

**Rapport de Projet Industriel**

présenté par

**Antoine Chou, Lucas Combe, Houda Jebbari,  
Josua Pannier, Brice Croix**

Élèves ingénieurs de l'INSA Rennes

Spécialité EII

Année universitaire 2020 - 2021

# Extraction et évaluation de signaux vitaux à partir de la caméra d'un smartphone





# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	L'entreprise Sensoria Analytics . . . . .	1
1.2	Contexte technique . . . . .	1
1.3	Contexte scolaire . . . . .	2
1.4	Plateforme OpenQ . . . . .	2
<b>2</b>	<b>Études préliminaires</b>	<b>3</b>
2.1	État de l'art . . . . .	3
2.2	Recherche sur le développement mobile . . . . .	3
<b>3</b>	<b>Organisation du projet</b>	<b>4</b>
3.1	Répartition des tâches . . . . .	4
3.2	Pratiques mises en oeuvre . . . . .	4
3.3	Logiciels utilisés . . . . .	5
<b>4</b>	<b>Première approche avec l'API Camera2</b>	<b>6</b>
<b>5</b>	<b>Réalisation effective</b>	<b>7</b>
5.1	L'API <i>CameraX</i> . . . . .	8
5.2	Structure du projet et des classes . . . . .	8
5.3	Algorithme d'extraction du signal PPG . . . . .	11
5.3.1	Premier algorithme : Évaluation du signal PPG à l'aide de centroïde . . . . .	11
5.3.2	Second algorithme : Utilisation d'opérations simples sur les pixels . . . . .	12
5.4	Algorithme de conversion YUV420 . . . . .	13
5.4.1	Optimisation de code . . . . .	14
5.5	Estimation de la fréquence de battement cardiaque . . . . .	14
5.6	Menu statistiques . . . . .	15
5.7	Axes d'amélioration . . . . .	16
5.7.1	Utilisation de code optimisé . . . . .	16
5.7.2	Capture automatique . . . . .	16
5.7.3	Cross-Validation . . . . .	16
5.7.4	Signes vitaux supplémentaires . . . . .	16
5.7.5	Utilisation de threads supplémentaires . . . . .	17
<b>6</b>	<b>Résultats et conclusion</b>	<b>18</b>
6.1	Résultats et tests sur téléphone Android . . . . .	18
6.2	Comparaison réalisations/CdC . . . . .	19
6.3	Perspective d'avenir . . . . .	20
<b>Annexes</b>		<b>I</b>
Cahier des charges . . . . .		I
Diagramme de Gantt . . . . .		II
<b>Bibliographie</b>		<b>III</b>
<b>Glossaire</b>		<b>IV</b>



# Chapitre 1

## Introduction

### 1.1 L'entreprise Sensoria Analytics

Le projet présenté dans ce rapport a été proposé à l'INSA Rennes par l'entreprise Sensoria Analytics. Sensoria Analytics est une petite entreprise créée en 2017. Elle est spécialisée dans le traitement de signal et le développement embarqué appliquée à la santé et plus particulièrement au dépistage des maladies cardio-vasculaires. Le produit phare de Sensoria Analytics est CardioSensys. Il s'agit d'un logiciel qui se couple à un oxymètre avancé de photopléthysmographie. C'est un outil qui va réaliser une analyse de 7 signes vitaux clés du patient en 2 minutes. Ces signes vitaux sont les suivants :

- La fréquence respiratoire
- L'indice de rigidité artérielle
- La saturation en oxygène
- La tension artérielle
- L'élasticité artérielle
- Le rythme cardiaque
- La variabilité cardiaque

Cet outil permettrait donc ainsi d'intégrer le dépistage des maladies cardio-vasculaires dans la pratique régulière des professionnels de santé, tout en leur permettant de gagner du temps. Il s'agira dans ce projet d'étudier une autre façon de mettre en place ce genre de technologie.

### 1.2 Contexte technique

Les maladies cardio-vasculaires et accidents vasculaires cérébraux comptent parmi les principales causes de décès et d'invalidité dans le monde. Les patients touchés par ces problèmes doivent suivre différents tests pour le diagnostic des différentes maladies du cœur, diagnostic pouvant être aussi long que coûteux. Selon certaines études [1], l'utilisation de smartphones pour l'estimation de la fréquence cardiaque, de la fréquence respiratoire et du niveau de saturation en oxygène serait envisageable. L'idée serait d'utiliser la caméra d'un smartphone pour collecter un signal photopléthysmographique. Ce signal serait ensuite évalué qualitativement et quantitativement. La précision des différents signes vitaux obtenus pourrait alors être comparée avec les résultats d'un oxymètre et du logiciel *CardioSensys*.

Le but de ce projet est d'étudier la faisabilité d'une telle extraction aussi bien avec une plate-forme *OpenQ 865XR* (voir 1.4), qu'avec les smartphones classiques d'aujourd'hui. Pour ce faire, une application Android qui se chargera de l'enregistrement et de l'extraction de signaux pourra être développée et testée avec différents modèles de téléphones Android.



## 1.3 Contexte scolaire

Dans le cadre de la 5e année au département EII de l'INSA Rennes, les étudiants sont sollicités par des professionnels afin de réaliser un projet en groupe pour une entreprise : il s'agit du projet industriel. Les groupes ont été formés selon l'intérêt que chacun porte à tel ou tel projet, afin de réunir des équipes d'étudiants intéressés par le projet à réaliser. Les responsables du suivi de ce projet sont M. Nicolas Beauve (tuteur INSA) et M. Jean-François Nezan (tuteur entreprise).

Les sujets de projets sont en général fortement liés aux différentes matières étudiées tout au long du cursus EII. Dans le cadre de ce projet proposé par Sensoria Analytics, les principales matières étudiées mises en application sont le traitement de signal ainsi que le traitement d'image. Des notions de développement pour système embarqué peuvent aussi être appliqués afin d'utiliser la plateforme *OpenQ*. En ce qui concerne le développement informatique, le choix de la réalisation d'une application Android s'éloigne légèrement de la formation EII mais a par conséquent permis aux membres du groupe de se familiariser avec le développement Android qui était jusqu'alors complètement inconnu pour la plupart.

Il s'est donc agi dans ce projet d'appliquer les compétences acquises au cours de ces trois années de cycle ingénieur à un problème professionnel, mais également d'acquérir en autodidacte de nouvelles compétences nécessaires au bon déroulement du projet.

## 1.4 Plateforme OpenQ

Au début de ce projet fut fournie une plate-forme *OpenQ 865XR* visible en figure 1.1. Cette dernière utilise un processeur Qualcomm extrêmement similaire à ceux embarqués sur la plupart des smartphones du commerce.

L'avantage important de cette plate-forme est qu'il est possible de moduler les équipements que connectés au téléphone de la plate-forme. Cependant, à cause de la situation sanitaire exceptionnelle de 2020 et du fait qu'il soit impossible de se rendre à l'INSA, il ne fut pas possible de modifier les équipements de la plate-forme. Hors, la plate-forme ne possédait pas de flash, outil indispensable pour obtenir une qualité convenable de la capture comme cela sera développé par la suite. Il était donc plus intéressant de travailler avec nos différents smartphones équipés de flash, qui représentent mieux le support de l'application final.



FIGURE 1.1 – Plate-forme OpenQ



## Chapitre 2

# Études préliminaires

### 2.1 État de l'art

Le signal PPG est un signal obtenu grâce à des mesures optiques telles que la diffraction ou réflexion, correspondant au changement de volume de sang dans les vaisseaux sanguins. À chaque cycle cardiaque, les canaux vont se dilater et se contracter pour pouvoir envoyer le sang dans les différentes extrémités du corps. À partir de ce signal, diverses informations concernant la santé d'un individu peuvent être récupérées. On peut notamment récupérer son rythme cardiaque, son taux de saturation d'oxygène dans le sang, ou encore son niveau de stress.

De ce fait, les premières semaines ont été consacrées à la recherche d'informations sur le sujet, nous avons notamment pu lire différents articles scientifiques afin d'établir un état de l'art pour mettre en place une première stratégie de récupération de signal PPG.

Deux publications scientifiques [2] [1], datant de 2012 et de 2015, ont été étudiées et confortent la possibilité de récupérer un signal PPG. Ces études ont été réalisées avec les modèles de smartphones de ces années-là. Pour la réalisation de ce projet industriel, ce seront les smartphones des différents membres de l'équipe qui seront utilisés, ces derniers sont plus récents que les modèles des 2 papiers de recherche. Ceux-ci permettent pour la plupart une meilleure résolution et/ou meilleure fréquence de capture. Les résultats obtenus devraient donc être équivalents voire meilleurs que ceux obtenus dans ces articles.

### 2.2 Recherche sur le développement mobile

Dans un même temps, l'équipe, plus habituée au langage C/C++ a pu étudier le langage Java afin de pouvoir l'utiliser dans le développement d'une application Android. En effet, aucun membre ne maîtrisait réellement le langage et il a fallu allouer du temps pour pouvoir étudier ce dernier.

L'équipe s'est aussi penchée plus en profondeur sur le fonctionnement et l'architecture d'une application Android. En effet, le travail à faire de ce côté était plutôt conséquent puisqu'aucun de ses membres n'avait jamais fait de développement Android non plus. Il a donc fallu étudier le fonctionnement d'une architecture Android. Certains se sont également plongés dans l'apprentissage et la compréhension du langage Kotlin, nécessaire à la compréhension de code et APIs disponibles en ligne, dont les templates de l'API Camera2 auxquels ce rapport reviendra.





## Chapitre 3

# Organisation du projet

### 3.1 Répartition des tâches

Un premier groupe s'est occupé de définir l'état de l'art de la récupération de signal PPG par un smartphone grâce au papiers scientifiques fournis par Nicolas Beauve. Il s'agit là du groupe qui s'occupera majoritairement de la partie de récupération et de traitement du signal dans la suite du projet. Un second groupe s'est aussi intéressé au développement Android comme développé précédemment.

L'équipe s'est donc séparée en deux groupes, l'un axé sur le traitement de signal et l'autre axé sur le développement Android pur. Cette organisation choisie au début du projet était bien évidemment vouée à évoluer selon les besoins changeants au fur et à mesure de l'avancement du projet.

### 3.2 Pratiques mises en oeuvre

Tout d'abord, le groupe de projet n'a pas été divisé en 5 rôles précis. Le seul rôle attribué a été celui de chef de projet à Antoine Chou en raison de son expérience avec les technologies mises en œuvre. Il aura donc été celui qui communiqua avec le client et les tuteurs. La structure de l'équipe est tout de même restée horizontale puisque toute l'équipe était engagée dans la procédure de réflexion et que chacun pouvait proposer ses idées et être écouté.

L'organisation du travail continu s'est mise en place assez rapidement. En effet, en plus des créneaux de quatre heures de l'emploi du temps tous les vendredi matins, l'habitude a rapidement été prise de faire des réunions en dehors de ces créneaux afin d'échanger sur la façon dont chacun avait avancé, sur les difficultés rencontrées, etc. Ces réunions supplémentaires étaient en général au rythme d'une par semaine en plus du créneau dédié au projet, puisque chaque groupe avançait sur sa partie respective également en dehors des créneaux de base. On peut noter que ces réunions n'étaient pas nécessairement décrétées par le chef de projet mais simplement réclamées par n'importe quel membre du groupe en ressentant le besoin.

Ces réunions furent très utiles puisqu'elles permettaient à la fois un échange régulier sur les idées de chacun, et permettaient également de faire en sorte que tout le monde soit au courant de l'avancement général du projet. Ces réunions supplémentaires ont donc été conservées tout au long du projet.



### 3.3 Logiciels utilisés

Les logiciels utilisés par le groupe tout au long du projet sont les suivant :

- **Git** afin de pouvoir faire aisément du contrôle de version
- **Android Studio** l'IDE de la firme *JetBrains* qui offre de nombreux outils pour développer des applications Android.
- **Kanban** pour l'organisation et la gestion d'équipe. Ce logiciel permet de lister les différents objectifs à court et long termes et d'attribuer aux différentes personnes du groupe les tâches à réaliser.
- **Discord** a été l'application principale pour discuter et s'organiser au sein du groupe de projet
- **Zoom** a été le logiciel privilégié pour échanger avec les tuteurs ainsi que l'entreprise
- **PlantUML** a été beaucoup utilisé au début du projet afin de faciliter le design des différentes classes qui forment l'architecture du projet.

De nombreux logiciels ont donc été utilisés afin de mener à bien ce projet de la meilleure des façons malgré le mode de travail en distanciel. Le principal problème qu'a posé le distanciel est le fait que la plateforme *OpenQ* n'a au final que très peu servi puisque tout le groupe ne pouvait pas y avoir accès. C'est pour cette raison que les axes de développements ont vite été mis dans le sens d'une application Android, afin que chacun puisse développer et tester avec son propre téléphone portable.



## Chapitre 4

# Première approche avec l'API Camera2

Pour étudier le développement Android avec Java, le groupe a suivi un tutoriel sur le site Open-Classroom. En plus de cela, il était intéressant de voir que des codes templates de l'API *Camera2* permettant d'accéder à la caméra existaient déjà. Cela a permis au groupe d'avoir une première approche de la routine utilisée sur Android afin d'accéder à la caméra d'un téléphone. L'idée de base a été de se servir d'un exemple de code de capture vidéo à l'aide de l'API *Camera2* comme base de notre application Android. Ce template listait à l'origine toutes les caméras disponibles sur le smartphone et leurs différentes résolutions disponibles et proposait à l'utilisateur de choisir celle avec laquelle il souhaitait faire une capture. Ensuite, un fragment Android de capture s'ouvrirait afin de permettre à l'utilisateur d'effectuer la capture de sa vidéo.

Cela a servi de premier exemple "jouet" sur lequel le groupe a pu modifier différents paramètres. Il fut notamment possible d'épurer le nombre de choix de caméras disponibles en se restreignant uniquement aux meilleures caméras, c'est à dire celle avec une résolution HD ou Full HD. La caméra optimale était ensuite directement choisie par l'application. L'ajout d'un bouton pour la capture et pour allumer le flash pour toute la durée d'une capture vidéo a également été réalisé. Pour finir, il était possible de choisir où enregistrer la vidéo qui venait d'être capturée. Cette version est cependant écrite en Kotlin. Kotlin est un langage spécialement créé pour le développement Android. Il reste donc des parties du code difficiles à apprécier en raison du manque d'expérience avec ce langage. C'est là un point noir majeur de cette version : certaines lignes ne sont pas maîtrisées, ne permettant pas le contrôle de l'intégralité de l'application. L'architecture d'application Android proposée dans le template ne correspondait d'ailleurs pas aux architectures de bases enseignées dans les tutoriels OpenClassroom, une mise à niveau de tout le groupe de développement Android était donc nécessaire. Le choix a donc été fait de plutôt repartir de zéro avec une application parfaitement maîtrisée par chacun, développée cette fois-ci en Java.

En outre, dans cette version utilisant le template *Camera2*, la caméra enregistre systématiquement une vidéo. Ce serait donc une stratégie utilisant l'analyse post-capture qui serait adoptée avec cette application. Cela peut rapidement devenir encombrant pour l'utilisateur qui risque de se retrouver avec de multiples fichiers vidéos inutiles. De plus, puisqu'il est nécessaire de capturer la vidéo afin de l'écrire en mémoire puis venir la lire en mémoire pour l'analyser, une amélioration vers une analyse "temps-réel" n'est pas envisageable. En effet, les opérations de lecture et d'écriture en mémoire sont coûteuses. Néanmoins, l'analyse post-capture aurait tout de même permis d'éviter certains soucis rencontrés plus tard sur l'application finale. Notamment, une telle analyse permet de s'affranchir du problème de variance des FPS qui va être un souci non négligeable pour le traitement de signal et notamment pour filtrer les résultats. Ce problème rencontré sera discuté plus en détail, plus tard dans ce rapport.





## Chapitre 5

# Réalisation effective

Les défauts de la version précédente ayant été jugés trop encombrants par le groupe et par le correspondant entreprise, une nouvelle version partant de zéro fut réalisée. Cette version écrite en Java aura les mêmes fonctionnalités de bases que l'application précédente :

- Accès à la caméra à partir de l'API *CameraX*
- Accès au flash et allumage pendant toute la durée de la capture vidéo
- Affichage de ce qui est vu par la caméra à l'utilisateur
- Interface graphique avec bouton pour lancer la capture

Avec cette version, l'organisation des fichiers du projet a été rendue très simple, permettant une maîtrise de ce qui est réalisé par les membres de l'équipe. La plus grande différence par rapport à la version précédente se trouve sur la méthode d'analyse de la vidéo : Le traitement sera réalisé non plus après capture de la vidéo, mais en même temps que la capture de la vidéo. Ainsi, aucun fichier superflu ne sera sauvegardé sur le téléphone de l'utilisateur.

Cependant, une telle méthode de capture va engendrer des contraintes supplémentaires :

- Les calculs ne devront pas interférer avec la fréquence de capture de la vidéo. En effet, pour avoir une mesure valide en temps-réel, un nombre minimal d'image par seconde doit être respecté.
- Une fonction permettant de savoir si la capture est valide en temps-réel devra aussi être implémentée.

Le but final de l'application serait de pouvoir récupérer rapidement les signes vitaux d'une personne à partir de la caméra de n'importe quel smartphone récent. Dans les faits, l'utilisateur devra placer son doigt devant la caméra avant de lancer une capture, la vidéo récupérée sera ensuite analysée par l'application, différents algorithmes de traitements d'image et éventuellement du filtrage devront être implémentés. Les images capturées seront des images en apparence entièrement rouges puisque le doigt est placé sur la caméra. En réalité, des variations minimes de couleurs invisibles à l'œil nu existent sur ces images et ce sont ces variations qui seront exploitées pour la récupération des différents signes vitaux.

Pour vérifier la cohérence des résultats obtenus sur l'application finale, une validation croisée avec les outils de Sensoria Analytics pourra être effectuée.



## 5.1 L'API *CameraX*

Le contrôle de la ou les caméras présents sur un téléphone Android est une opération assez complexe. La firme Google, responsable des APIs de développement Android, a par le passé proposé plusieurs versions d'APIs intitulées *Camera* et *Camera2*. C'est cette deuxième qui fut utilisée dans la première version de l'application. Google a introduit cette deuxième version pour de nombreuses raisons dont la compatibilité avec tous les téléphones et une abstraction plus avancée du matériel.

L'API *Camera2* reste toutefois extrêmement verbeuse et la demande croissante des développeurs mobiles à faire du traitement d'image embarqué a poussé Google à introduire en 2019 une nouvelle version de son API intitulée *CameraX*. Cette dernière permet une abstraction poussée à l'extrême du matériel en implémentant trois cas d'usage très communs :

- Enregistrement de captures vidéos.
- Preview temps-réel de la caméra.
- Analyse d'image temps-réel.

La nouvelle version de l'application utilise *CameraX*. Ces cas d'usage étant cumulables, cela permet notamment l'affichage de ce qui est vu par la caméra à l'écran pendant que l'analyse d'image tourne. *CameraX* se voulant aussi très abstrait, il devient impossible de choisir aisément quelle caméra est utilisée sur les caméras au nombre croissant embarquées sur les téléphones actuels. Cette impossibilité couplée à celle de ne pouvoir imposer de résolution ou de fréquence de capture pourra poser problème dans la suite de ce projet. Il reste toutefois possible de "demander" une résolution et fréquence de capture minimale, sans n'avoir aucune garantie de si le téléphone sera en mesure de fournir un flux vidéo aux caractéristiques requises. Ce qui pourrait apparaître comme un défaut majeur permet cependant à l'application réalisée d'être compatible avec la quasi-totalité des téléphones Android du marché, incluant des modèles plus anciens embarquant des caméras de moins bonnes qualité.

## 5.2 Structure du projet et des classes

L'application réalisée utilise le patron de développement MVC. Un ensemble de classes, le modèle, gère les données utiles à l'application et les accès à celles-ci. L'affichage, c'est-à-dire la vue, organise les différents éléments graphiques visibles par l'utilisateur. Le contrôleur permet de lier ces deux éléments, faisant appel au modèle lorsque la vue est sollicitée. La structure de base du développement Android est parfaitement adaptée à ce schéma et permet la réalisation de vues en langage XML.

Le modèle est composé de deux classes principales : *BloodAnalysisSession* et *FrameInfo*, la première étant en quelque sorte le conteneur de la seconde. La classe *BloodAnalysisSession* correspond, comme son nom l'indique, à toutes les données qui seront relevées lors d'une analyse lancée par l'utilisateur. Une simple méthode *process* permet d'ajouter une image capturée par la caméra aux données capturées. Une instance de la classe *FrameInfo* est alors créée pour chaque image passée. Cette dernière se charge de tous les traitements et données inhérents à une seule et unique image (calculs de moyennes, conversions d'espaces de couleurs, etc). Le choix est fait de ne conserver comme attributs de cette classe que les données pertinentes de l'image donnée plutôt que d'enregistrer ladite image afin de ne pas surcharger la mémoire de l'appareil. La classe *BloodAnalysisActivity* enregistre donc cette nouvelle instance de *FrameInfo* dans une collection ordonnée. Cette dernière permettra par la suite d'extraire les informations inhérentes aux variations des données enregistrées (battement cardiaque, taux sanguins spécifiques, etc). La figure 5.1 illustre les relations entre modèle et contrôleur.

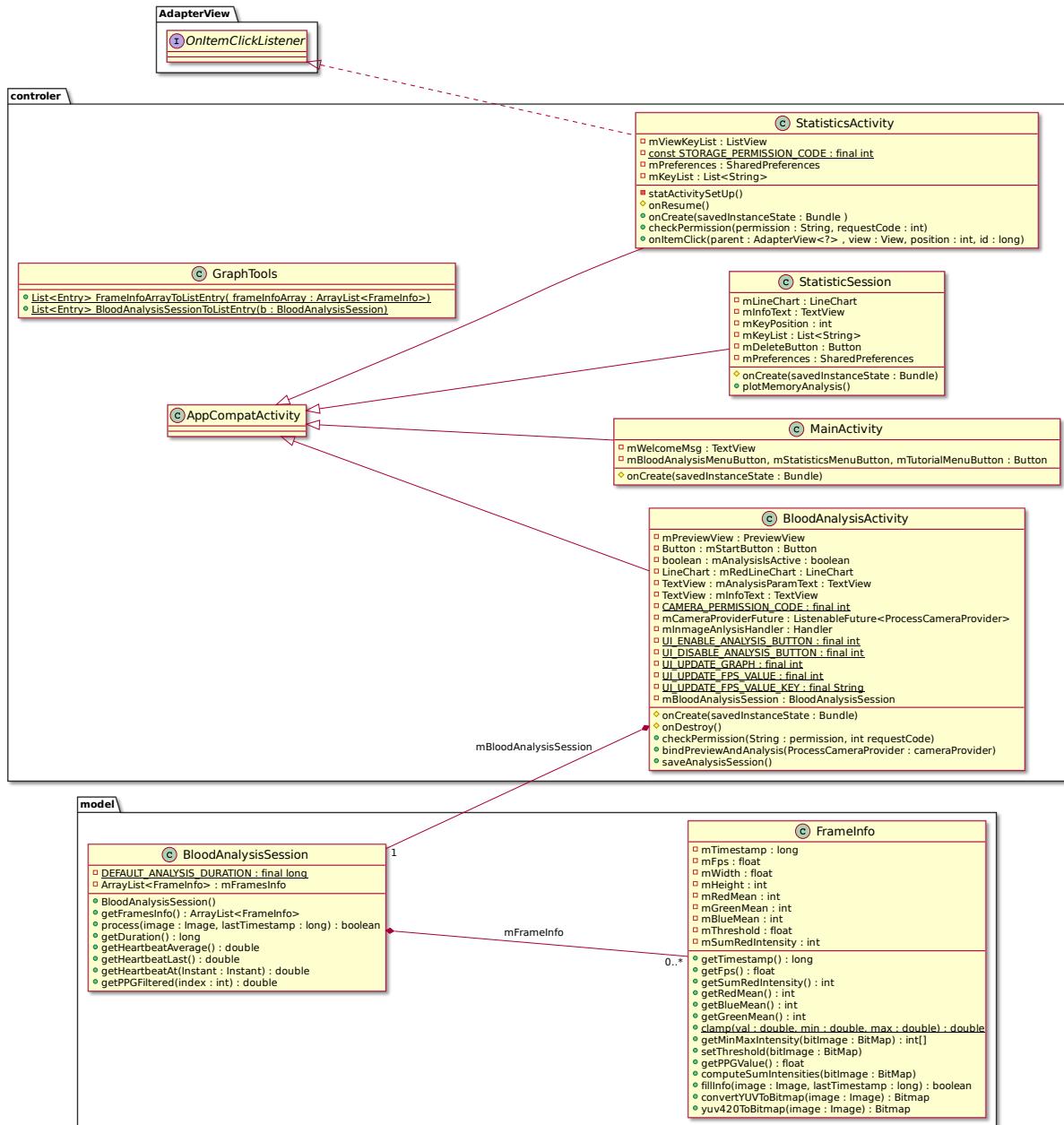


FIGURE 5.1 – Diagramme de classes du projet

Le contrôleur prend la forme d'un ensemble d'activités Android. Chaque "écran" correspond à une activité Android. Une activité est créée au lancement de chaque application, celle-ci gère ensuite le lancement d'autres activités, l'accès aux données, etc. Il apparaît donc naturel de créer une activité principale de type menu permettant le lancement de trois autres activités aux fonctions plus avancées : Analyse sanguine, statistiques, et tutoriel.

L'activité tutoriel, visible en figure 5.2 est assez simple et indique simplement à l'utilisateur comment positionner ses mains sur la caméra lors de l'extraction de signaux vitaux. Celle-ci permet aussi d'accéder à l'activité d'analyse sanguine sans repasser par le menu principal.

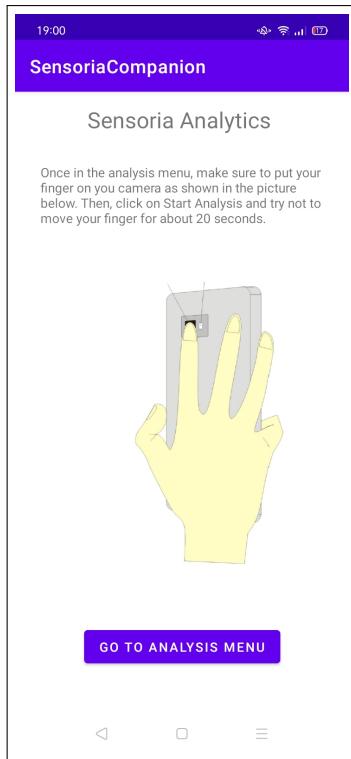


FIGURE 5.2 – L'activité "tutoriel"

L'activité Analyse sanguine constitue le principal intérêt de l'application réalisée. Celle-ci utilise les méthodes de *CameraX* pour permettre l'accès à la caméra et au flash de l'appareil. L'utilisateur pourra ainsi voir une *preview* de ce qui est vu par la caméra dans la partie supérieure de son écran, un emplacement destiné à accueillir un graphe lorsque l'analyse aura débuté, des zones de texte permettant l'affichage d'information pertinentes, et enfin un bouton pour débuter l'analyse sanguine. Ce dernier changera aussi de couleur afin de bien spécifier à l'utilisateur que l'analyse est lancée. La figure 5.3 illustre ces activités. Le rythme cardiaque n'est cependant pas cohérent, ce rapport y reviendra par la suite.

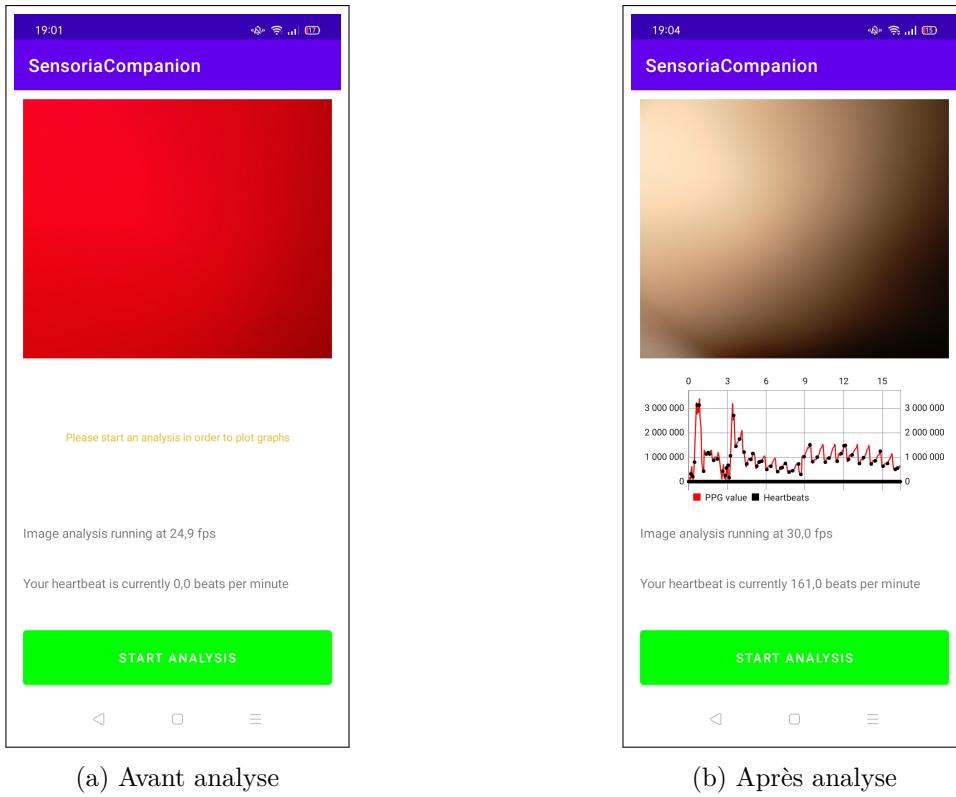


FIGURE 5.3 – L'activité "blood analysis"

## 5.3 Algorithme d'extraction du signal PPG

Les deux algorithmes d'extraction de signal PPG des deux articles scientifiques [2] et [1] ont été étudiés et comparés afin de déduire lequel est le plus adapté à l'application. Les deux approches proposées utilisent le même concept d'acquisition d'image avec caméra flash de smartphone et d'extraction de signal PPG.

### 5.3.1 Premier algorithme : Évaluation du signal PPG à l'aide de centroïde

Pour ce premier algorithme, le flash est optionnel dans le cas de bonnes conditions d'éclairage. Les mesures sont effectuées pour chaque nouvelle frame en fonction de la variation des couleurs capturées. Les moyennes et écart-types des composantes rouge, vert et bleu sont calculés et comparés à des constantes prédéfinies qui représentent les valeurs minimales et maximales atteignables, dépendantes du smartphone utilisé. Ces mesures permettent notamment de s'assurer que le doigt est placé correctement sur la caméra.

Après cette étape, l'algorithme s'intéresse uniquement à la composante rouge dont les mesures sont invariantes pour tout modèle de smartphone et toute condition d'éclairage. Un seuillage est alors appliqué sur la composante rouge afin de compter le nombre de valeurs dépassant ce seuil. Afin d'obtenir des mesures robustes, il est nécessaire de déterminer une zone en forme de disque qui correspond le mieux à l'emplacement où le doigt est le plus illuminé sur l'image, afin d'utiliser ce et seulement ce disque dans le calcul du signal PPG. Si le rayon du cercle dépasse la limite de la caméra, ces valeurs seront éliminées. Il est donc nécessaire de s'assurer que la zone formée par le disque soit bien dans l'image capturée. Sur la 5.4, se trouve en blanc, la zone délimitée par le seuillage, et en rouge, Le rayon correspondant à la valeur du PPG. Cet algorithme est extrêmement coûteux à mettre en place, l'article [2] utilise un ordinateur pour tracer les courbes après avoir enregistré les captures. Cet algorithme ne semble donc pas du tout adapté à une application Android aux performances limitées.

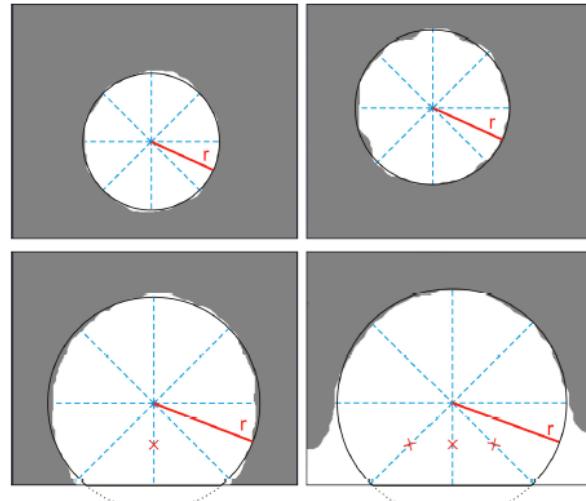


FIGURE 5.4 – Illustration de la méthode du centroïde

### 5.3.2 Second algorithme : Utilisation d'opérations simples sur les pixels

L'extraction de signal PPG sert à calculer la fréquence de battement cardiaque. Cet algorithme consiste à calculer certains paramètres pour chaque *frame* de flux vidéo afin d'extraire le signal PPG.

La première étape est l'extraction de la composante rouge du plan de l'image. D'après l'article [1], le canal rouge permet d'avoir le débit volumique sanguin pendant les cycles cardiaques sans autre traitement. Ainsi, cela simplifie la complexité de l'algorithme, diminuant la charge du processeur et rendant le système plus économique en énergie. Ensuite, l'intensité maximale et minimale de la composante rouge est calculée par la formule 5.1 pour définir le seuil.

$$seuil = 0.99 \cdot (I_{max} - I_{min}) \quad (5.1)$$

Les pixels avec une intensité supérieure au seuil sont sommés pour chaque frame, et cette somme constitue alors la valeur du signal PPG.

Ce deuxième algorithme est choisi pour être implémenté en raison de la simplicité des calculs afin de fournir moins de charge de travail au processeur. Le signal doit en effet pouvoir être récupéré même sur les smartphones les moins performants.

## 5.4 Algorithme de conversion YUV420

L'image récupérée par le téléphone et l'application Android est une image en format YUV420 dont les spécifications sont disponibles en figure 5.5.

Single Frame YUV420:



Position in byte stream:

Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15	Y16	Y17	Y18	Y19	Y20	Y21	Y22	Y23	Y24	U1	U2	U3	U4	U5	U6	V1	V2	V3	V4	V5	V6
----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	----	----	----	----	----	----	----	----	----	----	----	----

FIGURE 5.5 – Format de l'image YUV 420

Dans ce format, il y a 4 fois plus de valeurs de luminances que de chrominances qui sont stockées. Comme illustré sur la figure 5.5, la première valeur de chrominance (U ou V) est associée aux 2 premières valeurs des 2 premières lignes et ainsi de suite. Il faut ainsi parcourir l'image, ligne par ligne, et associer correctement le triplet de valeurs YUV pour reconstruire l'image au format RGB.

Il est possible grâce à l'API Android de récupérer indépendamment les 3 composantes de l'espace colorimétrique dans des tableaux. Les tableaux récupérés sont de dimension 1 et non de dimension 2, l'algorithme à implémenter devra donc segmenter ces tableaux selon la longueur de l'image. Il est ensuite nécessaire d'associer correctement les valeurs de chrominances et de luminances afin de les convertir en valeurs R, G, B. L'algorithme appliqué est visible en 5.4.

Algorithme de conversion en RGB

```

1 Fonction yuv420VersBitmap:
2     pixelStride = nombre de lignes sur une image
3     rgbArray = bitmap de sortie RGB
4     Y = buffer des valeurs de luminances
5     U = buffer des valeurs de chrominances
6     V = buffer des valeurs de chrominances
7     uvIndex = 0, index des tableaux de chrominances
8     RetourArriere = faux
9     Pour i de 0 jusqu'à la taille de l'image avec un pas de 1 :
10    Si(RetourArriere):
11        uvIndex = uvIndex - pixelStride
12        RetourArriere = faux
13    Sinon :
14        RetourArriere = vrai
15        yActuel = Y[i]
16        uActuel = U[uvIndex]
17        vActuel = V[uvIndex]
18        Calcul de conversion YUV vers RGB
19        rgbArray[i] = pixel en R,G,B
20        uvIndex++
21    return rgbArray

```

On utilise un booléen pour pouvoir retourner aux premiers éléments des tableaux UV une fois sur deux. Si cela n'est pas réalisé, l'image reconstruite sera faussée et des dépassemens mémoires seront même détectés.

Les formules utilisées pour les conversions sont celles indiquées dans l'article Wikipédia [3] et les suivantes :

- pixel rouge :  $Y + (1.402(V - 128))$
- pixel vert :  $Y - 0.71414(V - 128) - 0.34414(U - 128)$
- pixel bleu :  $Y + 1.772(U - 128)$

Cet algorithme permet de convertir l'image capturée par la caméra et de la convertir en format bitmap RGB. Ce dernier format permet des opérations au niveau des pixels plus simplement. L'image reconstituée n'est cependant pas parfaitement identique à celle de départ car le format YUV est un format introduisant des pertes.

#### 5.4.1 Optimisation de code

L'ajout de ces algorithmes introduit une forte complexité pour l'application impliquant certaines latences. En effet, un nombre élevé de calculs est demandé et la plupart des appareils Android vont enregistrer une capture avec un nombre d'images par seconde très faible (de l'ordre de l'unité), compromettant la possibilité d'obtenir des résultats exploitables.

L'optimisation de ces différents algorithmes a donc été étudiée en vue d'augmenter le nombre d'images par seconde. Différentes règles de codage ont été appliquées pour augmenter la vitesse d'exécution :

- Limiter l'accès aux buffers et tableaux, notamment si ces derniers sont des paramètres de la fonction.
- Mise en place de techniques de *loop-unrolling* pour pouvoir réutiliser les variables U et V dans l'algorithme de la conversion YUV en RGB. La nature de l'algorithme permet d'utiliser plusieurs fois les valeurs de U et V.
- Création de variables ou tableaux locaux pour stocker les différentes valeurs en local afin d'en accélérer la lecture. On peut directement charger un tableau bitmap entier au lieu de lire chaque valeur une à une.
- Utilisation, si possible, de boucles de type "*for : each*", qui sont les plus rapides et les plus optimisées sur les appareils Android.

Ainsi, chaque fonction de calcul utilisant des boucles a été revue pour suivre ces règles. Cela a permis d'obtenir un gain considérable au niveau du nombre d'image par seconde, atteignant jusqu'à 30 images par seconde pour les meilleurs appareils.

### 5.5 Estimation de la fréquence de battement cardiaque

Après extraction de signal PPG, un cycle cardiaque peut être calculé en prenant la différence entre deux minima consécutifs du signal. Le signal attendu est visible en 5.6 et présente en effet un minimum local à chaque battement cardiaque. La fréquence d'images dépend du nombre de frames par seconde capturées par l'appareil photo du smartphone et peut être estimée en calculant le nombre de cycles effectués par le signal PPG chaque minute, comme le montre l'équation 5.2.

$$f_{\text{battement cardiaque}} = 60 \cdot f_{\text{images/cycle}} \text{ beats/minute} \quad (5.2)$$

L'implémentation de cette fonctionnalité ne fonctionne cependant pas encore totalement pour plusieurs raisons. Tout d'abord, le signal PPG n'est pas toujours suffisamment propre pour qu'il soit possible de détecter les minima dont a besoin l'algorithme. De plus, le problème de la variation des FPS empêche le calcul précis et/ou simple de la fréquence cardiaque. La détection des pics correspondant aux battements cardiaque est néanmoins une fonctionnalité implémentée, mais qui ne fonctionne qu'approximativement et lorsque le signal est suffisamment propre.

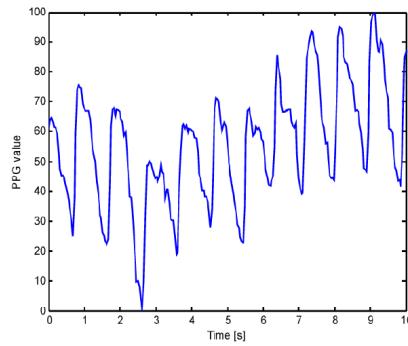


FIGURE 5.6 – Signal attendu

## 5.6 Menu statistiques

Le choix a été fait d'implémenter dans l'application un menu de statistiques, dans lequel seront recensées les différentes analyses précédemment effectuées. Le but est de pouvoir naviguer à travers les analyses effectuées, les revoir, afficher à nouveau les courbes ou même supprimer certaines anciennes analyses. Ce menu possède un affichage visible en figure 5.7.

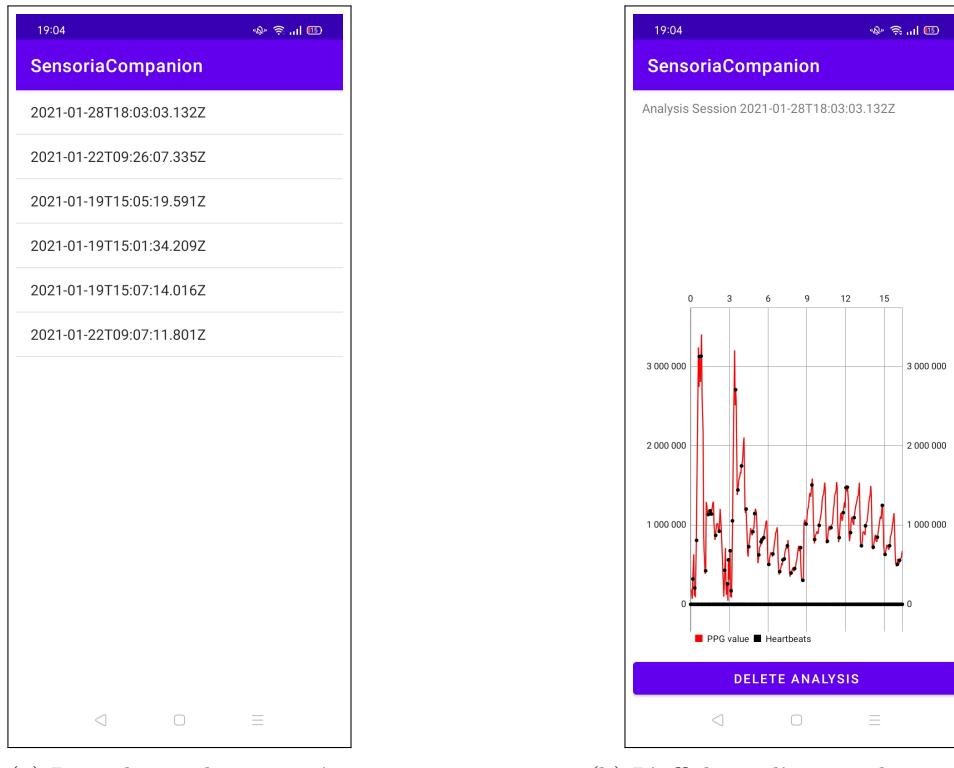


FIGURE 5.7 – Le menu statistique

Afin de réaliser ce menu, il a d'abord fallu se pencher sur la façons de faire transiter les informations d'une activité à une autre. Pour cela, l'application utilise les *SharedPreferences* d'Android. Il s'agit d'un fichier dans lequel il est possible d'enregistrer différentes informations inhérentes à chaque application : paramètres, données, etc. Néanmoins et afin d'écrire dans ce fichier les informations d'une analyse, il est nécessaire de sérialiser l'objet *BloodAnalysisSession* contenant toutes les *FrameInfo* enregistrées. Pour ce faire, la librairie GSON fut utilisée. Cette dernière permet de sérialiser directement une instance de *BloodAnalysisSession* en format JSON dans l'activité *BloodAnalysisAc-*

tivity dès qu'une analyse est terminée. L'analyse est alors stockée en mémoire avec pour clé la date et l'heure de la capture.

Ces informations sont donc stockées à la fin d'une analyse dans un fichier *SharedPreferences*. Ce fichier peut alors être lu dans l'activité statistiques afin d'afficher les différentes analyses disponibles. Une fois que l'utilisateur a choisi l'analyse qu'il veut visualiser, l'affichage est tel que décrit par la figure 5.7.

Le procédé de lecture est similaire à celui d'écriture : la librairie *GSON* déserialise les informations et les stocke dans un objet *BloodAnalysisSession* qui sera lu afin d'afficher les informations de l'analyse sur un graphe.

Une difficulté principale a été rencontrée lors du développement de ce menu. En effet, les objets *FrameInfo* avaient un attribut de la classe Android *Instant* permettant l'enregistrement d'un point dans le temps. Le problème est que la sérialisation de cet objet *Instant* ne fonctionnait pas avec la librairie *JSON*. Le choix a donc été fait d'enlever cet attribut *Instant* et de le remplacer par un type *long int* contenant le *timestamp* de chaque frame. En effet l'API *CameraX* encapsule chaque image dans une structure Android qui possède déjà un attribut *timestamp*, et c'est ce dernier qui est alors récupéré. Ce changement permet au menu statistiques de fonctionner parfaitement.

## 5.7 Axes d'amélioration

Actuellement, l'application manque encore d'importantes fonctionnalités pour rendre les mesures obtenues viables. Cette partie vise à mettre en évidence ces points qui ont besoin d'être améliorés. On peut notamment remarquer que le fait de fixer le *framerate* lors d'une capture est un prérequis nécessaire à la plupart des améliorations listées ci-dessous.

### 5.7.1 Utilisation de code optimisé

L'application réalisée souffre parfois de retards à cause des calculs effectués à chaque prise d'image. Cela a tendance à influer sur la fréquence de capture qui ralentit et n'est pas toujours constante. Une solution serait d'utiliser des langages et bibliothèques optimisées pour la vitesse, telles que OpenCV [4] [5], utilisée conjointement au langage C++. Ceci permettrait notamment d'ajouter du pré-filtrage directement sur l'image afin d'obtenir un signal plus propre en éliminant le bruit de capture.

### 5.7.2 Capture automatique

Afin de rendre l'application plus *user-friendly*, une des pistes possibles serait d'implémenter un algorithme de détection de mouvement pour détecter si l'utilisateur bouge trop [2]. Cela permettrait d'éviter que l'utilisateur ait à gérer le commencement et l'arrêt de la capture. L'idée est d'implémenter une méthode qui détecterait si l'image actuelle est suffisamment bonne ou non pour lancer la capture et s'assurer qu'elle sera de bonne qualité. L'application possède déjà cette fonctionnalité mais considère actuellement que toute image est exploitable.

### 5.7.3 Cross-Validation

Avant de commercialiser l'application, il faut impérativement faire de la cross-validation. En effet, la méthode utilisée pour extraire le signal de la vidéo laisse penser que le signal extrait est bien un signal PPG. Néanmoins, il va être nécessaire de comparer ce signal à un signal PPG extrait par un oxymètre de Sensoria Analytics afin de s'assurer que le signal est bien le même.

### 5.7.4 Signes vitaux supplémentaires

Comme expliqué dans l'introduction à propos de l'entreprise, Sensoria Analytics dispose d'un algorithme suffisamment efficace pour sortir 7 signes vitaux différents d'un signal PPG. On peut donc imaginer que si un signal suffisamment propre est obtenu via l'application, il serait également possible de détecter d'autres signes vitaux. Il faudra pour cela étudier les limites de la capture sur smartphone. L'application n'utilise notamment que le canal rouge des images, et l'obtention de signaux supplémentaires impliquerait de s'intéresser aux canaux bleu et vert, voire de déduire des longueurs d'onde supplémentaires comme le jaune.

De même la détection de pulsation cardiaque n'est actuellement pas correctement implémentée, et de nombreux faux positifs sont visibles (voir figures 5.3b et 5.6). Il résulte de ce problème que la valeur du rythme cardiaque n'est pas correcte et est bien trop élevée.

### 5.7.5 Utilisation de threads supplémentaires

Actuellement, l'analyse d'image et du signal PPG est pleinement exécutée sur un seul thread, en plus du thread principal servant à l'affichage de l'interface. Pourtant, la majorité des processeurs dans les smartphones d'aujourd'hui possèdent plusieurs coeurs. L'idée serait de paralléliser capture et analyse afin de gagner en performances. On peut notamment imaginer une parallélisation conséquente lors de la comptabilisation des pixels afin d'extraire le signal PPG. Les méthodes à paralléliser pourront être identifiées à l'aide d'outils d'analyses de performances et les gains théoriques pourront être évalués à l'aide de loi telles que celle d'Amdahl.

## Chapitre 6

# Résultats et conclusion

### 6.1 Résultats et tests sur téléphone Android

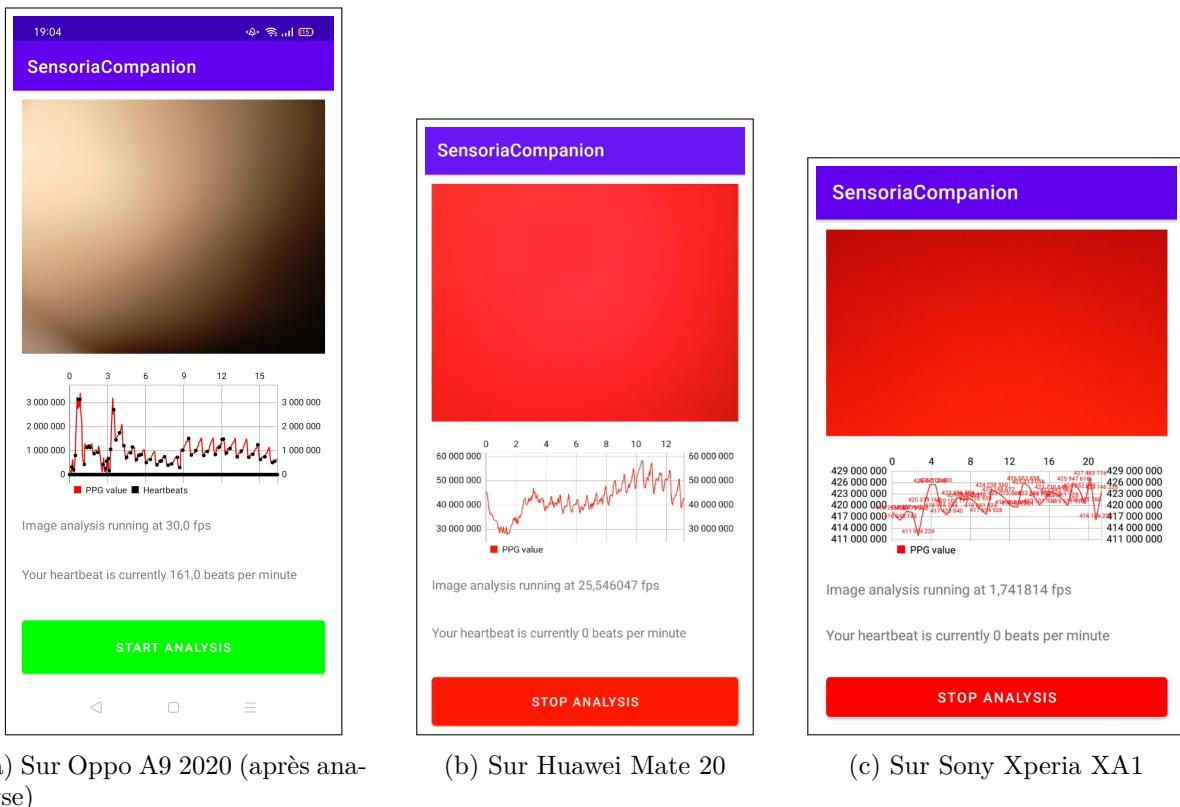


FIGURE 6.1 – Capture avec 3 smartphones différents

Les membres du groupe ont pu tester l’application sur leur propre appareil pour en vérifier le bon fonctionnement. Il est notable que les résultats dépendent fortement de l’appareil utilisé. Par exemple, sur la figure 6.1, il est possible de voir ce qu’il semble être un signal PPG entre 4 et 12 secondes sur la capture 6.1b ou après 4 secondes sur la capture 6.1a . Sur la capture 6.1c, la même personne a pris une capture avec un téléphone différent et les courbes obtenues sont beaucoup moins lisses, même différentes (en faisant abstraction des valeurs numériques qui sont affichés à chaque point). Cette différence peut être expliquée en partie par les performances de ces différents appareils :

- Les deux premiers appareils sont des téléphones haut et milieu de gamme possédant donc de bonnes performances permettant d’atteindre les 25 FPS et de rester stable lors d’une capture.

- Le deuxième téléphone est un appareil milieu de gamme ne supportant pas un nombre élevé de calculs. Le nombre d'images par seconde obtenu va osciller entre 1.5 et 4 FPS lors d'une capture.

Un des défauts notables actuellement de l'application est qu'il n'a aucun moyen de détecter si une mesure est correcte ou non. En effet, chaque capture a été lancée et arrêtée manuellement, et l'application ne mesure que les variations du canal de couleur rouge. De ce fait, l'application est très sensible au mouvement du doigt de l'utilisateur et les mesures ne sont pas robustes.

L'application a été testée plusieurs fois dans des conditions différentes jusqu'à obtenir les meilleurs résultats possibles, permettant ainsi de définir des critères pour une mesure correcte :

- Placer le doigt en face de la caméra et du flash pour obtenir une image la plus rouge possible comme visible sur les figure 6.1b et 6.1c.
- Prendre l'enregistrement dans un environnement sombre pour ne pas que la lumière ambiante interfère lors d'une capture.
- Limiter au maximum les mouvements, notamment ceux du doigt présent sur la caméra. Le mouvement est vraiment le paramètre qui génère le plus d'erreur.

Encore une fois, le nombre d'images par seconde obtenu ne peut-être contrôlé. Cela peut être dommageable aux personnes ne possédant pas d'appareil aux bonnes performances : Le but d'une telle application étant de rendre accessible facilement ce type de mesures pour toutes personnes possédant un smartphone.

## 6.2 Comparaison réalisations/CdC

Il va ici s'agir de dresser un bilan de ce qui a été réalisé par rapport aux réalisations prévues par le CdC. Tout d'abord, on peut noter que la version 0, étant la version la plus basique de l'application a été entièrement réalisée. Cette version 0 a d'ailleurs été réalisée sous deux angles différents. Une version Kotlin utilisant l'API *Camera2* de cette version 0 est disponible, et de même qu'une version Java utilisant l'API *CameraX*. C'est cette deuxième version qui a servi de point de départ pour la suite de l'application, la première étant restée à son stade de version 0. Ainsi, les points réalisés sur les 2 versions sont les suivants : accès à la caméra et au flash afin de faire une capture, affichage de la vue caméra et interface graphique simple pour lancer la capture.

La version la plus avancée de l'application étant la version Java, il va maintenant s'agir d'expliquer les implémentations supplémentaires de cette application. Premièrement, comme expliqué précédemment, l'extraction d'un signal PPG simultané à la prise de vidéo est réalisée. Cette extraction demande des calculs conséquents qu'il a fallu ensuite optimiser afin de garder un framerate correct. Néanmoins, un problème se pose à ce niveau puisque le signal récupéré est encore très bruité, même après essais de filtrage. Le résultat est que le temps a manqué pour pouvoir lisser et rendre lisible ce signal, il n'a donc pas été possible de réussir à véritablement récupérer un battement cardiaque cohérent. En revanche, le graphique du signal PPG obtenu est affiché en temps réel pendant la prise de vidéo. La forme du signal correspond bien à ce qui pouvait être attendu, il faudra toutefois confirmer la cohérence des résultats obtenus dès que possible. Un autre point intéressant du cahier des charges est l'idée du menu statistique. Ce menu statistique a été implémenté, il est donc possible de revoir les graphiques des signaux PPG des anciennes analyses réalisées.

## 6.3 Perspective d'avenir

Les deux dernières semaines ont été consacrées à la rédaction du rapport ainsi qu'au nettoyage des différents fichiers sources afin que le projet puisse être repris par d'autres développeurs. Bien que le groupe n'ait pas réussi à implémenter toutes les fonctionnalités voulues dû au manque de temps et de connaissances, l'objectif premier a bien été atteint : Un signal est bel et bien extrait à partir de la caméra d'un smartphone. Bien qu'aucune vérification croisée n'ait été réalisée à cause du contexte sanitaire actuelle, les résultats obtenus sont plutôt encourageants et permettront aux futurs développeurs de continuer là où le groupe s'est arrêté.

Enfin, personne dans le groupe de développement n'avait de compétences en Android au début de ce projet, qui a ainsi permis à tous de parfaire ses connaissances. Cela a vraiment été une première approche pour tous. La production réalisée reste de qualité satisfaisante, les normes de codage ont été respectées, le code a été soigneusement ré-écrit et des commentaires ainsi que l'historique des versions GIT sont aussi à disposition pour les futurs développeurs.



# Annexes

## Cahier des charges

### **Cahier des charges pour développement d'une application Android Révision du mois de Décembre**

#### *Présentation de l'application :*

Le but final de l'application serait de pouvoir analyser des images/vidéos afin d'en extraire un signal pléthysmographique. La vidéo serait aussi capturée par cette même application. On utilisera l'API CameraX, qui semble être compatible avec toutes les versions Android supérieures à 5.0 (API level 21).

La version 0 a déjà été terminée et présentée le 1/12/2020. On peut toutefois rappeler ses fonctionnalités :

- **Version 0** : version la plus basique de l'application nous permettant de nous assurer de la faisabilité de l'application.
  - Accès à la caméra pour chaque appareil Android afin de lancer la capture
  - Accès au flash pour pouvoir l'allumer pendant tout le temps de la capture
  - Interface graphique très simple avec un bouton pour lancer la capture
  - Affichage de ce qui est vue par la caméra

Deux approches ont été étudiées :

- Une application "post-traitement" qui capture d'abord les vidéos avant de les analyser. Cette application a l'inconvénient d'accéder souvent à la mémoire et de remplir le téléphone de l'utilisateur avec des vidéos.
  - Une application "temps-réel" qui capture la vidéo et réalise les analyses, possiblement avec un délai, lorsque la vidéo en lecture est valide. Il n'y a pas de vidéos enregistrées ainsi, et c'est sur cette approche là que nous avons décidé de continuer pour la V1.
- **Version 1** : version qui supportera la lecture et l'analyse d'une vidéo frame par frame.
    - Implémentation des algorithmes de traitement d'image avec OpenCV
    - Affichage de graphiques et des premiers signes vitaux.
    - Voyant permettant de savoir si la capture est valide ou non
    - Ajout d'un menu supplémentaire pour récupérer les statistiques des mesures antérieures
    - Récupération des caractéristiques de la caméra, notamment le nombre de FPS (Voir l'étude d'un mode slow-motion, on veut 60 Hz dans l'idéal, 30 sinon)

En plus de l'application mobile, une autre partie du groupe se charge de l'étude des algorithmes de traitement d'images permettant d'extraire le signal PPG. Ces algorithmes seront à priori implémentés en Java, toutefois, si les performances sont jugées insuffisantes, l'utilisation du C++ sera privilégiée.

Si le temps le permet, des algorithmes plus complexes seront implémentés afin de récupérer des signes vitaux complémentaires. Les tests unitaires et d'intégrations pourront aussi être écrits pour valider la robustesse du code.

FIGURE 6.2 – Cahier des charges

## Diagramme de Gantt

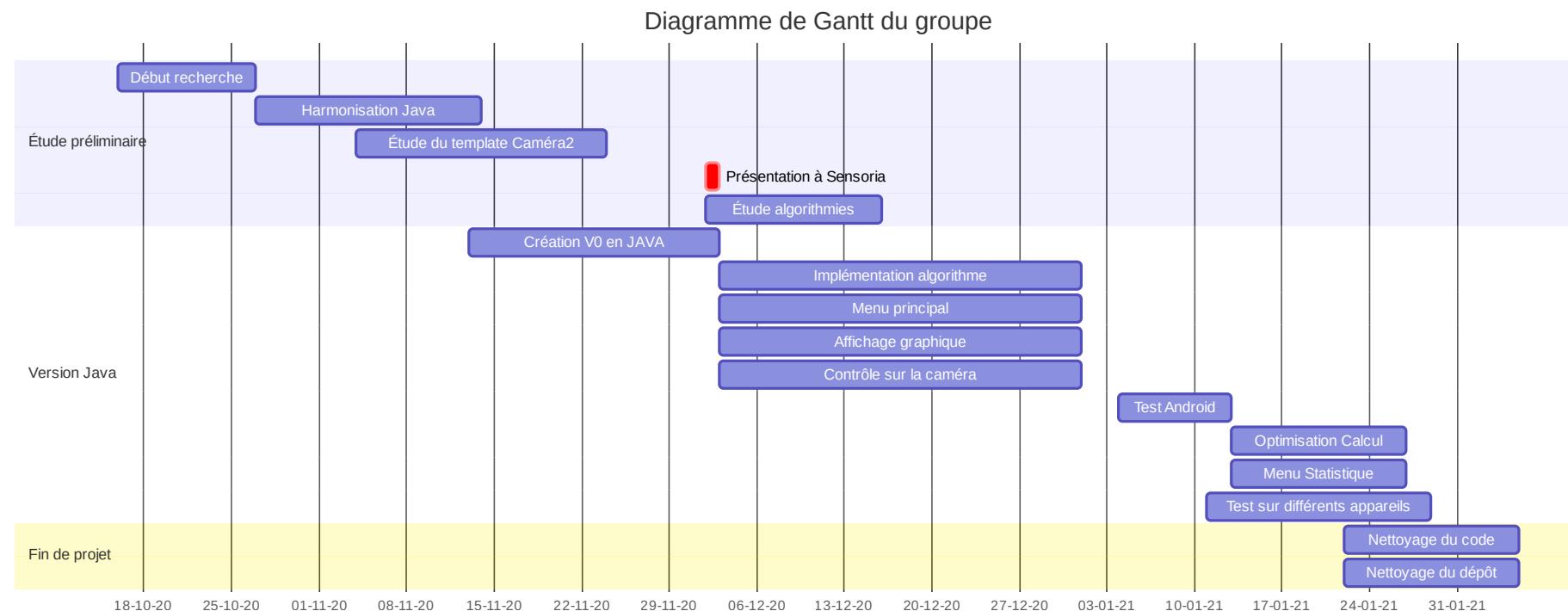


FIGURE 6.3 – Gantt réel



## Bibliographie

- [1] Yuriy Kurylyak, Francesco Lamonaca, and Domenico Grimaldi. *Smartphone-Based Photoplethysmogram Measurement*, pages 135–164. River Publishers, 01 2012.
- [2] Sarah Ali Siddiqui, Yuan Zhang, Zhiqian Feng, and Anton Kos. A pulse rate estimation algorithm using ppg and smartphone camera. *Journal of Medical Systems*, 40(5) :126, Apr 2016.
- [3] Wikipédia. Y'uv420sp (nv21) to rgb conversion, 2020.
- [4] OpenCV. Open source computer vision library, 2015.
- [5] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.





# Glossaire

**API** *Application Programming Interface* : l'interface de programmation applicative est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels. , 3, 6, 8, 13, 16, 19

**CdC** Cahier des charges. , 19

**FPS** *Frames per seconds*, un certain nombre d'images par secondes. , 6, 14, 18, 19

**HD** Haute Définition. , 6

**IDE** *Integrated Development Environment* (Environnement de développement) : un ensemble d'outils permettant de faciliter le développement logiciels. , 5

**Kotlin** Langage de programmation créée pour les applications Android. 3, 6, 19

**MVC** Modèle Vue Contrôleur, un patronne de développement commun dans la réalisation d'applications graphiques. , 8

**PPG** *Photoplethysmography*, technique optique employée pour trouver les changements volumétriques du sang dans le circulation périphérique. , 3, 4, 11, 12, 14, 16–19

**RGB** Espace colorimétrique *Red-Green-Blue*. , 13, 14

**XML** *Extensible Markup Language*, langage informatique. , 8

**YUV** Espace colorimétrique (Y : Luminance, U et V : Chrominance). , 13, 14



**INSA**





**INSA Rennes**  
20 Avenue des Buttes de Coësmes  
CS 70839  
35708 Rennes Cedex 7  
Tél. +33 [0] 2 23 23 82 00  
Fax +33 [0] 2 23 23 83 96

[www.insa-rennes.fr](http://www.insa-rennes.fr)

**INSA**

UNIVERSITE  
BRETAGNE  
LOIRE

Cti  
Commission  
des Titres d'Ingénieur

