Université Toulouse III - Paul Sabatier
M2 IM - Master Informatique Image et Multimédia

Recette

Débruitage Doppler

Équipe :
Benjamin Azzini
Lucien Mahot
Aline Lepailleur
Kevin Lepan

Client et superviseur :
M. Denis Kouamé

18 Février 2016

Table des matières

1. Introduction

Ce rapport final entre dans le cadre du chef d'œuvre du Master Informatique Image et Multimédia. Notre sujet concerne le développement d'une application permettant de débruiter le signal correspondant au flux sanguin capté par un système d'acquisition basé sur l'effet Doppler.

L'objectif de la recette est de présenter le logiciel ainsi que son fonctionnement. Nous présenterons toutes les fonctionnalités et les scénarios prévus dans le cahier des charges, donnerons un descriptif du code final avec les résultats des jeux de test.

2. Couverture des spécifications

Dans un premier temps, nous avons fait une étude bibliographique sur différentes techniques de débruitages qui nous a permis d'isoler 3 types de filtres à même de nous aider au débruitage du signal. Ceux-ci étant le filtrage par ondelettes, le filtrage FIR (Finite Impulse Response) et le filtrage par EMD (Empirical Mode Decomposition). Après études des résultats de ces filtres sur des échantillons de test (codage en matlab) nous avons choisi d'inplémenter le filtrage par ondelettes.

Nous avons ensuite implémenté ce filtre dans notre application pour smartphones Android. Cette application permet comme demandé par le client d'afficher des informations telles que le spectrogramme. De plus elle traite le signal qu'elle reçoit en le

filtrant avec les ondelettes et permet d'enregistrer ce signal débruité.

3. Conception détaillée et description du code

3.1 Diagramme de classe

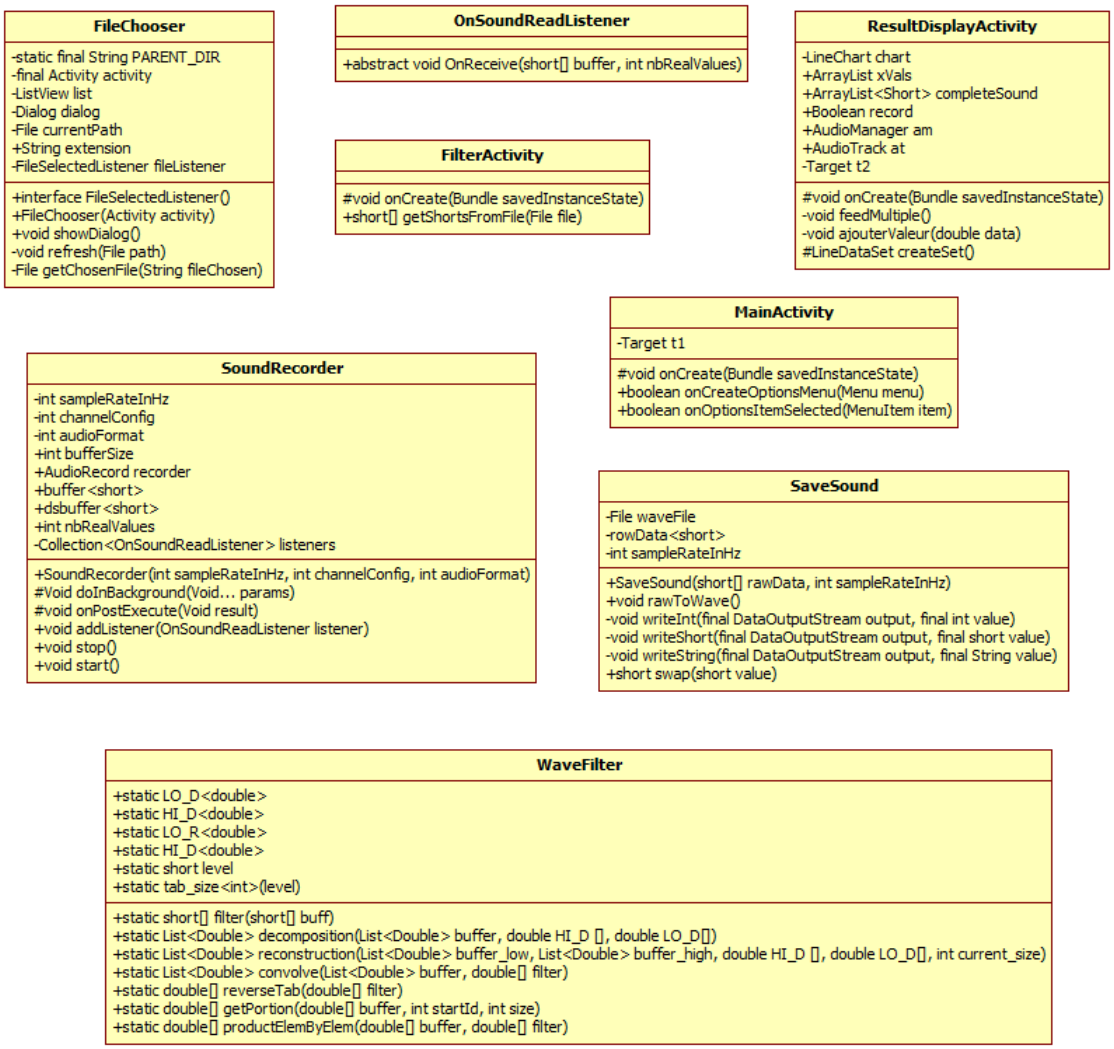

FIGURE 3.1 – Diagramme de classe

3.2 Descriptif des classes

Notre projet est divisé en plusieurs classes et chacune d'entre elles correspond à une étape clé de la chaîne de traitement. Nous allons aborder de façon plus précise ce que contient chaque classe ainsi que leur rôle.

3.2.1 SoundRecorder

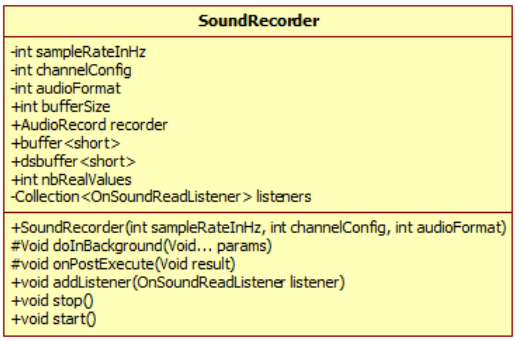

FIGURE 3.2 – Classe SoundRecorder

Cette classe récupère par le biais d'un buffer le son provenant de l'appareil de manière continue. Une fois récupéré, le buffer est concaténé pour former le signal que l'utilisateur récupère pour pouvoir passer dans la chaîne de traitement.
Le constructeur initialise les variables pour l'enregistrement.
La fonction doInBackground réalise la récupération de manière paralelle au thread UI.
La fonction onPostExecute réalise des opérations a la fermeture de la tache asynchrone (ici ferme le recorder).

3.2.2 SaveSound

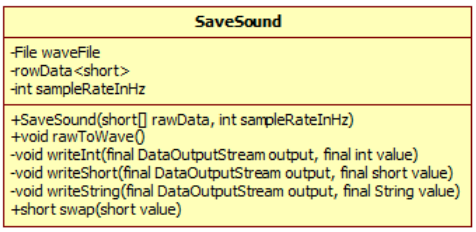

FIGURE 3.3 – Classe SaveSound

La classe SaveSound rend possible la sauvegarde du son avec le format WAV sur l'appareil de l'utilisateur. On créé alors un fichier contenant le header WAV (généré avec les informations du son) et le son sous forme d'un tableau de short.
Le Constructeur ouvre le fichier dans lequel on va ecrire.
La fonction rawToWave ecrit dans le fichier (le header puis les données).
Les fonctions WriteInt, writeShort, etc permettent d'ecrire directement des données de type short ou int a la place de les décomposer en bytes. La fonction swap permet de passer du format BigEndian au format LittleEndian et inversement.

3.2.3 WaveletFilter

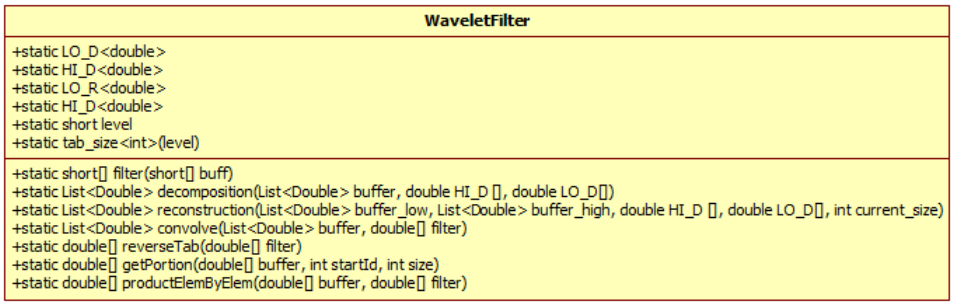

FIGURE 3.4 – Classe WaveletFilter

On transmet à cette classe les buffers représentant les morceaux du signal, c'est dans la fonction "filter(short[] buffer)" qu'on va exécuter le filtre que nous avons retenu. Les tableaux "LO_D, HI_D, LO_R et HI_R" sont les 4 filtres dont nous nous servons pour filtrer chaque buffer du signal sonore. Le filtre de décomposition en ondelettes est limité à un niveau de décomposition qu'on a fixé grâce à l'attribut "level". Ce filtre a été divisé en 2 parties, ces dernières sont les fonctions decomposition et reconstruction, et entre les deux on effectue un seuillage.

3.2.4 MainActivity

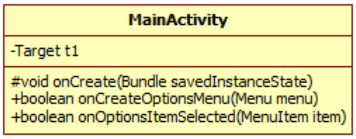

FIGURE 3.5 – Classe MainActivity

Cette classe est à l'origine du lancement des processus de l'application et de toutes les activités sous-jacentes correspondant à nos différentes fonctionnalités. Elle n'effectue aucun traitement particulier.

3.2.5 ResultDisplayActivity

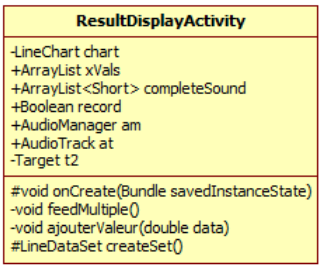

FIGURE 3.6 – Classe ResultDisplayActivity

Cette classe s'occupe de tous les affichages de notre application et de la réception de son. Le son est capté par le microphone et placé dans un buffer. La classe comprend les boutons d'enregistrement d'un son et de débruitage, qui lance donc les tâche d'enregistrement et de débruitage. Elle contient aussi le bouton d'aide pour informer l'utilisateur des différentes fonctionnalités comprises dans cette activité Android. On y affiche également le spectrogramme du signal capté en continu.

3.2.6 FilterActivity

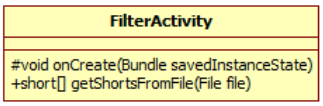

FIGURE 3.7 – Classe FilterActivity

La classe FilterActivity permet à l'utilisateur de choisir un fichier .wav stocké dans la carte SD grâce à la classe FileChooser. Le filtrage est ensuite effectué sur le dit fichier avec la fonction getShortsFromFile récupérant le buffer du signal pour l'activité de filtre.

3.2.7 FileChooser

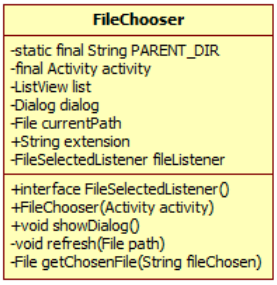

FIGURE 3.8 – Classe FileChooser

Cette classe permet l'intération avec l'utilisateur pour le choix du fichier d'un enregistrement pour l'activité de filtrage.

3.2.8 OnSoundReadListerner

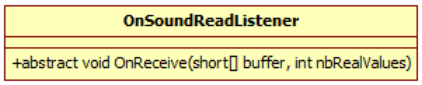

FIGURE 3.9 – Classe OnSoundReadListerner

Cette classe représente un listener sur un son. Elle permet de lancer un traitement à chaque fois que l'on reçoit un nouveau buffer contenant le son récuperé.

3.3 Descriptif du code

3.3.1 Entrées - sorties

La lecture d'un son (à partir du microphone interne ou d'un microphone externe) se fait grâce à la classe AudioRecord. Elle permet de lire le son en remplissant des buffers au fur et a mesure.

AudioRecord recorder = AudioRecord(MediaRecorder.AudioSource.DEFAULT, sampleRateInHz, channelConfig, audioFormat, bufferSize) ;

recorder.startRecording() ;

$while(recorder.getRecordingState() == AudioRecord.RECORDSTATE_RECORDING)$
$nbRealValues = recorder.read(buffer, 0, bufferSize);$

Cet exemple permet de lire en continu ce qui viens de la source DEFAULT (mic externe si branché, mic inter sinon) en echantillonnant à la fréquence sampleRateInHz sur channelConfig canaux le tout dans un buffer de taille bufferSize.

Pour le lire en retour on utilise :

$at = newAudioTrack(AudioManager.STREAM_MUSIC, sampleRateInHz,$
$AudioFormat.CHANNEL_OUT_MONO, recorder.getAudioFormat(), bufferSize, AudioTrack.MODE$

```
at.play() ;
at.write(buffer, 0, nbRealValues) ;
```

Ce code permet d'initialiser le playback et de le lancer (at.write doit etre executé pour chaque buffer).
Enfin, pour sauvegarder le signal sur la carte sd du téléphone il faut créer un header WAV avec les différentes informations (Fréquence d'échantillonnage, taille, etc ...) et ensuite ajouter les données récupérées.

3.3.2 Filtrage

Pour le filtrage nous sommes donc passés sur un filtrage par ondelettes discrètes. Nous sommes allés jusqu'à 4 niveaux de décompositions pour le traitement des hautes fréquences. Les premières versions du filtrage n'utilisaient que des fonctions Matlab pour déterminer quel filtre choisir rapidement. Par la suite, un premier algorithme a été écrit n'utilisant aucune fonctions Matlab pour faciliter le passage de Matlab vers Java.

Nous avons donc deux fonctions permettant de décomposer le signal reçu en deux parties, hautes et basses fréquences, et de reconstruire le signal à partir de ces fréquences. Pour chaque niveau de décomposition nous retravaillons sur le nouveau signal contenant les basses fréquences. Nous disposons à la fin de nos décompositions, de 4 vecteurs de hautes fréquences qui sont normalisés et filtrés avec

$$t = \sqrt{2log(length(signal)}$$

La normalisation est essentielle sans quoi aucune valeur n'est seuillée correctement. La reconstruction après seuillage nous donne un vecteur qui sera utilisé pour la

reconstruction vers un niveau supérieur jusqu'à l'obtention du signal sonore débruité.

4. Tests

4.1 Tests unitaires

Pour s'assurer du bon fonctionnement de notre application, nous avons effectué plusieurs tests unitaires dès lors qu'une fonctionnalité semblait terminée. Ce sont des tests que nous avons énumérés lors du rapport de conception.

1. Filtrage/debruitage

   Pour s'assurer du bon fonctionnement du filtrage, nous avons testé notre fonction sur des enregistrements déjà fournis sans avoir à se soucier de la bonne récupération du son de l'application. L'écoute du résultat de la soustraction du signal débruité au signal original nous suffit pour s'assurer du bon fonctionnement du filtrage.

2. Récupération du son

   Afin de tester la récupération de son, nous lisons le fichier audio par la suite. Les tests de récupération et d'enregistrement sont donc quasiment identiques.

3. Enregistrement

   Nous avons dans un premier temps envoyé un son quelconque sur le micro pour tester notre fonction d'enregistrement. La lecture de ce son sur le lecteur du téléphone fait office de validation de test.

4. Fft

Nous utilisons une bibliothèque existante pour le calcul des transformées de fourier et le calcul inverse. Mais cela n'empêche pas de comparer les résultats des tests avec ceux de Matlab.

5. Affichage

Notre IHM doit rester cohérente avec nos besoins et aussi par rapport à l'existant. Le spectre doit correspondre au signal que l'on écoute. Toutes les fonctionnalités de l'application doivent être disponible à l'utilisateur sur l'écran du téléphone, les tests d'affichage sont par conséquent pûrement visuels.

4.2 Tests finaux

Nous avons fait une comparaison entre le signal enregistré par notre application et celui enregistré par le système Cocoon Life sur la même personne. Mais ce test n'est pas très représentatif car en faisant les 2 enregistrements de manière consécutive, nous ne somme pas certain de placer la sonde exactement au même endroit et donc la comparaison n'est pas valable.

Pour pallier à ce problème, nous avons utilisé un enregistrement effectué avec la sonde doppler de Cocoon Life, que nous avons ensuite placé en dur dans notre application. Ainsi, nous avons pu faire une comparaison entre les applications avec le même signal.

5. Problèmes rencontrés et solutions apportées

5.1 Problèmes matériels

Le premier problème que nous avons rencontré est que nos téléphones n'étaient pas tous compatibles avec l'application Cocoon Life. Ce qui ne nous a pas facilité la vie pour la comparaison. Mais heureusement 3 de nos téléphones sur 5 étaient compatibles

avec l'application mais la sauvegarde d'une écoute sur un des 3 téléphones avec l'application originale se terminant par l'arrêt inexpliqué de l'application, donc seulement 2 téléphones étaient utilisables.

Un autre problème à été que les tests demandaient un materiel adapté. En effet, pour tester avec un son provenant de l'ordinateur il faut un dispositif adapté avec une prise TRRS permettant l'entrée d'un signal (correspondant au microphone) et la restitution de ce signal. De plus, le micro externe (ici simplement notre entrée jack) nécessite d'avoir une impédance d'au moins 1000 ohms pour être détecté par les telephones de marque Samsung. La solution a été dans un premier temps artisanale : fabrication par nous même d'un adapteur spécifique. Nous avons ensuite récupéré la sonde avec l'adaptateur fourni qui corrige ce problème.

5.2 Problèmes logiciels

Afin de développer notre application Android dans les meilleures conditions, nous avons utilisé l'environnement de développement Android Studio. Cependant même si l'environnement permet d'éditer des fichiers Java et de configuration, l'installation sur toutes nos machines à été relativement longue car nous devions avoir exactement les mêmes configurations et les bonnes versions du SDK afin que chaque membre de l'équipe soit capable de lancer l'application sur des téléphones vieilles générations. De plus nous avons eu quelques soucis avec Git et des modifications de fichiers de configuration qui nous ont forcé à revenir sur des versions précédentes.

5.3 Problèmes organisationnels

Nous n'avons pas pu developper notre application depuis Octobre à cause des cours, des rendus de tp et des partiels. Cependant, nous nous sommes éfforcés de préparer au mieux les dernières semaines qui étaient cruciales dans la réalisation de notre projet. Il apparait néanmoins évident que le travail d'étude sur la faisabilité du filtrage en tant direct aurait du être effectuée plus tot.

Enfin, la disponibilité de la salle 112 à été un réel problème car nous avons eu des difficultés a trouvr une salle on nous puissions rester toute la journée.

6. Ameliorations

Arrivé à ce stade de développement il y a 2 différents types d'améliorations à faire, la première amélioration correspond aux fonctionnalités prévues dès le début dans le cahier des charges qui n'ont pas été aboutis au jour de l'écriture du rapport et qu'il faudrait terminer. Le second type d'amélioration regroupe toutes les nouvelles fonctionnalités qui seraient utiles et qui feraient augmenter l'intérêt de l'application.

Parmi les fonctionnalités prévues au lancement du projet, déduire le rytme cardiaque du foetus est la partie de notre application qui a été la moins avancée. L'amélioration serait de réussir à retrouver le rythme cardiaque grâce au code Matlab et évidemment de transformer ce code en Java ou bien en C++ afin de l'insérer dans l'application. L'optimisation de la fonction de filtrage rendrait possible l'utilisation en temps réel de notre application. On pourrait paralléliser nos différents traitements ou modifier notre algorithme de filtrage.

D'autres fonctionnalités pourraient être utiles, pour la qualité du son par exemple, un avantage serait de pouvoir choisir, via une liste, le filtre que l'on veut appliquer sur le signal bruité. Pour améliorer l'interaction de l'utilisateur avec l'application, une nouvelle fonctionnalité de partage vers les réseaux sociaux serait intéressant afin que l'utilisateur puisse partager ces moments de bonheurs.

7. Planning prévisionnel

Ci-joint le diagramme initial idéal avant la réalisation du chef d'oeuvre en comparaison avec le diagramme effectif.

7.1 Initial

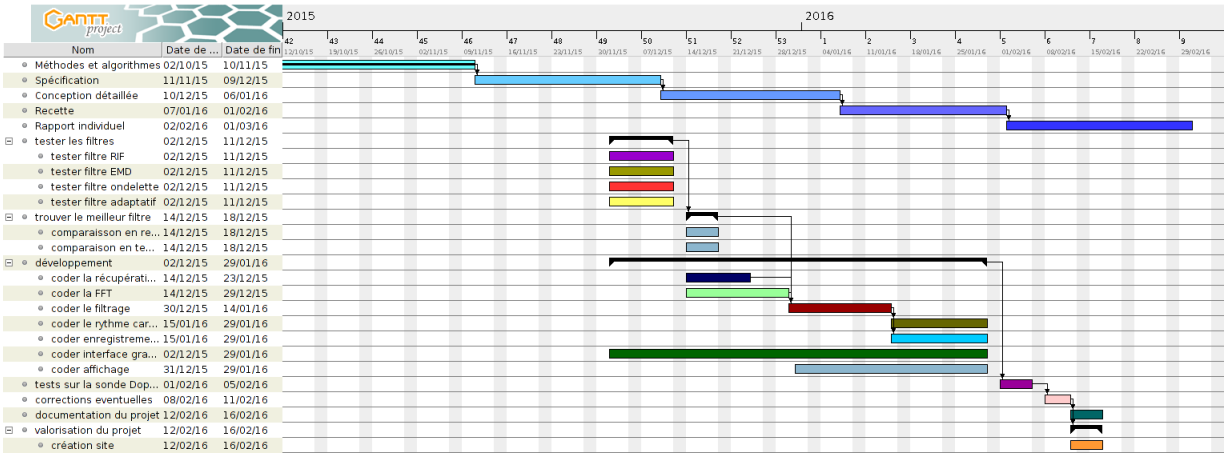

FIGURE 7.1 – Planning prévisionnel au lancement

7.2 Final

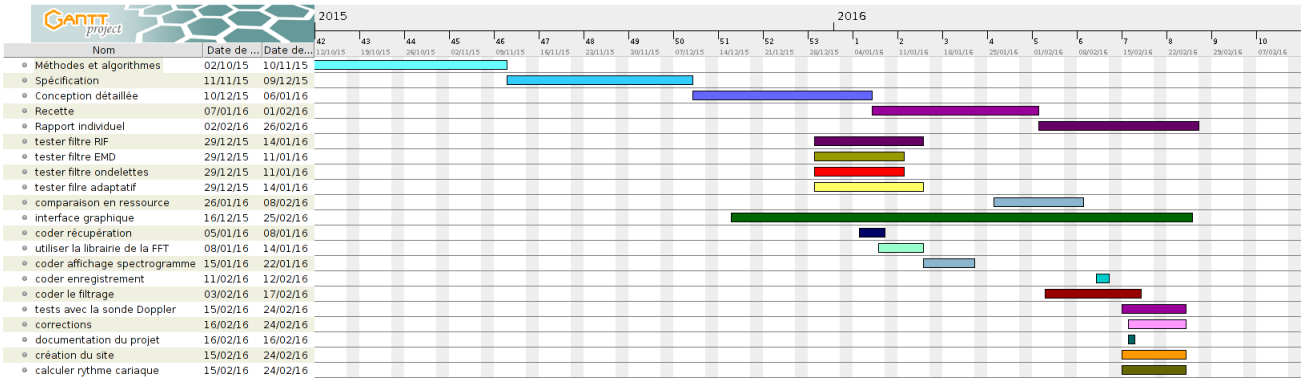

FIGURE 7.2 – Planning final à la recette

On peut se rendre compte qu'il y a un décalage global des tâches du projet qui est dû à la charge de travail du master. On remarque aussi que certaines tâches ont été plus longues à réaliser que d'autres. Comme c'est le cas pour la partie codage du filtre. Les tests ont été réalisés rapidement sous Matlab, mais il a fallu tout réécrire en java par la suite. Nous avons pu utiliser une bibliothèque pour la partie concernant la

transformée de Fourier, cela a pu nous faire gagner du temps en plus.

8. Documentation du projet

8.1 Manuel d'utilisation

Au démarrage de l'application, le fait de cliquer sur le gros coeur rouge lance l' affichage qui est le centre de l'application.

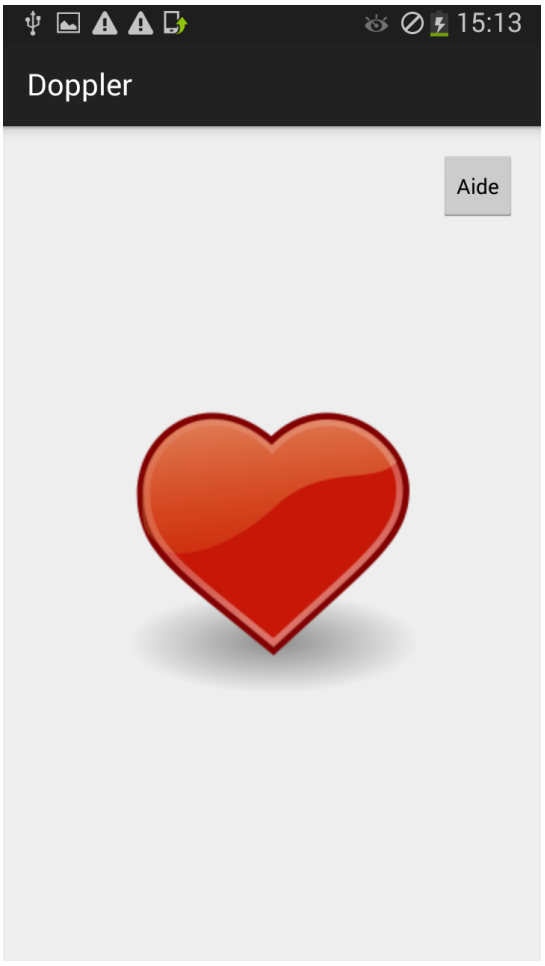

FIGURE 8.1 – Lancement application

L'activité principale s'ouvre et le spectrogramme du signal reçu s'affiche automatiquement sans qu'une action de l'utilisateur ne soit requise. Le bouton le plus gros permet de lancer l'enregistrement et de le stopper. Le bouton rectangulaire permet de lancer le débruitage sur un enregistrement précédement effectué.

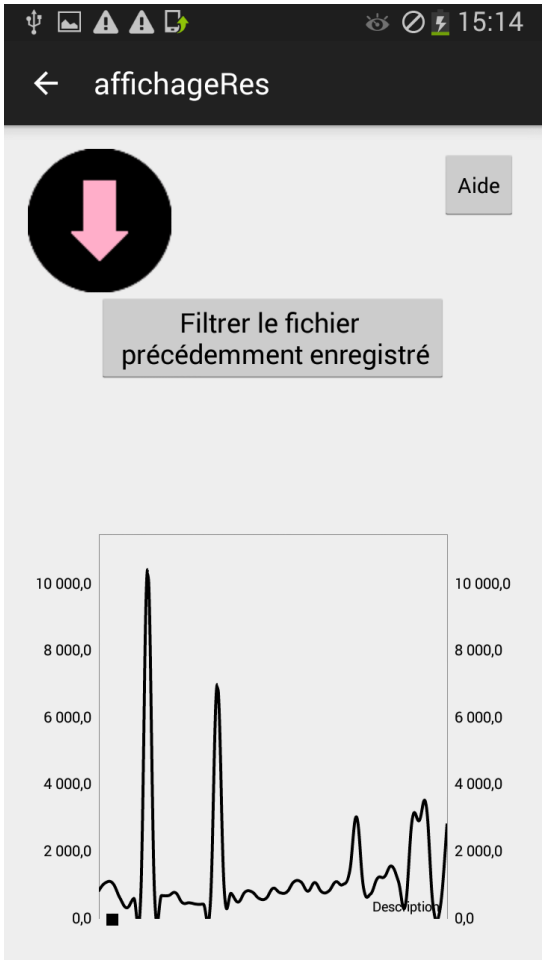

FIGURE 8.2 – Activité principale

L'utilisateur est ensuite invité a choisir un fichier a débruiter puis le débruitage se lance.

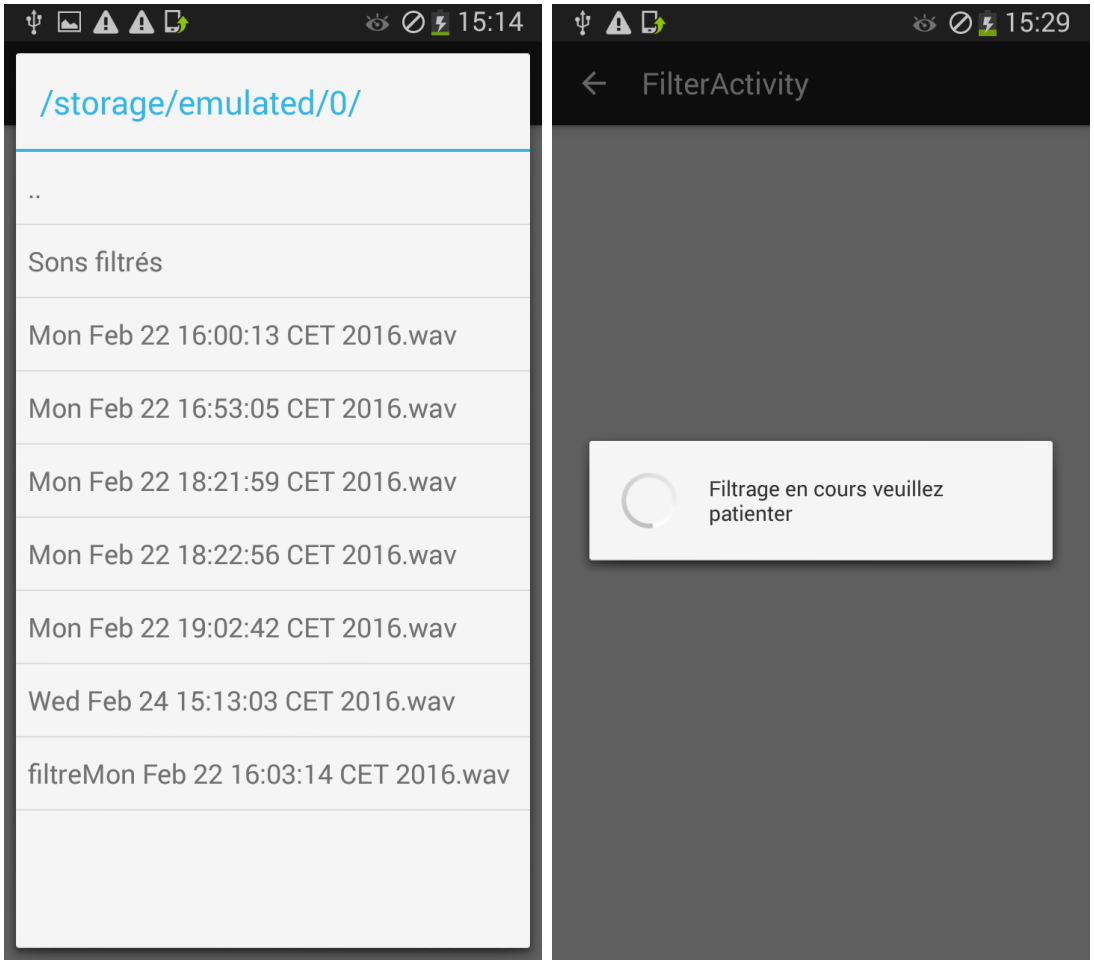

FIGURE 8.3 – Filtrage

Enfin, dans toutes les interfaces, le petit bouton d'aide permet d'afficher l'aide en localisant les objets avec lesquels l'utilisateur peut interagir.

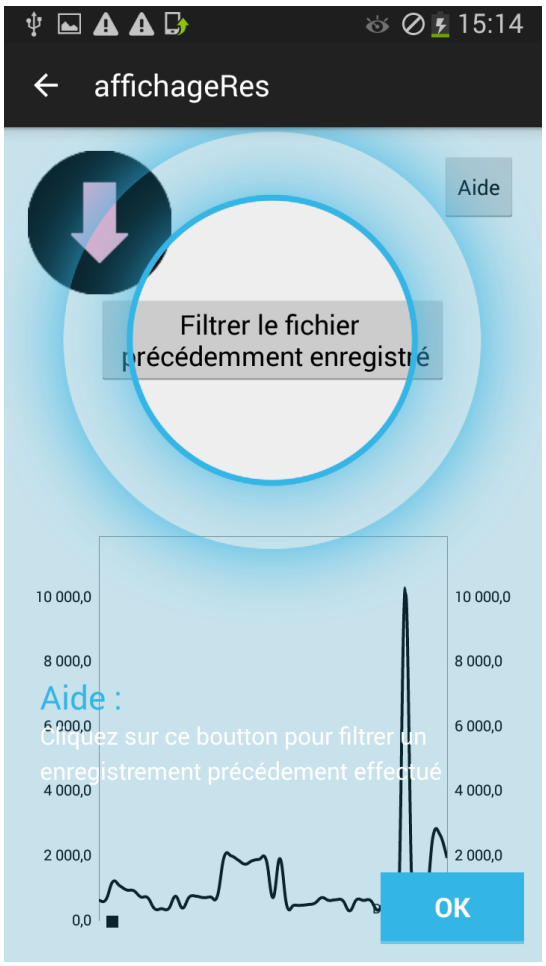

FIGURE 8.4 – Activité principale

8.2 Site web

Nous avons fait les pages du site de notre chef d'oeuvre pour exposer notre travail et permettre a des personnes intéréssées de nous contacter. Les pages web du chef d'oeuvre seront hébergées sur le site du master de notre formation, ce dernier contient donc une description du sujet, tous les rapports, les présentations orales, le code source,

la documentation du projet ainsi que nos coordonnéees.

9. Conclusion

Ce rapport de recette conclut sur l'avancée de l'application. Après 5 mois, l' objectif de notre chef d'oeuvre a été atteint car nous avons fait une application Android qui débruite un signal. Cependant, pour une meilleure utilisation de l'application nous avons des optimisations a effectuer afin de pouvoir atteindre un débruitage en temps réel. Nous avons prévu un entretien avec l'entreprise LATL TECHNOLOGY, créateur de la marque Cocoon Life, afin de leur présenter la solution que nous leur proposons au problème du signal Doppler bruité.

Bien que le calcul du rythme cardique n'était qu'un objectif secondaire que nous nous sommes ajoutés. Nous souhaitons continuer le projet après la soutenance, afin d'implémenter cette fonctionnalité qui nous tient à coeur et de pouvoir la présenter lors de notre entretien avec l'entreprise.