converted to a double. Rather just read in the WAV data, which is binary, and convert the binary data straight to a double. Doing it this way sped up the loading of the file by 10x and meant classifying the data was possible on each load.

After this I will only need two options from the main menu screen, ‘Record‘ and ‘Classify’. This will cut half the program as I no long have to convert, save and read CSV files. However I still have all the code in case the this feature is needed and we can export more than just the audio as an array, such as the classification and other captured information.

### Feature extraction

EA Technology was able to give me one audio file and inform me this was partial discharge that was recorded. With this file it was possible to use to test some of the theory learnt up to now. The file had very prominent pattern in the audio see Figure 12, from this pattern I was able to count each spike, calculate its distance they are apart, and the duration of the full spike. Average, variance and Skewness were also calculated but with nothing to compare against so there was nothing to find from these results

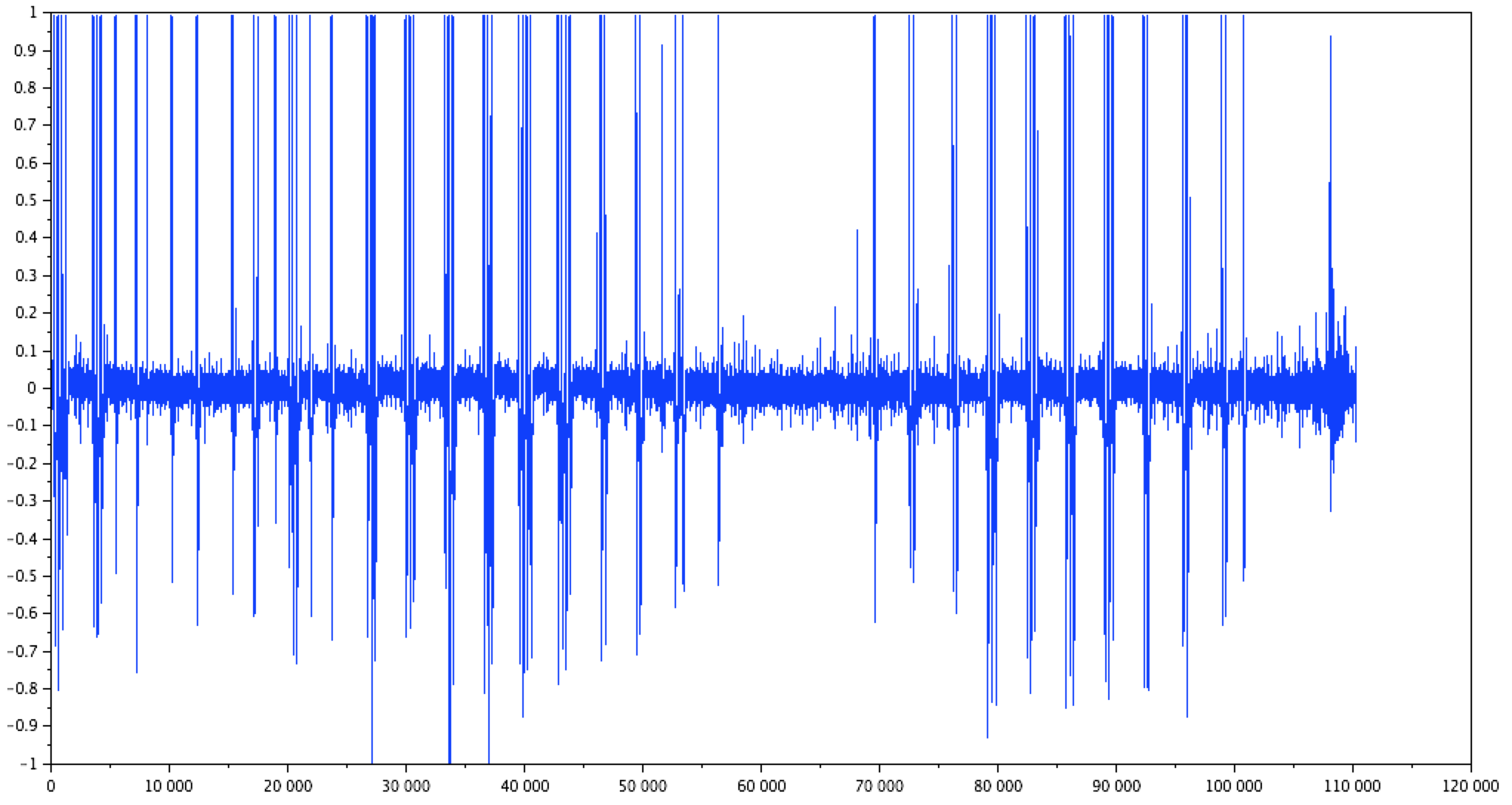


Figure - Plot of the test.wav file

This functionality was very delayed because of the time it took the CDA to be signed. EA Technology provided myself with 20 files to train and test the classifier; each one of these audio files had been listened to, classified and describes what they hear by 4 domain experts.

From the 20 files provided by EA Technology I picked out a fair split for the training data of both partial discharge and non-partial discharge. Having the training data I ran them all through the same code I had at the start to see if any were linearly separable, sadly taking the whole audio file and working out these features showed nothing up. To take this one step further I tried the same features after doing an FFT on the audio files, as this will show the frequency response and help separate partial discharge from noise.

Scilab is an open source program for numerical computation providing a powerful computing environment for engineering and scientific applications. Scilab has many built in functions to help process the audio data as it can read in in the audio WAV files run an FFT and display the results. Scilab allowed myself to be able to quickly test some of my ideas rather than learning how to program these features in C or Java. The results from this experiment see section 6.4 shows that the FFT can show the differences from partial to non-partial discharge. But from running the same features did not show the expected results, as Skewness did not separate the data apart.

The work done on the first audio file given and the features that I found from the pattern shown had no relation with the new data provided. This meant that the majority of the spike work is not needed and a waste.

## Issues that arose

Having not been able to make any real headway with my own investigations into extracting features, using a feature extraction tool made this a lot quicker to get results. Finding a program to do this was a lot harder than I would have imagined, this was not because of choice, it was finding the right program that can offer the options required.

An example extract tool used I looked at YAAFE (Yet Another Audio Feature Extractor) this is a C++ library with many dependencies, one of which is the libsndfile library for the reading of the audio files. A full list of the feature YAAFE can output is in section 6.12, when installing the program I came across many problems with the dependencies needed and when researching into these it was clear that some of the feature would not work if I ported this project across to Android.

I had many minor issues with the code and Stack Overflow was critical in over coming these. For example when searching a folder in Android for the WAV files I came across an example on Stack Overflow, which did everything I needed. See section 6.13. Stack Overflow was also used for further understanding into the programming language Java and how it handles audio, as I was not the first to try and do this and there were many opinions on how to best handle this. It was also very hard to find the correct answer a lot of the time and a lot of time was wasted in this manner, but with out that time the final solution would of not of came about.

A set back was the CDA being signed for the information and data from EA Technology and Aberystwyth University. I understood that these agreements could take time and I had given myself extra time when planning this in the Gantt chart. Conversely taking an extra 5 weeks then planned was a real set back, and the project was very close to being changed because of this reason. With the delay it means that some of the functions needed in the application will not be as polished as possible or even complete.

Having some experience in creating an Android application it meant when planning this I knew I could complete this task quickly. From research and the design add native support to a application was not part of my previous experience and learning this skill would take time. Making a simple app using the native support was very simple however adding in the library and the dependencies needed provided a challenge. The only debugging in formation available was limited because we build inside the terminal and I found the build will complete but when running the application it would close with a null pointer. Having less feedback was a hindrance and not fully understanding the way the code worked it took me longer than normal to overcome errors.

# Testing

Testing is an integral part of any software program, Google know this and have many features in place. The Android framework offers JUnit, instrumentation, test case and assertion testing to fully make sure the application will not fail, stop or behave unexpectedly.

Time was very tight towards the end of the project but using the extreme programming approach and creating the tests when creating the java class’s saved a lot of time. The main part of this project is made using Java, this means I will be able to use JUnit testing, JUnit allows to test the logic of the algorithms, the return values expected from the third party code and the user interface components.

Using C code in my project meant it came with its own test functions and was very simple to run them in the terminal. To test my own C code I was able to use JUnit, as my code was a broker to the C library and by testing the calls to the library, and the return values meant it ran as I expected.

## Overall Approach to Testing

## Automated Testing

### Unit Tests

My approach for this was to create the tests along with the java code. This meant I could test the code just written, while fresh in the mind and for later in case of being affected by a refactor. The IDE Android Studio allows the test classes to be made automatically with all the methods having the corresponding tests.

When creating a test class it can extend many different way of testing, for example we can test the UI variables by extending InstrumentationTestCase or the general logic by extending AndroidTestCase.

I split up the tests created for the features and the rest of the system; this was done as both these parts were in separate packages. The tests are run on the Samsung S4, the device the application has been developed on.

Feature testing set up for every test was to create a double array 80,000 in size, its 80,000 as this is exactly 10 seconds of a WAV sample at 8000 sample rate. Each array is populated with the value 1.0; this will allow the features to be easily calculated and to check the correct return value. While this does not test real values expected I was able to create the answers in Scilab and confirm the answers with assertEqual() methods.

Some features that require extra information for example Compactness, needs the magnitude spectrum I pass the same array. Other features return an array of the processed data, like in the FFTBinFrequenciesTest class. Within the test method I check both the first value in the array as I feel if the first value is correct then the rest will be and the array length. To test this feature if the array is only 1 I pass in a new array with length of 1 and test the output. All features created by jAudio throw exceptions and in JUnit 3.0 I am not able to test the exceptions so when passing a null array we get an exception from the feature.

### User Interface Testing

I was able to create some user interface unit tests; these just make sure that the local variables are assigned and not equal to null. By doing this I can assure that whichever screen the user is on all the variable and listeners are created.

I had found some problems when trying to do this, as my Activities extend ActionBarActivity to allow the app to use a pop up menu this meant in unit testing the user interface the onCreate method happened before the theme was set. This returned an ‘*IllegalStateException*’, and the explanation was

*“You need to use a Theme.AppCompat theme (or descendant) with this activity”*

After much searching for a solution, I came across 2. The first was to take the class I was extending ActionBarActivity and change it to just Activity, by doing this it took away the action bar theme and options and set up as a blank activity to then inflate the XML file. I did not decide on doing this as the action bar was necessary on all screens and is the title as well as how to get to the setting screen. However when changing to just Activity, running the tests did succeed and pass.

The second solution was simple and almost elegant, as all that was missing was the theme being set before onCreate. In JUnit testing we can build a ContextThemeWrapper in this we can set the theme to the context; we do this before we set and start the Activity. This stops the ‘*IllegalStateException*’ and allows the testing to run like normal. Though by setting the theme within the test, if it main theme is ever changed we need to remember to update the test to match the change in the AndroidManifest.

### Stress Testing

To stress test the application in JUnit test we set the full array size to 80,000. While this is just 10 seconds of recording at 8KHz sample rate the features only get the window size of this (default is 512), by passing the full array we give each feature its full capacity.

To best stress test it would be wise to test this on multiple devices. Unfortunately I do not have access to any other device other than the Samsung S4, however Android Studio allows you to create a virtual Android device and set the hardware specification to your own requirements.

I set up multiple virtual devices and ran the test code on them. I di not have any issues it ran very slowly in the lowest hardware specification device but I used the minimum requirements to run the Android operating system 4.4.

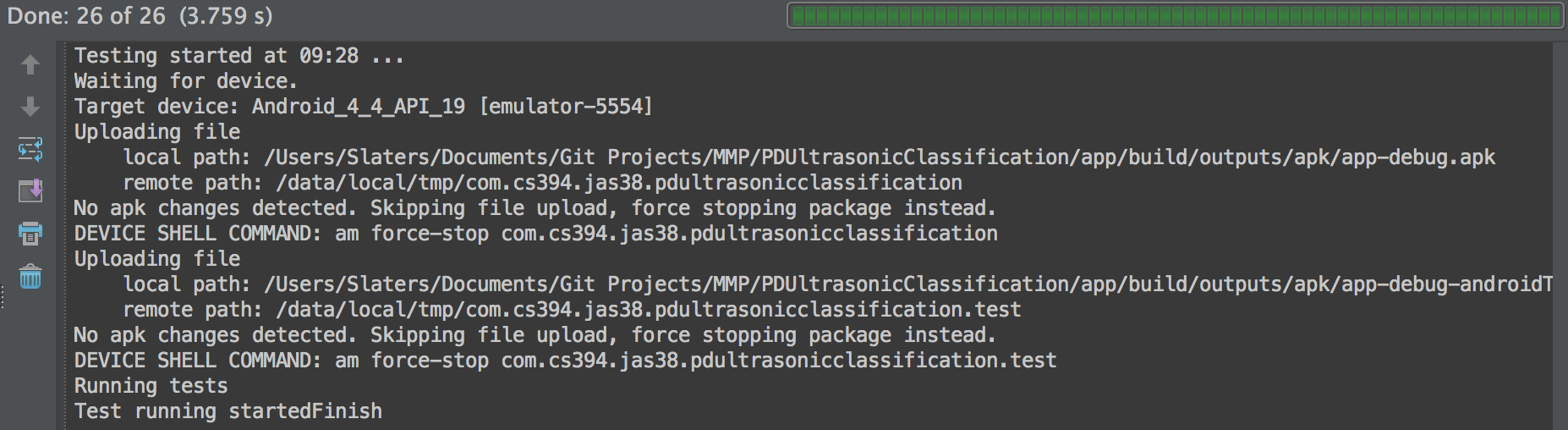


Figure - Android 4.4 API 19 Emulator

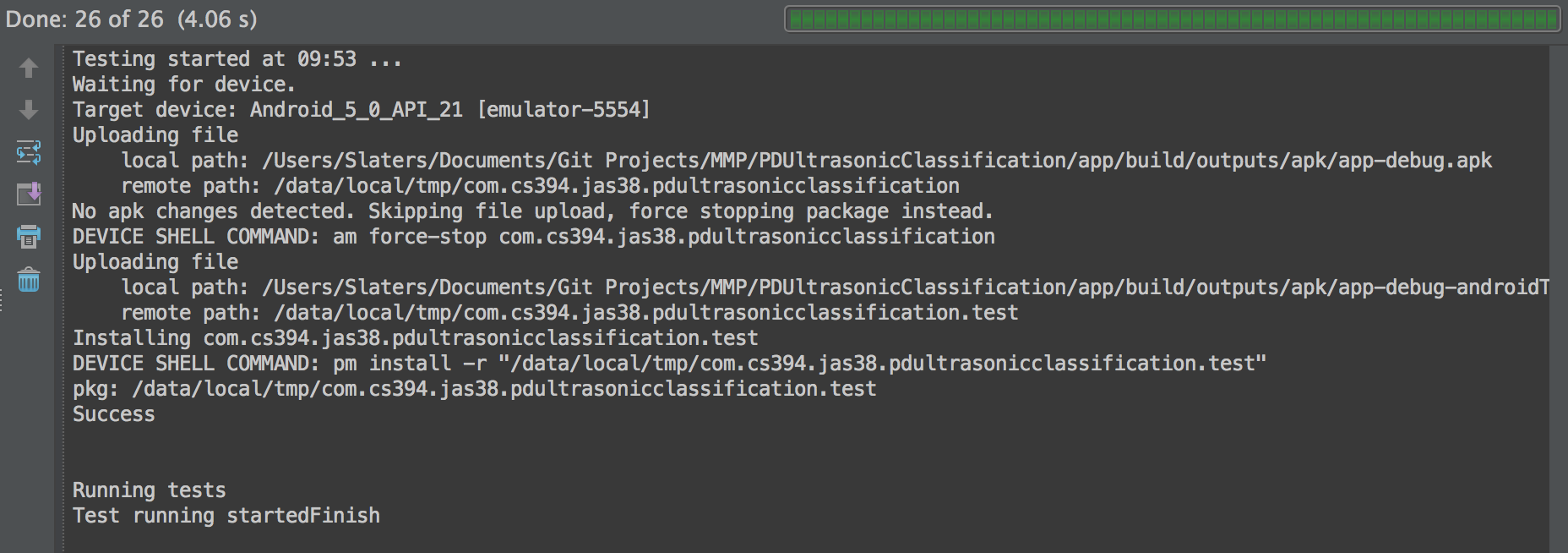


Figure -- Android 5.0 API 21

## Integration Testing

After refactors running the unit tests

Shows the whole system working

Test table here

## User Testing

Tho no user has done this

Test Table

Have you tested your system on ’real users’? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

# Critical Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

* Were the requirements correctly identified?
* Were the design decisions correct?
* Could a more suitable set of tools have been chosen?
* How well did the software meet the needs of those who were expecting to use it?
* How well were any other project aims achieved?
* If you were starting again, what would you do differently?

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honors degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

Review the discussion on the Evaluation section from the lectures. A recording is available on Blackboard.

# Appendices

* 1. Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

## Project Specification

For the data, I will need to decide which features are important and decide how they will impact on the required functionality of the App. As stated by Sahoo and Salama, “The first step in any recognition process is to consider the problem of what discriminatory features to select and how to extract these features from the patterns.”[4] I will use this as my starting point. In order to do this, I will need to process the data into a more relevant and more manageable form, I will need to do some additional research in order to fully understand how to accomplish this in the most straightforward manner.

In order to design a compact classifier it will be best to use data mining tools (e.g. WEKA) to extract rules based on the training data. This rule base could then hard-coded in the App and used to classify new data.

### Android App

* The Android app will work on Android 4.4+ upwards
* It will be able to record audio given
* Javadoc *“Javadoc is a tool for generating API documentation in HTML format from doc comments in source code”.*
* Working executable *APK files are a type of archive file, specifically in zip format packages based on the JAR file format, with .apk as the filename extension.*
* Testing *“The Android testing framework, an integral part of the development environment, provides an architecture and powerful tools that help you test every aspect of your application at every level from unit to framework.”*
* Test Tables for UI While I can use some testing already built into the Android API and testing tools there will be some aspects of the Application that cannot be tested and will need a table.

## UML Diagrams

### Class Diagrams

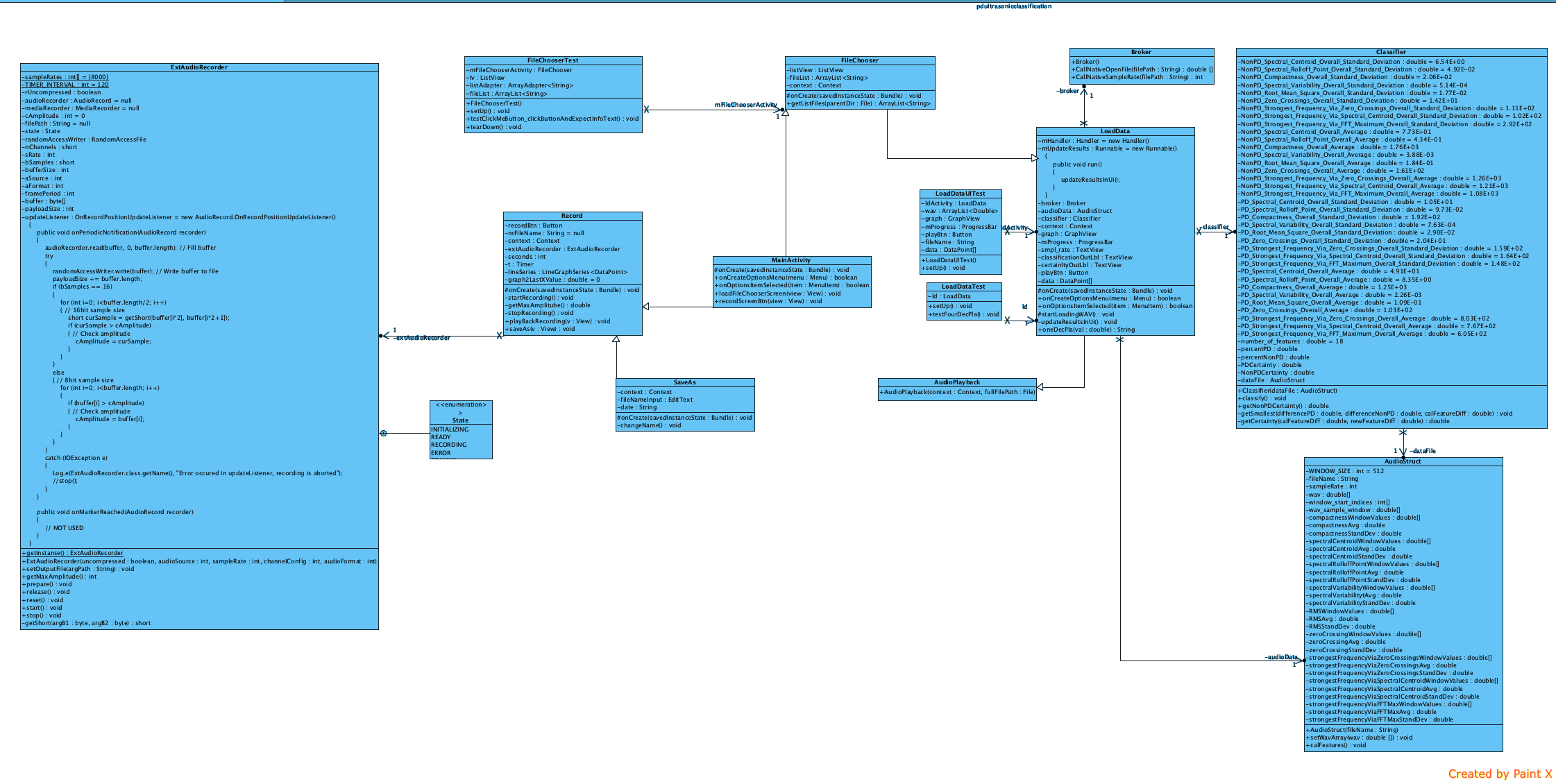
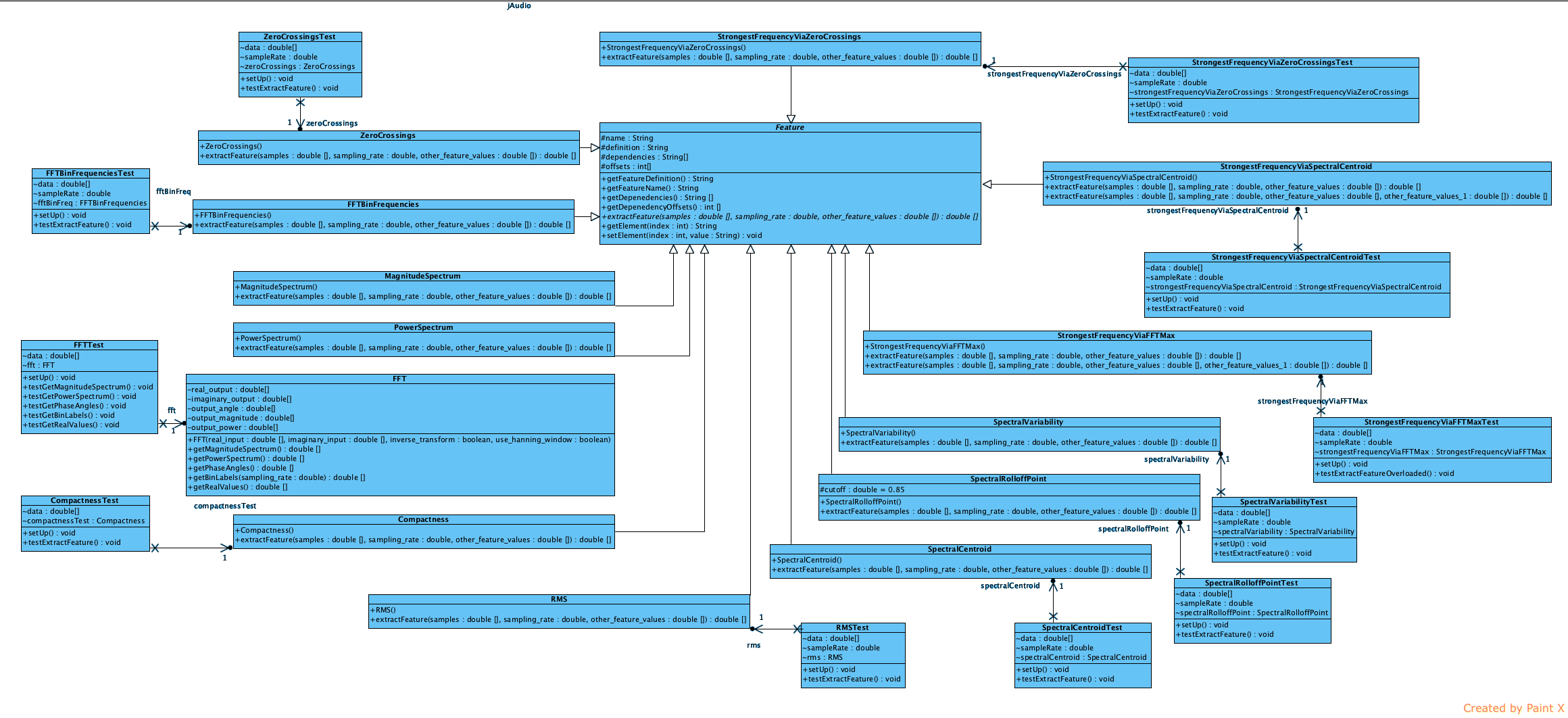


Figure - Class diagram of the Android application

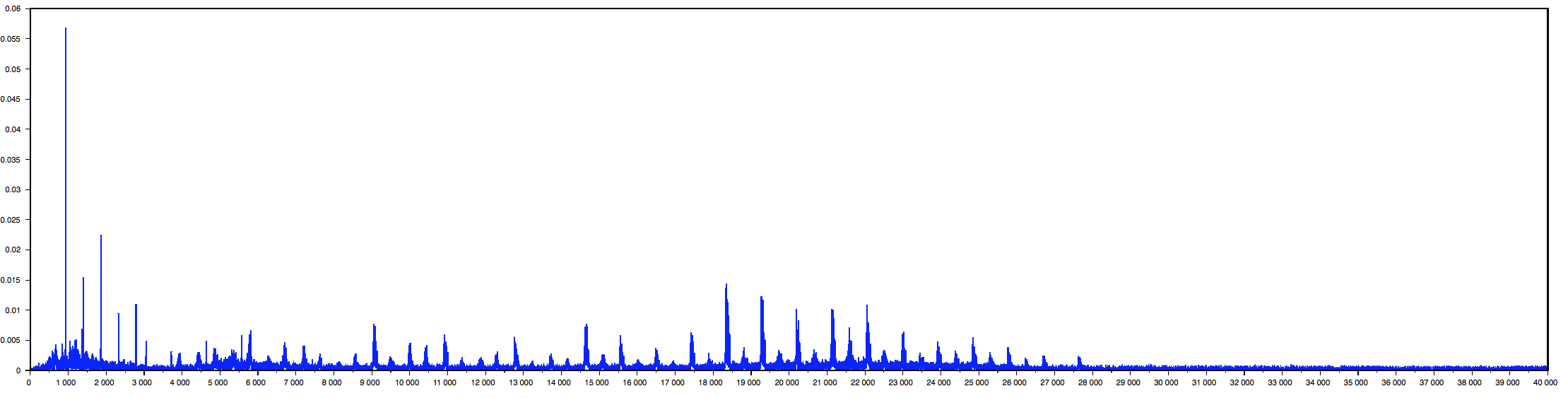


## UML Creator Visual-Paradigm

### [http://www.visual-paradigm.com/](http://www.visual-paradigm.com/sdeij.php)

## Feature Spike work

### NON Partial Discharge

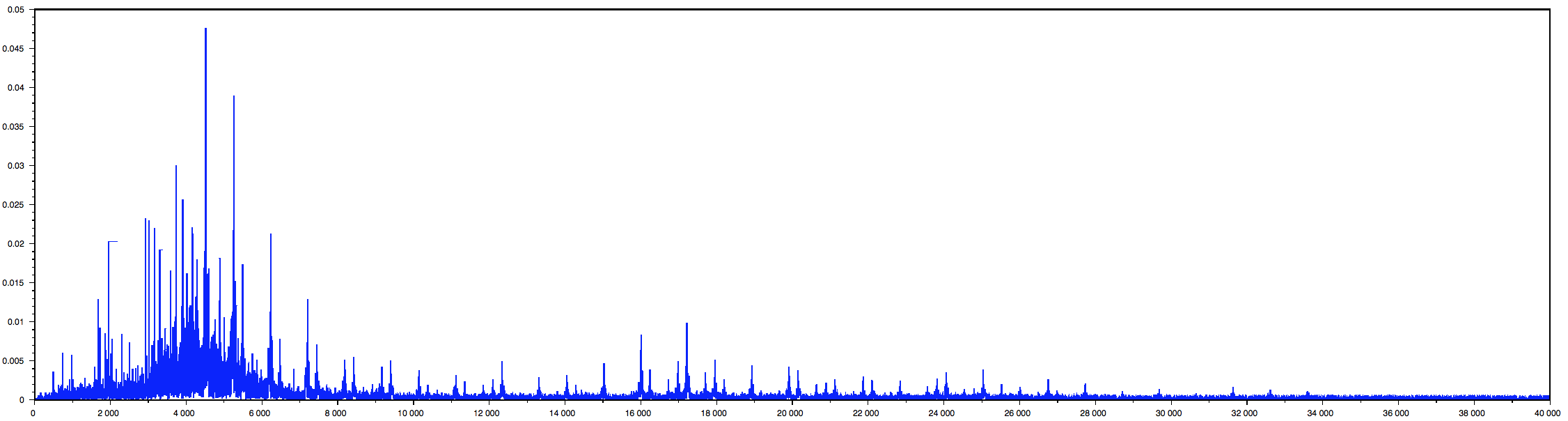
**Ref\_05\_WSP\_Util\_swbd\_feeder 1**

Notes - Very regular, probably not PD

Skewness = 10.929229

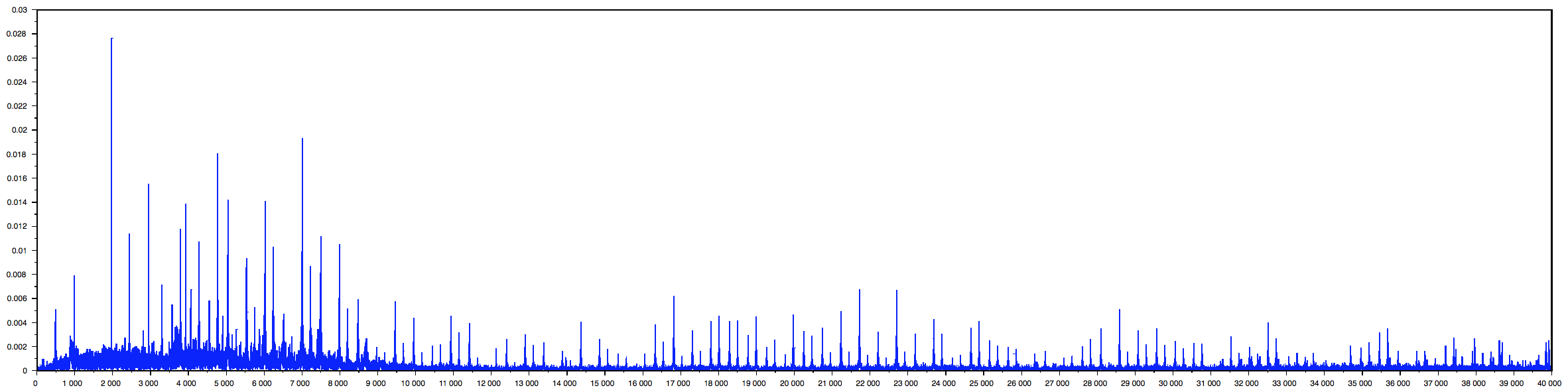
Kurtosis = 451.34994

**Ref\_01\_Sub\_1\_TR\_61\_CD\_1**



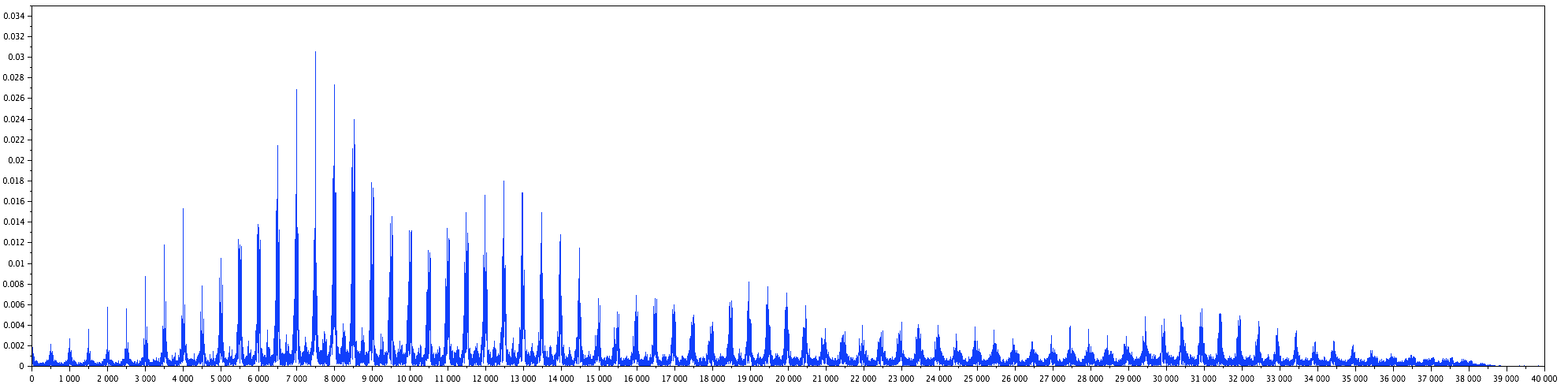
Notes - Very repetitive.

**Ref\_07 Sub 9 42P-371**



Notes - Too regular

**Ref\_11 Central \_ Tx No. Tx01 CHP Building**



Notes - Too regular

Skewness = 5.0396758

Kurtosis = 39.860225

**Conclusion of information**

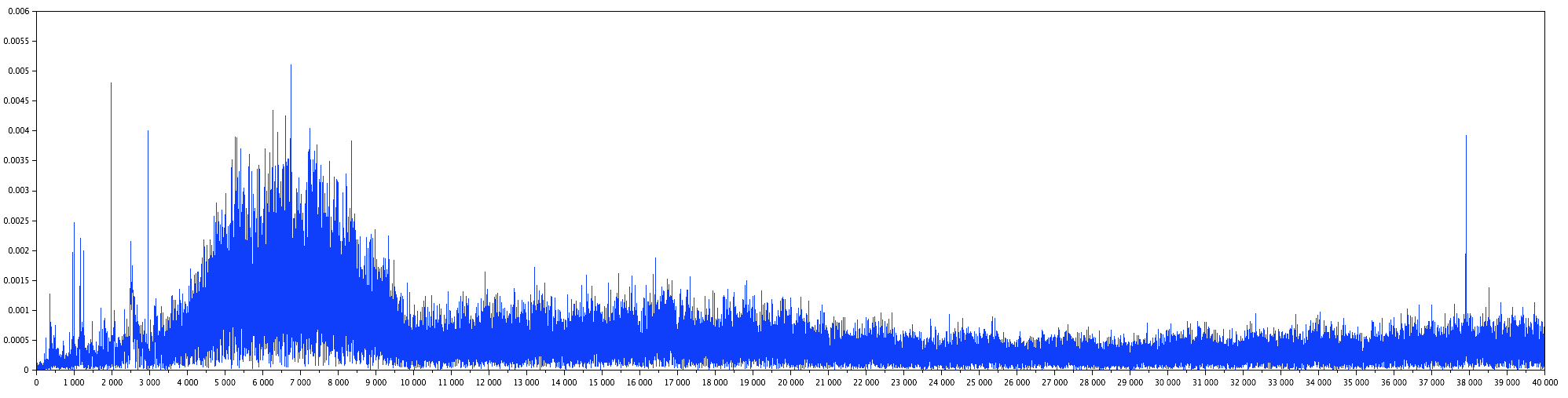
The plots are of the wav form after FFT and only showing the absolute values.

Find a way to find small peaks and if they are regular and full defiantly not PD.

As this is signs of Transformer vibrations/noise

### Partial Discharge

**Ref\_09\_sub9rec2**

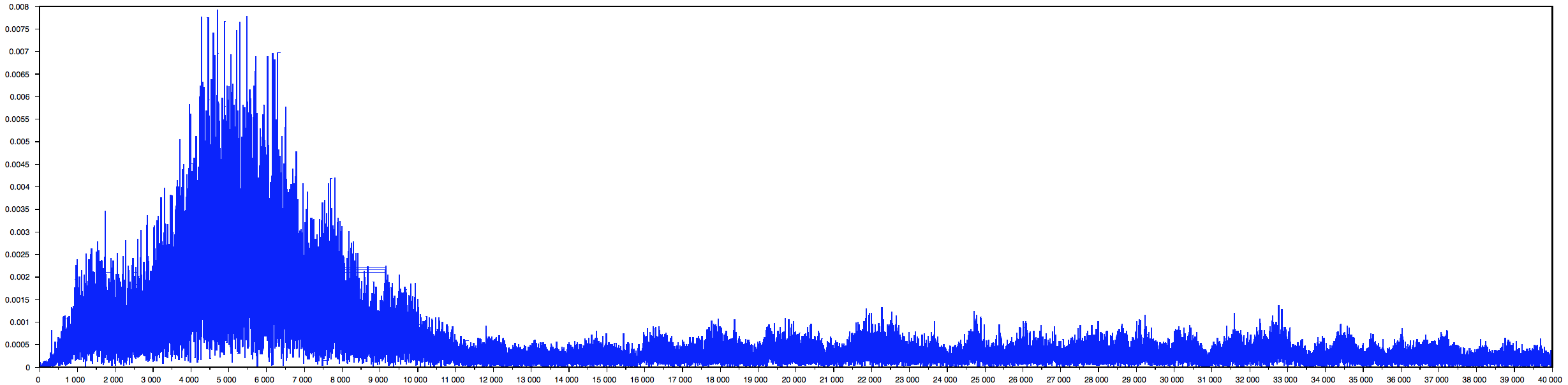


Notes - the spikiness of the noise, and it's dropout

Skewness = 2.7725939

Kurtosis = 13.791916

**Ref\_10 dss k (2)**

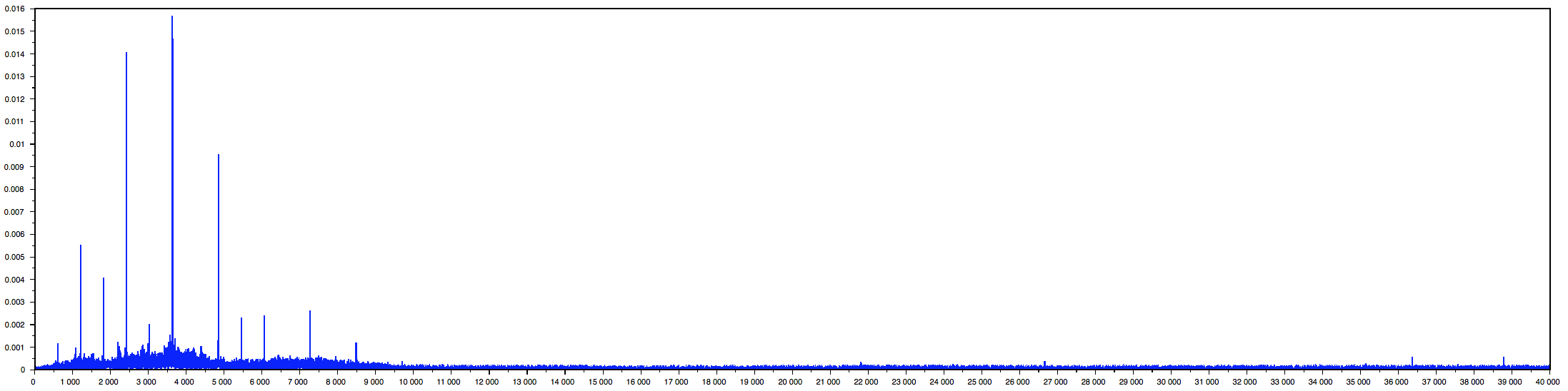


Notes - Characteristic crackle, wind noise ambient adds complication for analysis

Skewness = 3.3425825

Kurtosis = 16.76697

**Ref\_16 Cherokee lhs aux**

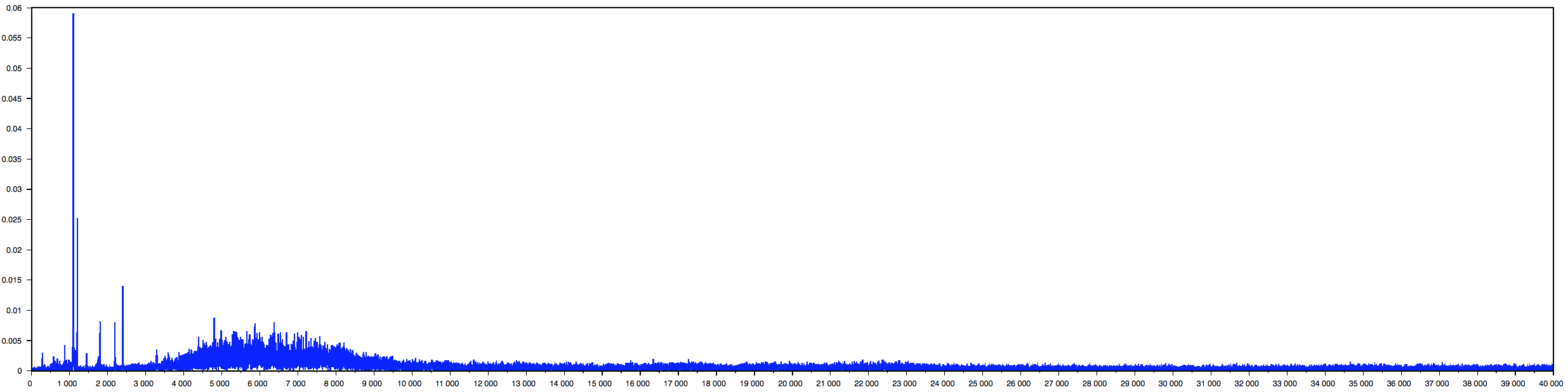


Notes - Crackling noise in background - Possibly PD

Skewness = 38.799082

Kurtosis = 2612.5601

**Ref\_17 emerson**



Notes - Something like across a too small air gap. Vicious sawing noise

Skewness = 16.444505

Kurtosis = 810.7896

**Conclusion of information**

All seem very one sided, still a lot of noise in some of them

Skewness could be a very good factor to determine against vibration noise

Many classical statistical tests and intervals depend on normality assumptions. Significant skewness and kurtosis clearly indicate that data are not normal.

## jAudio Features Analysis

## Testing Tables

Setup:

* Have an android device running Android 4.4+
* Have the app installed on the android device.
* Load over the test audio file into the apps folder on the internal storage at:
  + “/storage/emulated/0/Android/data/com.cs294.jas38.pdultrasonicclassification/files”

### Load the application.

Setup:

* Find the Application with the Apps section.

|  |  |  |
| --- | --- | --- |
| **Test** | **Successful** | **Comments/ Outcome** |
| Is the Apps name “PD Ultrasonic Classification” | Yes |  |
| Once clicked on the icon, does the app load. | Yes | The main menu screen is loaded. |
| Is the Icon the EA Technology Android logo. | No | It is the default Android icon |

### Main Menu Screen

Setup:

* Orientation - portrait

|  |  |  |
| --- | --- | --- |
| **Test** | **Successful** | **Comments/ Outcome** |
| Is the title “Main Menu” | Yes |  |
| “File Chooser” button is present | Yes | Big clear and green |
| “Record” button is present | Yes | Big clear and red |

Setup:

* Orientation - landscape

|  |  |  |
| --- | --- | --- |
| **Test** | **Successful** | **Comments/ Outcome** |
| Is the title “Main Menu” | Yes |  |
| “File Chooser” button is present | Yes | Big clear and green |
| “Record” button is present | Yes | Big clear and red |

|  |  |  |
| --- | --- | --- |
| **Test** | **Successful** | **Comments/ Outcome** |
| “File Chooser” button is clicked | Yes | A new screen is loaded. A list of files appears. |
| “Record” button is clicked | Yes | The “Record” screen in loaded |
| Click the android back button. | Yes | It takes the screen back to the devices home screen. |

### Record screen

Setup:

* Click on the “Record” button from the home screen.
* Orientation – portrait

|  |  |  |
| --- | --- | --- |
| **Test** | **Successful** | **Comments/ Outcome** |
| The title says “Record” | Yes |  |
| Has label “Count down: ” | Yes |  |
| Has number of 10 next to it | Yes |  |
| Under the label and count is a graph with a title of “Volume Input” | Yes | Title is present however the graph does not show until record starts. |
| A playback button is present | Yes |  |
| A Save as… button is present | Yes |  |
| A record button is present | Yes |  |

Setup:

* Orientation – landscape

|  |  |  |
| --- | --- | --- |
| **Test** | **Successful** | **Comments/ Outcome** |
| The title says “Record” | Yes |  |
| Has label “Count down: ” | Yes |  |
| Has number of 10 next to it | Yes |  |
| Under the label and count is a graph with a title of “Volume Input” | Yes | Title is present however the graph does not show until record starts. |
| A playback button is present | Yes |  |
| A Save as… button is present | Yes |  |
| A record button is present | Yes |  |

|  |  |  |
| --- | --- | --- |
| **Test** | **Successful** | **Comments/ Outcome** |
| Click the button record | Yes | The count down value counts down in 0.1 increments.  The graph shows the sound volume input over time.  After the count hits zero and the graph stops moving along and the count resets. |
| Click on Play back button | Yes | The Audio is played. |
| Click on the Save as button | Yes | Loads the Save as screen. |

### Save As Screens

Setup:

* Make sure there is a recorded audio tmp file.

|  |  |  |
| --- | --- | --- |
| **Test** | **Successful** | **Comments/ Outcome** |
| Reference label is present | Yes |  |
| Edit text input is present | Yes |  |
| Date label is present | Yes |  |
| The current date is shown in the format YYYY\_MM\_DD | Yes |  |
| Save button is present | Yes |  |
| Enter txt in the edit text aspect, test max length is 15 characters | Yes | Limits text to 15 characters |
| Only [a-z A-Z 0-9 \_-] characters can be used | Yes | Can press other characters however nothing is inputted |
| Enter “Testing” into the edit text and click the save button | Yes | Takes you back to the Record screen and a pop up saying saved file “Testing”. |
| Check the file store and see if the file has been saved Testing\_YYYY\_MM\_DD.wav | Yes |  |

### File Chooser Screen

Setup:

* Load the app and click the File Chooser button
* Have the test.wav file loaded within the apps folder or record some audio within the record screen.

|  |  |  |
| --- | --- | --- |
| **Test** | **Successful** | **Comments/ Outcome** |
| Can see a list of files on screen | Yes |  |
| Test.wav file is an option | Yes |  |
| If you change orientation all the same files exist | Yes |  |
| There is a scroll bar to the right, while this may not move anything if the user has many files it will be needed | Yes | It does nothing with only one .wav file in the folder. |
| Each file shown is clickable | Yes | Clicked on test.wav file and it loads View data screen. |
| Click the Android back button | Yes | Takes me back to the main menu. |
| After clicking the test.wav file and the new screen loads click the Android back button. | Yes | Takes me back to the list of files. |

### View Data screen

Setup:

* From the main menu click File chooser, click the test.wav file
* Make sure the volume is turned up to an audible level
* Have in portrait

|  |  |  |
| --- | --- | --- |
| **Test** | **Successful** | **Comments/ Outcome** |
| Title is View Data: test.wav | Yes | The title is the file that was clicked from the previous screen |
| There is a plot present of the audio file | Yes |  |
| The plot y-axis = 1 to -1 | Yes |  |
| The plots x-axis is from 0 to 250 | Yes |  |
| Move your thumb from right to left along the plot. Does the X-axis move along? | Yes | Very responsive. |
| Pinch together over the plot and the X scale will change | Yes | It went from 0-250 to 0-2000. Though the load time of this was very high and after the moving of the plot was very delayed. |
| Under the plot there is a label called Sample rate | Yes |  |
| To the right of Sample rate is the number that is the sample rate it should be one of these values: {44100, 22050, 11025, 8000}; | Yes | The value was 8000 |
| A label situated under the sample rate called “Classification”, the label will be in larger text | Yes |  |
| The value next to the Classification label will be the classification of the Audio file, either PD or Non-PD. For the ‘test.wav file’ the classification will be PD | Yes |  |
| Each Classification comes with a certainty percentage, the one present for the test.wav file will be 95% | Yes |  |
| Button called ‘Play Audio File’ is present? | Yes |  |
| Click on the button ‘Play Audio File’, does the file play? | Yes |  |
| When tilting to a landscape orientation, all labels, buttons and plot still present with the same values? | Yes |  |

## Final Screen Shots

* 1. Code Samples

This is an example appendix. Include as many appendices as you need. The appendices do not count towards the overall word count for the report.

## Graph View

The project is a library for Android to programmatically create  
flexible diagrams. Version 4.0.0 is the latest instalment created by Jonas Gehring, is open source under the GNU general public license, version 2, June 1991. The GNU license see [5]. Full website for more information see [4]. I used the .jar file and incorporated into the final Android app.

## jAudio 2.0

jAudio is a software package for extracting features from audio files as well as for iteratively developing and sharing new features. These extracted features can then be used in many areas of music information retrieval (MIR) research, often via processing with machine learning frameworks. Authors Daniel McEnnis and Cory McKay, I used jAudio 2.0 to do some feature extraction from the data provided. TODO – add more information on how and where I used this software and on licensing

## Libsndfile

Libsndfile is a C library for reading and writing files containing sampled sound (such as MS Windows WAV and the Apple/SGI AIFF format) through one standard library interface. It is released in source code format under the [Gnu Lesser General Public License](http://www.gnu.org/copyleft/lesser.html). I used this library to be able to read wav audio files on the android application; all the code was copied in to the jni folder.

Libsndfile has the following main features:

* Ability to read and write a large number of file formats.
* A simple, elegant and easy to use Applications Programming Interface.
* Usable on Unix, Win32, MacOS and others.
* On the fly format conversion, including endian-ness swapping, type conversion and bit width scaling.
* Optional normalisation when reading floating-point data from files containing integer data.
* Ability to open files in read/write mode.
* The ability to write the file header without closing the file (only on files open for write or read/write).
* Ability to query the library about all supported formats and retrieve text strings describing each format.

The code used was from <https://github.com/michaelwu/libsndfile>

## Recording WAV format audio in Java

<http://i-liger.com/article/android-wav-audio-recording>

## YAAFE

*YAAFE, an Easy to Use and Efficient Audio Feature Extraction Software*, B.Mathieu, S.Essid, T.Fillon, J.Prado, G.Richard, proceedings of the 11th ISMIR conference, Utrecht, Netherlands, 2010.

Available features:

- AmplitudeModulation

- AutoCorrelation

- ComplexDomainOnsetDetection

- Energy

- Envelope

- EnvelopeShapeStatistics

- Frames

- LPC

- LSF

- Loudness

- MFCC

- MagnitudeSpectrum

- OBSI

- OBSIR

- PerceptualSharpness

- PerceptualSpread

- SpectralCrestFactorPerBand

- SpectralDecrease

- SpectralFlatness

- SpectralFlatnessPerBand

- SpectralFlux

- SpectralRolloff

- SpectralShapeStatistics

- SpectralSlope

- SpectralVariation

- TemporalShapeStatistics

- ZCR

Available feature transforms:

- AutoCorrelationPeaksIntegrator

- Cepstrum

- Derivate

- HistogramIntegrator

- SlopeIntegrator

- StatisticalIntegrator

Available Output formats:

- csv

- h5

## Stack Overflow

Searching a folder using java for certain files. See - <http://stackoverflow.com/questions/9530921/list-all-the-files-from-all-the-folder-in-a-single-list>

Solving the Testing error:

<http://stackoverflow.com/questions/22364433/activityunittestcase-and-startactivity-with-actionbaractivity>

## 

# Annotated Bibliography

This final section should list all relevant resources that you have consulted in researching your project. Each reference should also include a brief annotation.

1. The PD Academy is an online resource for sharing knowledge on Partial Discharge. Link: <http://www.partial-discharge-academy.com/what-is-partial-discharge>
2. Sinan Si Alhir, UML in a nutshell – A quick reference, First Edition. 1998 O’Reilly & Associates. ISBN 1-56592-448-7   
     
   I used this book for a reference for the UML diagrams used is this report. I found the overall organization of the Diagrams and which ones to put where most helpful.
3. Apache Software Foundation (2004) “Apache License, Version 2.0” (Online) Available at: <http://www.apache.org/licenses/LICENSE-2.0> Accessed: 14th March 2014.
4. Graph View - open source graph plotting library for Android. Version 4.0.0 – Available at: <http://www.android-graphview.org/> Accessed: 28th February 2015
5. GNU General Public License, link - <http://www.gnu.org/copyleft/gpl.html> Accessed: 28th February 2015.
6. Adding native support to the Android app. Link -- <http://www.javaworld.com/article/2077513/learn-java/java-tip-17--integrating-java-with-c--.html>

This is a tutorial I used to be able to understand how to set up the environment and utilise the C/C++ code. It gives an example and very clear diagrams on how issues can arise when using native code in the Android environment.

1. YouTube tutorial Available at <https://www.youtube.com/watch?v=kFtxo7rr2HQ>

Building NDK apps with Android studio is the tutorial title. I used this to help understand how to implement support of native code into the android app. While the tutorial is for an older version of Android it is still relevant.

1. Android NDK Available at <https://developer.android.com/tools/sdk/ndk/index.html>

Here is the NDK to build and compile the C/C++ code. It has instructions on how to install and implement it into the Android SDK.

1. Java Sound Resource Available at <http://www.jsresources.org/>

I used this website for further information into how to record and read wav formatted files. I found this very insightful and in depth regarding how java can handle audio data.

1. Audio file C library used in my Android app <http://www.mega-nerd.com/libsndfile/>

The libsndfile C library is used to read WAV format audio files. I used this library to read the audio file in and process it to a double array. I also got the sample rate from the file as it is needed for the feature extraction. This has been released under the GNU license.

1. H. William, P. E. Bartley, “Analysis of Transformer Failures”, International Association of Engineering Insurers 36th Annual Conference – Stockholm, 2003. Link -- <https://www.hsb.com/TheLocomotive/AnAnalysisOfInternationalTransformerFailuresPart1.aspx>

I used part one and part 2 for information on how transformers fail. It has all the stats and was very helpful in understanding how transformers fail and knowing that insulation failure

1. *Service handbook for transformers-3rd edition*, published by ABB.

I only read chapter 9.3 as this section was on monitoring. This game me an extensive list on how high and low level monitoring of electrical assests.

1. <http://www.fftw.org/index.html>

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST). FFTW, which is [free software](http://www.fftw.org/faq/section1.html#isfftwfree) under the GNU General Public License.

1. Cigre - Recognition of Discharges, Electra No 11 page 61-98.

This paper goes through the types of partial discharge that can occur on high voltage assets. It shows the observed discharge and its characteristics of any unwanted disturbances, and some means of diagnosing them. It is not a complete list and all results are shown on an oscilloscope.

1. Android developer website, <http://developer.android.com/index.html>. Accessed through out the project.

The Android developer website has a lot of information regarding the design, the development and the finishing touches on how to create an Android app.

1. PD Electrical sensors, <http://www.fargocontrols.com/sensors/>

I used this website for a further insight into how both capacitive and inductive sensors work.

1. Partial Discharge Detection and Localization in High Voltage Transformers Using an Optical Acoustic Sensor by Alison K. Lazarevich.

This paper has a better understanding of the ways PD can be detected. I used this for an in-depth background into chemical, electrical and acoustic detection of Partial Discharge.

1. Partial Discharge Signatures of Defects in Insulation Systems Consisting of Oil and Oil-impregnated Paper by Mohamad Ghaffarian Niasar, 2012

This paper has an in-depth knowledge into partial discharge and its many forms. I have used this paper to refer to the definitions of the many types of PD and detection methods.

1. Partial Discharge Pattern Recognition of Transformer Based on Electric Signal and Ultrasonic Comprehensive Analysis, by Shutao Zhao, Baoshu Li and Yong Wang.

Abstract - According to the characteristics and disadvantages of PD electrical analysis method, an electro-acoustic and ultrasound combination partial discharge (PD) pattern recognition method is proposed.

I used this paper to see what statistical features can be found from the waveforms.

1. The hardware specification for the Samsung S4 - <http://www.gsmarena.com/samsung_i9500_galaxy_s4-5125.php>
2. History of Android, <http://en.wikipedia.org/wiki/Android_version_history#Android_1.5_Cupcake_.28API_level_3.29>

Accessed: 21th February 2015. I used this for the background on Android.

1. Android 4.4 compatibility pdf file, <https://static.googleusercontent.com/media/source.android.com/en//compatibility/4.4/android-4.4-cdd.pdf>

This document has all the information on the needed hardware and specification of any mobile device running Android 4.4. I only used the hardware section 7, for the storage information on the minimum requirements.

1. Trends in Partial Discharge Pattern Classification: A Survey, by N.C. Sahoo, M.M.A. Salama and R. Bartnikas

Abstract - Partial discharge (PD) detection, measurement and classification constitute an important tool for quality assessment of insulation systems utilized in HV power apparatus and cables.

This paper describes some of the tools used to extract features and compare how different AI approaches perform.

1. Android overall Architecture picture is from <http://theworkshop.screeneros.com/android-source-code-under-miscroscope/>

Access on 6th March, it was created by [Nanik Tolaram](http://theworkshop.screeneros.com/author/nanik-tolaram/), on 02 October 2014.