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

Rapport de "Méthodes et Algorithmes"

Débruitage Doppler

Équipe :
Benjamin Azzini
Lucien Mahot
Aline Lepailleur
Kevin Lepan

Client et superviseur :
M. Denis Kouamé

6 Novembre 2015

Table des matières

1 Introduction **2**

2 Problématique du Chef d'œuvre **3**

3 L'existant **4**
3.1 L'échocardiographie et l'effet Doppler 4

4 Algorithmes et Outils **5**
4.1 Débruitage 5
4.1.1 Introduction 5
4.1.2 Ondelettes 6
4.1.3 Seuillage 7
4.2 Création de l'application sous Android 9
4.2.1 Méthode de récupération d'un son en continu 10
4.2.2 Mise en place d'un filtre 11
4.2.3 Méthode de calcul de fréquence cardiaque 12
4.2.4 Mise en place d'un affichage temps réel d'une courbe 13

5 Conclusion **14**

1. Introduction

Ce premier rapport 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.

Le rapport « Méthodes et Algorithmes » permet à l'équipe de prendre du recul en étudiant des travaux existants susceptibles de nous aider à la réalisation de ce projet. Il s'agit donc de comprendre et d'expliquer les algorithmes et les formules mathématiques que nous allons intégrer dans nos travaux, ainsi que de présenter les outils que nous utiliserons.

2. Problématique du Chef d'œuvre

L'écho-cardiographie fœtale permet d'enregistrer le cœur du fœtus dès le 3^{e} mois de grossesse et son but est de porter un diagnostic précis des malformations cardiaques congénitales dépistées par le gynécologue lors des échographies. Depuis peu, une multitude de sociétés ont adapté l' écho-Doppler pour l'utilisation des particuliers. C'est le cas de la société Cocoon Life.

En effet, Cocoon Life produit un écho Doppler portatif qui permet aux femmes enceintes d'écouter le flux sanguin circulant au niveau du cœur de leur bébé (ou du cordon ombilical) et de voir un signal correspondant où qu'elles soient. Cependant le signal de retour comporte un bruit.

L'objectif de ce chef d'œuvre est de trouver le filtre adéquate permettant d'éliminer ce bruit, de débruiter le signal ainsi que d'implémenter une application pour smartphone qui affiche le spectrogramme et permette d'entendre le flux sanguin du bébé.

3. L'existant

3.1 L'échocardiographie et l'effet Doppler

L'échographe Doppler est constitué d'un échographe classique couplé à une sonde Doppler. Il existe plusieurs modes d'émission Doppler, dans notre cas d'étude le mode utilisé est le Doppler continu : Ce mode utilise une émission continue d'ultrasons avec une sonde à deux cristaux, l'un émetteur et l'autre récepteur. Il permet d'enregistrer des flux de très haute vélocité, sans limitation de vitesse mesurable. Son inconvénient est une moins bonne localisation du flux analysé.

Lors d'une échocardiographie Doppler, l'échographe envoie un signal ultrason $e(t)$ de fréquence $f0$. Quand un faisceau ultrasonique traverse le flux sanguin, la fréquence du signal $r_s(t)$ de retour sera différente de celle du signal émis. Elle peut être augmentée ou diminuée selon la direction du flux sanguin vers le capteur. C'est l'effet Doppler. L'effet Doppler désigne le décalage de fréquence d'une onde observée entre les mesures à l'émission et à la réception d'un signal.

Pour trouver le décalage Doppler, on doit d'abord multiplier le signal de retour par un signal en quadrature de fréquence $f0$. Cela nous donne un signal $m(t)$ contenant les fréquences des signaux émis et reçus. On utilise ensuite un filtre passe-bande pour supprimer les hautes fréquences du signal $m(t)$. Ce qui nous donne le signal $m_f(t)$ contenant le décalage Doppler c'est-à-dire la différence des fréquences d'émission $f0$ et de réception f : $f - f0$.

L'amplitude du changement de fréquence est proportionnelle à la vitesse du flux sanguin ainsi qu'à l'angle d'incidence du faisceau du signal. On pourra donc en déduire la vélocité et la direction du flux sanguin.

4. Algorithmes et Outils

4.1 Débruitage

4.1.1 Introduction

Les signaux obtenus lors de l'acquisition sont bruités, comme le montrent les exemples de sessions d'enregistrements de femmes enceintes. Afin de débruiter ces signaux, nous nous intéressons à l'article "Doppler Ultrasound Signal Denoising Based on Wavelet Frames" [3] se prêtant parfaitement à notre sujet, qui dans un premier temps calcule les coefficients des ondelettes du signal sur plusieurs niveaux de décomposition avec l'utilisation d'ondelettes non-décimées. Par la suite un algorithme de seuillage doux est appliqué afin de traiter ces coefficients pour obtenir le signal débruité.

Nous avons pu étudier avec l'UE de Traitement du Signal une méthode de filtre adaptatif qui débruite le signal en améliorant l'erreur quadratique moyenne, mais ce n'est pas suffisant ici. D'après les auteurs du document, l'erreur quadratique moyenne doit être minimisée sans ajout de fréquences. Ces fréquences supplémentaires proviennent du bruit et peuvent dégrader l'estimation de la résolution. L'étape de débruitage de signaux Doppler est importante, surtout lorsque le rapport Signal à Bruit est faible (inférieur à 10 dB).

Ces deux prérequis amènent à l'utilisation d'un seuillage doux. Afin de débruiter des signaux Doppler avec des transformées en ondelettes discrètes, Donoho [1], dont on ne traitera pas l'article ici, montre qu'il obtient de très bons résultats en comparaison à un seuillage dur. Mais la transformée en ondelette standard est variant par translation à cause du sous-échantillonnage, cela aura pour conséquences d'apporter quelques distorsions sur le signal reconstruit avec le seuillage doux, d'où l'utilisation d'une autre méthode de transformée en ondelette décrite dans cette publication. Le résultat du filtre n'est pas sous-échantillonné, les coefficients obtenus lors de la décomposition en ondelettes dans ce cas sont invariants par translation et appliqués à un seuillage doux, fournissent une meilleure performance en terme de résultat.

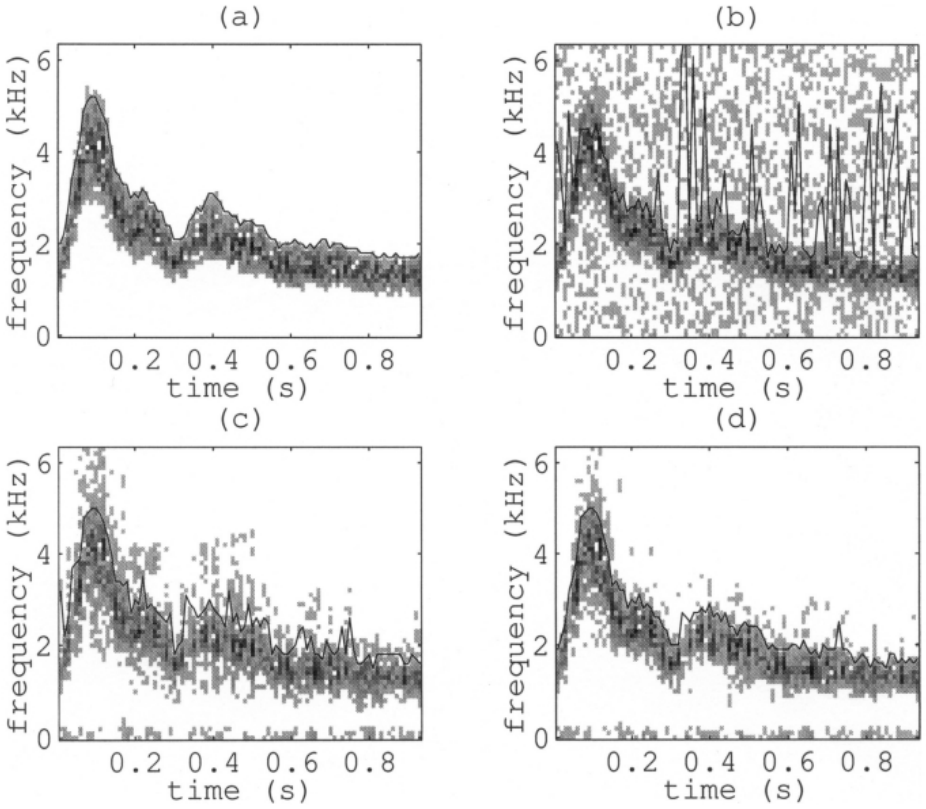

FIGURE 4.1 – Spectrogramme des résultats issus d'une simulation. (a) correspond au signal sans bruit. (b) est le signal bruité avec un rapport Signal à Bruit de 5 dB. (c) est le débruitage en utilisant la transformée en ondelettes standard, avec le 8^{e}ème niveau de décomposition. (d) est le spectrogramme du signal débruité avec la version non-décimée au 4ème niveau de décomposition. Le résultat en (d) est plus adouci contrairement en (c).

4.1.2 Ondelettes

Afin d'utiliser la transformée en ondelette nous avons besoin d'un filtre passe-haut et passe-bas, notre filtre adaptatif correspond à un filtre passe-bas qui vérifie la condition suivante :

$$H(z)H(z^{-1}) + H(-z)H(-z^{-1}) = 1$$

Ici, $H(z)$ correspond à la transformée en Z du filtre h. On peut facilement retrouver le filtre passe-haut comme étant le complémentaire du filtre passe-bas comme suit :

$$G(z) = zH(-z^{-1})$$

Nous nous intéressons aux ondelettes non-décimées, la reconstruction du signal est obtenue de la façon suivante :

$$x(k) = \sum_{l \in Z} s_{(I)}(l) h_I(k-l) + \sum_{i=1}^{I} \sum_{l \in Z} d_{(i)}(l) g_i(k-l)$$

$s_{(i)}(l)$ et $d_{(i)}(l)$ correspondent aux coefficients des ondelettes où i est un niveau de décomposition et sont respectivement liés au filtre passe-bas et passe-haut. Ces coefficients sont calculés de façon itérative rapidement :

$$\begin{aligned} s_{i+1}(k) &= [h]_{\uparrow 2^i} * s_i(k) \\ d_{i+1}(k) &= [g]_{\uparrow 2^i} * s_i(k), \\ &(i = 0, ..., I) \end{aligned}$$

Pour chaque niveau de décomposition, ces coefficients correspondent simplement à une convolution entre les filtres et les coefficients précédemment calculés. Pour le cas initial, le premier niveau de décomposition, les coefficients correspondent au signal.

4.1.3 Seuillage

Nous avons vu comment obtenir les coefficients des ondelettes, il faut maintenant effectuer un seuillage sur ces derniers. Cette technique de seuillage a l'avantage d'avoir l'erreur quadratique moyenne la plus basse tout en gardant une régularité dans le signal débruité. Le signal d'entrée est de la forme suivante :

$$F(n) = F_0(n) + \sigma e(n)$$

$F_0(n)$ étant le signal que l'on souhaite récupérer et $\sigma e(n)$ le bruit. Pour récupérer ce signal, 3 étapes majeures sont développées. La première étape a été décrite précédemment, il s'agit de la décomposition en ondelette, après avoir choisi le niveau de décomposition, on récupère les coefficients $d_{(i)}(l)$ du signal.

La seconde étape est l'application d'un seuil sur ces coefficients, le seuil étant calculé comme suit avec L la longueur du signal :

$$t = r_1.\sigma\sqrt{2log(L)}$$

Ce seuil appliqué aux coefficients nous donne les nouveaux coefficients suivants :

$$\eta_t(d_i(l)) = sgn(d_i(l))(|d_i(l)| - t)_+$$

Le signe $+$ signifie simplement que l'on garde seulement les valeurs positives.

Il nous reste à reconstruire le signal avec ces nouveaux coefficients.

Ci-joint les résultats issus de test clinique.

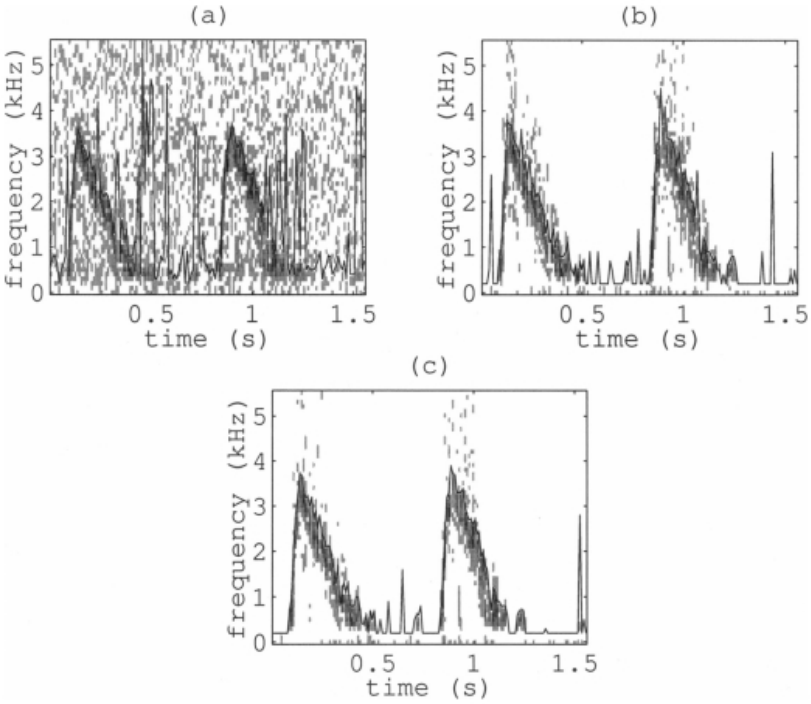

FIGURE 4.2 – L'image (a) est le spectrogramme du signal mesuré. (b) est le débruitage en utilisant la transformée en ondelette standard, avec le 8ème niveau de décomposition. (c) est le spectrogramme du signal débruité avec la version non-décimée au 4ème niveau de décomposition.

4.2 Création de l'application sous Android

Nous allons atteindre nos objectifs via une application Android qui sera décomposée en plusieurs parties. Tout d'abord nous devrons récupérer le son obtenu lié au Doppler, afin de pouvoir ensuite le filtrer en vue d'avoir un signal débruité le plus net possible. Avec ce dernier son filtré nous allons essayer d'extraire l'information caractéristique de la fréquence cardiaque du fœtus.

Depuis le signal obtenu on compte afficher l'information de ce son sous forme d'une courbe, comme par exemple l'affichage du spectrogramme correspondant au son. Bien évidemment toutes ces étapes devront s'exécuter en temps réel et en continu afin d'avoir une application la plus rapide possible pour l'utilisateur.

4.2.1 Méthode de récupération d'un son en continu

Nous voulons récupérer de manière continue un son, pour cela Android propose un framework multimédia incluant la capture et l'enregistrement d'éléments audio. Il y a la classe « MediaRecorder » et la classe « AudioRecord ». Le choix de la classe la plus adaptée dépend de ce que l'on veut faire avec le signal audio. D'un côté les données sont sauvegardées dans un fichier et de l'autre elles sont sauvegardées dans un tampon mémoire temporaire.

Ici nous avons besoin d'analyser le signal tant que l'enregistrement est en progression, nous avons également besoin d'effectuer des traitements et d'avoir accès au flux audio brut (simplement un vecteur de données). C'est la classe « AudioRecord » qui est la plus appropriée en nous donnant accès à toutes ces fonctionnalités. Attention tout de même avec « AudioRecord », puisqu'elle va ranger les informations dans un tampon mémoire, on doit alors lire et donc faire le traitement voulu assez rapidement de telle sorte que la mémoire (tampon interne) ne soit pas remplie.

Lors de la création, un objet « AudioRecord » initialise sa mémoire tampon audio associée qu'il remplira avec les nouvelles données audio. La taille de ce tampon, qui est à préciser lors de la construction de l'objet, détermine combien de temps cet objet peut enregistrer avant de réécrire par dessus des données qui n'ont pas été encore lu. Les données doivent être lues à partir du matériel audio en morceaux de façon à ce que leurs tailles soient inférieures à la taille totale de la mémoire tampon d'enregistrement.

4.2.2 Mise en place d'un filtre

Il existe de nombreuses grosses librairies comportant les fonctionnalités que nous voulons comme « libgdx », mais qui comporte également des méthodes dont nous n'auront pas l'utilité pour notre application. Pour palier à ces surplus de fonctionnalités inutiles, on peut utiliser de plus petites librairies spécialisées dans le traitement audio ou de signaux comme « Aquila ». Mais encore on peut très bien prendre une librairie correspondant à une méthode comme par exemple pour le calcul de la transformée de Fourier, la librairie « Kiss FFT » qui possède de bons avis concernant la rapidité d'exécution.

Cependant ces librairies spécialisées et petites librairies sont souvent codées en C ou C++. Pour ce soucis de rapidité, puisque nous somme en calcul en temps réel, il serait sûrement préférable de ne pas utiliser l'approche de Java avec Android afin d'éviter un ralentissement selon les appareils. Une approche native avec du C ou C++ serait beaucoup plus efficace et rapide que Java. Le NDK (Native Developpement Kit) est un outil très utile permettant d'implémenter des parties natives (en C ou C++) pour des applications sous Android. Ce choix d'utiliser des librairies natives en C/C++ pourra se faire si, une fois l'application terminée, nous observons que l'exécution est trop lente.

Malgré l'inévitable utilisation de librairies nous aurons des traitements à effectuer, le calcul de la transformée en ondelettes non-décimées par exemple.

4.2.3 Méthode de calcul de fréquence cardiaque

Après avoir récupéré le signal et l'avoir débruité, nous mettrons en place une technique permettant d'extraire la fréquence cardiaque foetale du signal. Dans leur article [2], Iulian Voicu , Jean-Marc Girault , Catherine Roussel , Aliette Decock et Denis Kouamé mettent en concurrence 4 méthodes permettant d'extraire le RCF (Rythme Cardiaque Fœtal). Nous utiliserons la méthode présentée comme étant la plus efficace : l'autocorrélation. Dans cet article, les algorithmes sont utilisés en parallèles sur différents signaux récupérés sur différents capteurs.

Malheureusement, le système que nous utilisons ne comporte qu'un seul capteur, nous devrons donc nous contenter d'un seul signal.

Selon l'article, l'autocorrélation offre une meilleure précision, un plus grand nombre de RCF estimés et demande moins de calcul que les autres méthodes.

Nous allons donc appliquer la méthode décrite dans cette publication à savoir :

- Appliquer un filtre passe-bande et son conjugué pour obtenir respectivement les amplitudes positives et les amplitudes négatives
- Filtrer ces amplitudes à 4 Hz qui correspond au maximum de RCF (240 battement/min)
- Autocorréler le signal
- Rechercher les N (que nous fixerons ultérieurement) maximums de cette fonction.
- Créer un vecteur de valeurs absolues des distances entre les maximums
- Si toutes les distances de ce vecteur sont inférieures à un seuil, le rythme cardiaque fœtal est donné par la moyenne des distances. Sinon, aucun rythme cardiaque ne peut être déterminé de manière formelle.

Ils nous reste alors à déterminer le nombre N de maximums que nous allons utiliser (en fonction de la mise à jour du RCF en temps réel dans l'application) ainsi que le seuil afin de trouver la meilleure configuration possible.

4.2.4 Mise en place d'un affichage temps réel d'une courbe

Une fois le signal débruité, il nous faudra l'afficher (et/ou afficher son spectre) dynamiquement et en temps réel (au fur et à mesure que le signal arrive). Pour faire cela sur une plateforme Android, nous devons utiliser une bibliothèque compatible, proposant une interface graphique agréable, réactive et qui nous permette d'ajouter et de supprimer des données dynamiquement.

Nous utiliserons la bibliothèque MPAndroidChart spécifiquement créée pour afficher des données sur Android. Après vérifications, nous sommes convaincus que cette bibliothèque répond à nos attentes.

Cette bibliothèque utilise des structures de données sous forme de classes (programmation en Java Android) ChartData. Nous utiliserons principalement la sous-classe LineData qui fonctionne avec des LineDataSet qui eux fonctionnent comme des listes.

Elle permet, en utilisant ces structures de données de réaliser un affichage en temps réel de données (ici notre signal Doppler et/ou son spectre).

5. Conclusion

Nous venons de décrire au travers des différentes méthodes, algorithmes et bibliothèques, les bases de fonctionnement de notre projet. Elles permettront d'apporter une solution au problème de bruitage et de produire un résultat propre et clair sur l'application. Il apparaît, au vu de ces méthodes que la contrainte d'un dispositif embarqué sera une problématique majeure de notre travail que ce soit en terme de temps de calculs ou de précision des résultats. Faire cohabiter plusieurs bibliothèques ainsi que les algorithmes que nous devons coder sera compliqué. De plus, il s'avère essentiel que nous réussissions à débruiter le signal sans quoi nous serions incapables de poursuivre en réalisant l'application.

Bibliographie

[1] David L Donoho. De-noising by soft-thresholding. *Information Theory, IEEE Transactions on*, 41(3) :613–627, 1995.

[2] Iulian Voicu, Jean-Marc Girault, Catherine Roussel, Aliette Decock, and Denis Kouame. Robust estimation of fetal heart rate from us doppler signals. *Physics Procedia*, 3(1) :691–699, 2010.

[3] Yu Zhang, Yuanyuan Wang, Weiqi Wang, and Bin Liu. Doppler ultrasound signal denoising based on wavelet frames. *Ultrasonics, Ferroelectrics, and Frequency Control, IEEE Transactions on*, 48(3) :709–716, 2001.