ELE3000

Projets personnels en génie électrique

Conception d'un système d'acquisition sans-fil de signaux biologiques

Étudiant:
Vincent Pérot – 1584955

Présenté à:

Dr. Mohamad Sawan

Le 18 avril 2014

École Polytechnique de Montréal

Table des matières

Table des figures et tableaux

Introduction

Identification et présentation du problème

Les systèmes d'acquisition de signaux biologiques sont extensivement utilisés dans le domaine médical. Par le biais d'électrodes, on peut désormais acquérir de nombreux signaux, comme l'activité cardiaque, l'activité neuronale, les mouvements des yeux et les mouvements musculaires. Cela est utile pour les professionnels de la santé pour effectuer des diagnostiques plus éclairés.

Cependant, les systèmes actuels présentent certains défauts. En effet, la majorité de ces systèmes sont câblés. Cela limite grandement le confort des patients. Aussi, ils sont reliés à une station d'acquisition. Cela a donc un impact sur la mobilité des patients. Cela limite aussi grandement leur domaine d'application. Par exemple, on pourrait penser à utiliser l'activité neuronale et les mouvements des yeux et des muscles comme système de contrôle pour les jeux vidéo. On pourrait aussi utiliser des systèmes à ECG sur les individus à risque qui appellent automatiquement les secours en cas de problème cardiaque. Cependant, pour mettre cela en application, il faudrait avoir un moyen d'enlever tous les câbles. Ceci constitue le mandat du projet.

Présentation de la solution

Plus précisément, pour le projet, on se concentre sur l'électrocardiogramme, ou ECG. L'objectif est de concevoir un système qui acquiert l'ECG d'un individu, et de le transmettre de manière sans-fil à un téléphone intelligent. Le téléphone doit ensuite pouvoir l'afficher en temps réel afin de prouver qu'il obtient bien l'ECG. Le système doit être portatif et non-encombrant afin qu'il puisse être utilisé toute la journée par un individu.

Ce type de système élargirait considérablement les domaines d'application des systèmes d'acquisition d'ECG. Tout d'abord, il pourrait bien évidemment permettre aux patients dans les hôpitaux de se déplacer plus librement, mais cela permettrait aussi la surveillance des personnes à risque cardiaque. Si la personne a un problème cardiaque, le téléphone, connaissant l'ECG de la personne, est en mesure d'identifier le problème et d'appeler les urgences. Un autre domaine d'application possible est le fitness. A partir des ECG, le téléphone intelligent peut proposer des exercices personnalisés à un individu, en fonction de sa forme physique.

Annonce du plan

Ce rapport présente la solution proposée. Plus précisément, après une revue de la documentation et des systèmes actuels, la méthode de conception est détaillée. Tout d'abord, on abordera le design préliminaire. Cela nous permettra de définir une architecture, puis un design détaillé. Par la suite, les différents tests effectués afin d'obtenir un système robuste seront présentés. Finalement, la conclusion résumera les différents résultats obtenus ainsi que les apprentissages réalisés.

Développement

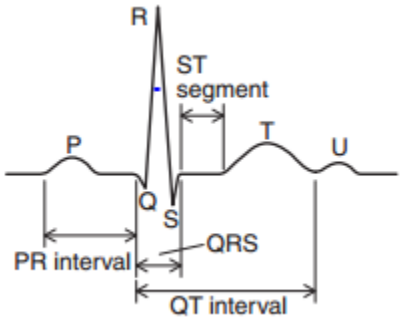
Pour la réalisation de ce projet, on utilise une méthode de conception classique. On commence par une revue de la documentation, puis on commence par définir le design préliminaire.

Revue de la documentation

Pour mener à bien le projet, il faut d'abord savoir comment les systèmes actuels acquièrent les ECG. Pour cela, j'ai donc consulté différents documents pour m'informer sur comment acquérir l'électrocardiogramme[1]. Grace à cela, j'ai pris connaissances de faits importants.

Tout d'abord, l'électrocardiogramme se fait par des électrodes. Il en existe plusieurs types (avec contact ou sans contact, sèche ou mouillée, réutilisable ou à usage unique). Il existe plusieurs placements d'électrodes standards. Plus les électrodes sont proches du cœur, plus le signal sera fort. J'ai aussi appris que les fréquences composant l'ECG vont de 0,1 Hz à 150 Hz. Ces informations-là seront utiles lors de la conception afin de développer une solution censée qui répond aux contraintes de l'application.

L'ECG est composé de cycles cardiaques qui se répètent. Les différentes étapes du cycle cardiaque sont montrées ci-dessous :



[2]Figure 1 - Représentation du cycle cardiaque

Cette allure vient des contractions et relâchement des différentes parties du cœur, notamment les ventricules. Les détails du fonctionnement du cœur ne sont pas abordés dans ce rapport, car cela n'est pas utile à la compréhension de la solution proposée.

De plus, j'ai aussi consulté des documents sur l'amplification des ECG. Mon directeur de projet m'a beaucoup aidé dans ce domaine, en me montrant quelques circuits possibles. Notamment, un des circuits est le suivant :

[1, 2] Auteur inconnu, «What is the ECG about». [En ligne]. Disponible: https://www.us.elsevierhealth.com/media/us/samplechapters/9780443068171/ECG%20Made%20Easy%201-40.pdf. [Accès le 18 Avril 2014].

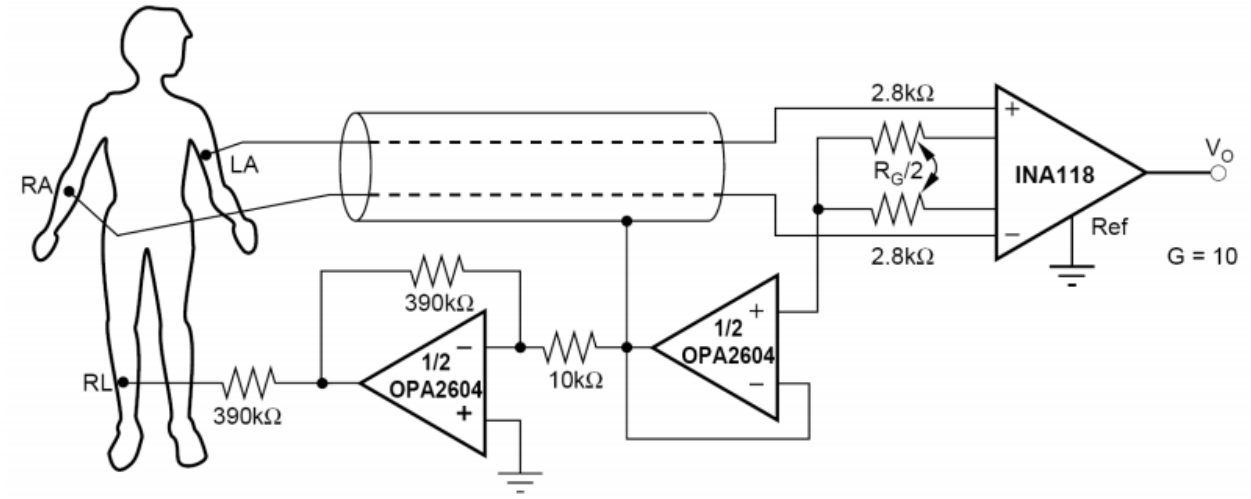


Figure 2 - Circuit d'acquisition d'ECG

La base de ce circuit est l'amplificateur d'instrumentation. Celui-ci amplifie la différence de potentiel entre les électrodes RA et LA. Son gain est déterminé par la résistance Rg. La troisième électrode est appelée le *Right-Leg Drive* (ou RL). Cette électrode sert à annuler le mode commun entre RA et LA et d'imposer un potentiel sur le corps afin que celui-ci ne soit pas flottant (et n'agisse pas comme antenne). Cela permet d'obtenir un signal beaucoup plus propre. Ce circuit sera à la base du module analogique développé lors du projet. On utilisera aussi trois électrodes, et des amplificateurs d'instrumentation.

Spécifications fonctionnelles

La revue de la documentation m'a permis de me donner une idée claire des différentes contraintes entourant l'acquisition d'un ECG. On peut donc commencer la conception. Pour cela, on définit tout d'abord les spécifications fonctionnelles, c'est-à-dire définir les entrées et les sorties du système. On définit aussi les besoins et les fonctions que le système doit réaliser. On définit aussi les facteurs humains qui entrent en compte dans le comportement du système ainsi que le comportement du système face aux erreurs. Schématiquement, cela donne :

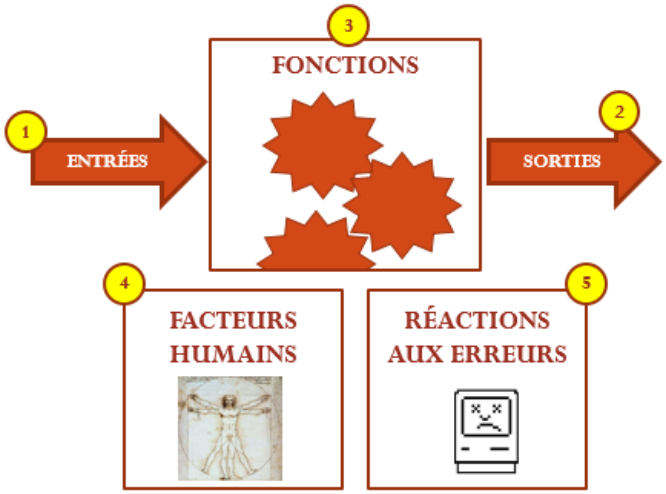


Figure 3 - Représentation schématique des spécifications fonctionnelles

Fonctions, besoins et contraintes du système

Le système doit pouvoir acquérir l'électrocardiogramme d'un individu, l'amplifier, le filtrer, puis l'envoyer de manière sans-fil à un téléphone intelligent. Ce téléphone doit ensuite afficher l'électrocardiogramme en temps réel. Le système complet doit être portable et peu cher, afin qu'il soit accessible au grand public. Ceci permettrait de le commercialiser à grande échelle. L'interface graphique sur le téléphone intelligent doit être simple et intuitive pour que n'importe quel utilisateur puisse l'utiliser.

Entrées du système

Le système comporte plusieurs entrées; certaines sont désirables et d'autres non. La première entrée est évidemment l'électrocardiogramme de l'individu. Il s'agit du signal à envoyer au téléphone. Cependant, ce signal est bruité. Ce bruit constitue une entrée indésirable du système, qui devra être filtrée. Finalement, l'utilisateur peut choisir certains paramètres de configuration (démarrer, arrêter, se connecter, etc.), ce qui constitue la dernière entrée.

Sorties du système

Le système ne possède qu'une seule sortie. Il s'agit de l'électrocardiogramme affiche sur l'écran du téléphone. L'électrocardiogramme obtenu est après filtrage et amplification, afin de minimiser le bruit indésirable.

Facteurs humains

Les facteurs humains entrants en compte sont les commandes de l'utilisateur sur le téléphone par le biais de l'interface graphique. L'utilisateur doit pouvoir se connecter au système en appuyant sur un bouton. Il doit aussi pouvoir démarrer et arrêter le système avec un bouton sur l'interface graphique.

Réactions aux erreurs

Plusieurs erreurs peuvent survenir, notamment la réception de mauvaises données et la perte de connexion. Dans le cas de la réception de mauvaises données, celles-ci sont simplement supprimées. Dans le cas de la perte de connexion avec le téléphone, le système d'acquisition revient simplement dans son état initial, et attend une nouvelle connexion.

Spécifications techniques

A partir de cela, on peut en déduire des spécifications techniques minimales, que le système doit respecter.

Tout d'abord, on cherche à éliminer le bruit sur l'ECG. Il est donc logique de définir une spécification sur cela. On considère que le bruit est négligeable lorsqu'il représente moins de 10% du signal. Cela se traduit par un ratio signal sur bruit minimal de 20 dB.

Ensuite, le système doit être portable. Plusieurs spécifications peuvent être dérivées de cela :

- Le système ne doit pas mesurer plus de 10 cm x 5 cm x 4 cm. Ces dimensions permettent de le mettre facilement dans une poche.
- Le système ne doit pas consommer plus de 10 mW au repos et 25 mW en transmission. Cela assure de ne pas avoir à utiliser une batterie avec une grosse capacité, ce qui prendrait de l'espace.

- Le système doit avoir une durée de vie d'au moins 24 heures sur une batterie. Ce chiffre vient du fait qu'il y a 24 heures dans une journée, et que donc, chaque soir, l'individu peut recharger ou changer la batterie du système.
- Le système doit avoir une bande passante allant de 0,2 Hz à 150 Hz. Cela permet d'éliminer la composante DC, de garder toutes les fréquences composant l'ECG, et de minimiser le bruit.
- Le système doit échantillonner l'ECG à une fréquence d'au moins 300 Hz. Ce nombre vient du théorème de Nyquist-Shannon qui stipule que la fréquence d'échantillonnage doit au moins être le double de la fréquence maximale du signal (150 Hz ici). Cela permet de s'assurer d'obtenir un signal fidèle a l'ECG.
- Une autre spécification est le prix. Le système doit en effet être abordable pour qu'il soit accessible au grand public. Ainsi, le prix doit être inférieur à 100$.

Design préliminaire

Exploration des approches de résolution

Maintenant que les spécifications fonctionnelles ont été définies. On peut en dériver un diagramme de flot de données (DFD). Le DFD permet de décomposer le problème en différentes étapes ce qui permet d'explorer des méthodes de résolution. En effet, l'ECG passe par plusieurs étapes avant l'affichage. Tout d'abord, il est acquis par les électrodes, puis le signal est amplifié et filtré. Il est ensuite échantillonné et envoyer sans fil a un téléphone intelligent, qui l'affiche en temps réel sur l'écran. On en déduit donc le diagramme de flot de données :

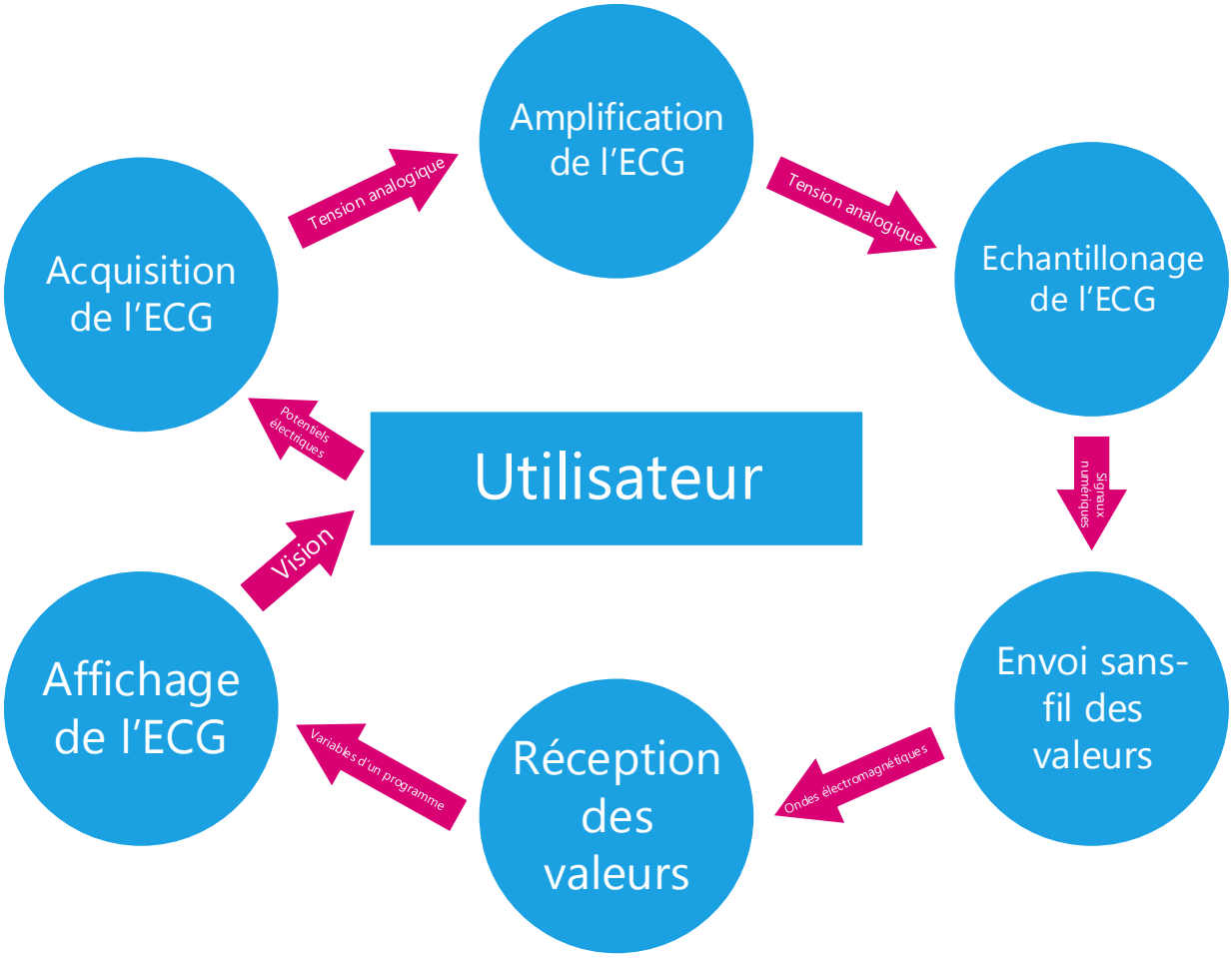


Figure 4 - Diagramme de flot de données

Le DFD est linéaire, comme c'est souvent le cas en traitement de signal. Comme on a décomposé le problème en plusieurs étapes, on peut trouver une architecture pour résoudre le problème. Chaque étape (cercle bleu sur le DFD) peut être réalisée par un module.

Tout d'abord, l'étape d'acquisition de l'ECG se fait par des électrodes. Cela permet d'obtenir l'ECG sous forme de tension de quelques millivolts. L'étape d'amplification se fait grâce à un circuit d'amplification analogique. Cette amplification se fera par des amplificateurs d'instrumentation et des amplificateurs opérationnels. Cela permet d'obtenir en sortie un ECG filtré d'une amplitude de l'ordre du volt.

Le signal est toujours analogique, ce qui n'est pas pratique pour la transmission. Le signal est donc numérisé à l'étape d'échantillonnage de l'ECG. On réalise cela grâce à un convertisseur analogique/numérique. Les échantillons sont ensuite envoyés de manière sans-fil. Cela est réalisé par un microcontrôleur avec antenne *Bluetooth Low Energy* intégrée. Ce choix s'explique par plusieurs raisons :

- Le microcontrôleur est peu cher, reconfigurable, et son développement se passe à haut niveau. Il s'agit donc d'un bon choix pour le système car il permet le respect des spécifications de prix et il permet un développement rapide.

- Le protocole *Bluetooth Low Energy* est adopté car il permet une très faible consommation, tout en garantissant un débit de données suffisant pour notre application. Cela permettra donc de respecter les spécifications de consommations, de durée de vie et d'échantillonnage. De plus, ce protocole est supporté par plusieurs téléphones. D'autres protocoles permettent de réaliser l'envoi des données, comme le protocole *Wi-Fi*, *Bluetooth* et *Zigbee*. Cependant, le protocole *Wi-fi* consomme beaucoup trop notre application. De plus, *Zigbee* n'est pas supporté par les téléphones intelligents. Finalement, le protocole *Bluetooth* consomme plus que le protocole *Bluetooth Low Energy*. Le protocole *Bluetooth Low Energy* est donc le meilleur choix pour notre application.

La réception des valeurs se fait par une antenne *Bluetooth* intégrée au téléphone intelligent. Une application tourne sur le téléphone intelligent pour recevoir l'ECG et l'afficher en temps réel.

Architecture du système

On obtient donc l'architecture suivante, qui nous permettra de répondre aux spécifications demandées :

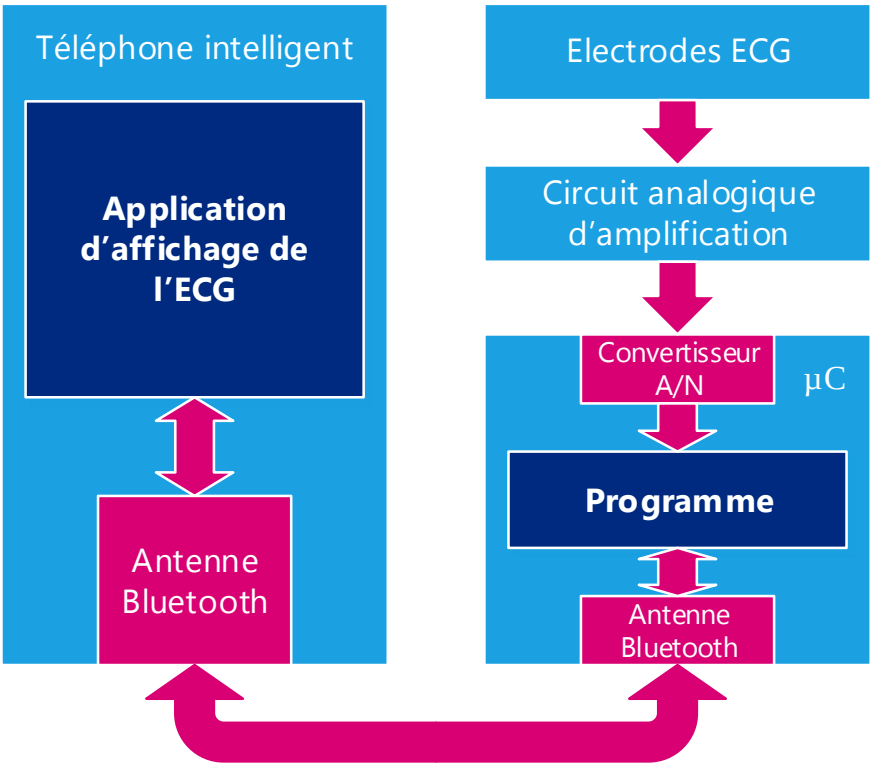


Figure 5 - Architecture du système

Étude de praticabilité et justification de la solution retenue

La solution obtenue permettra de respecter les spécifications techniques. En effet :

- Toutes les composantes utilisées sont peu chères. Cela permet de respecter la spécification sur le prix.

- Toutes les composantes peuvent fonctionner sur de basses tensions. Cela permettra de respecter les contraintes sur la consommation et la durée de vie. Le protocole choisi*, Bluetooth Low Energy*, est aussi optimisé pour une consommation très faible, tout en ayant un débit de données suffisant pour un échantillonnage à 300 Hz.
- Les composantes sont très petites, ce qui permettra de respecter les contraintes sur les dimensions.
- Le circuit analogique permet d'obtenir un signal propre et avec très peu de bruit, permettant de respecter les spécifications d'intégrité de signal.

Le téléphone intelligent qu'on choisit est le téléphone *Nexus 4* par *Google*. Cela s'explique par plusieurs raisons :

- Le *Nexus 4* supporte le protocole *Bluetooth Low Energy*, permettant une basse consommation.
- Le *Nexus 4* tourne le système d'exploitation *Android*. Il y a donc des outils matures pour le développement pour ce système d'exploitation. Il y a aussi beaucoup de tutoriaux ce qui permettra un apprentissage rapide du développement d'application.
- Le système d'exploitation *Android* a la plus grande part de marché parmi tous les systèmes d'exploitation pour téléphone intelligent.
- Le *Nexus 4* était déjà en ma possession avant le début du projet. Cela réduit donc énormément les coûts du projet.

Design détaillé

Maintenant que l'architecture a été définie, on peut procéder au design détaillé. Cela consiste à concevoir chaque module au niveau des composantes, du code, etc.

Choix des électrodes

Tout d'abord, il est nécessaire de définir le type d'électrodes à utiliser. On sait que:

- Le système doit être peu cher. Il faut donc que les électrodes le soient aussi. Les électrodes jetables sont peu dispendieuses, et sont donc les meilleures candidates pour notre prototype.
- Le signal doit être de bonne qualité. Il faut éviter le plus possible d'avoir du bruit. Les électrodes doivent donc avoir un bon contact. Les électrodes avec gel donnent le meilleur résultat, car elles permettent de diminuer l'impédance de la peau. Elles constituent donc les meilleures candidates pour le prototype.
- Le patient portera le système pendant une durée de plusieurs heures (voir un jour). Il faut donc que les électrodes tiennent longtemps sur le patient et qu'elles ne s'enlèvent pas facilement. Les électrodes collantes sont donc un choix adapté.

Suite à cette réflexion, les électrodes les plus adaptées sont des électrodes collantes et jetables avec gel conducteur. Aussi, on choisit d'utiliser des câbles *shieldés* afin de diminuer le bruit accumulé sur l'ECG.

Conception du circuit d'amplification analogique

Tout d'abord il faut décider sur quelle alimentation le système doit être. Étant donné qu'il doit être portable, l'alimentation doit évidemment être une batterie. Comme on recherche une petite taille et une basse consommation, les piles CR2032 sont tout à fait adaptées. Elles fournissent 3V et ont une

capacité de 225 mAh, tout en étant de la taille d'une pièce de monnaie. Ainsi, notre système utilisera une pile CR2032 comme alimentation.

Le circuit d'amplification analogique doit respecter les contraintes suivantes :

- Il doit avoir un gain suffisant. Étant donné que le signal venant des électrodes est de un millivolt, et que le signal de sortie est d'amplitude un volt, il faut un gain de G = 1000 V/V = 60 dB.
- Il doit éliminer le DC et avoir une fréquence de coupure de 150 Hz.
- Avoir une très forte impédance d'entrée afin d'obtenir un signal de bonne qualité venant des électrodes.
- Rejeter le mode commun entre les deux électrodes différentielles.
- Fonctionner sur une alimentation de 3V.

Pour obtenir une forte impédance d'entrée et rejeter le mode commun, on utilise un amplificateur d'instrumentation. Un des seuls pouvant fonctionner sur une alimentation de 3V est l'INA122 de *Texas Instruments*. Il permet de faire un gain selon la formule suivante :

$$G = 5 + \frac{200\ k\Omega}{R_g}$$

Avec Rg la résistance entre les pates 1 et 8. Selon la fiche technique, plus le gain est élevé, moins l'INA122 rejette le mode commun. Ainsi, il est préférable d'effectuer un gain faible sur amplificateur d'instrumentation, puis faire un autre gain ultérieurement. Ainsi, on cherche à avoir un gain de 14,1. Cela nous donne Rg = 22 kΩ. On utilise un amplificateur opérationnel en montage amplificateur non inverseur pour effectuer le deuxième gain. Le deuxième gain est :

$$G_2 = \frac{1000}{14{,}1} = 71 = 1 + \frac{R_1}{R_2} = 1 + \frac{300k\Omega}{4{,}7k\Omega}$$

On obtient ainsi notre gain de 60 dB, l'impédance d'entrée élevée et un bon rejet du mode commun. Pour les amplificateurs opérationnels, on utilise les TLC277 (deux amplificateurs par puce) à cause de leur haute précision, et de leur basse tension d'alimentation minimale.

Pour les fréquences de coupure, on utilise des filtres passifs RC afin de réduire le cout. On peut déduire les valeurs des condensateurs et résistances à partir des fréquences de coupure désirées.

<u>Coupure du DC :</u>

$$f_{c1} = \frac{1}{2\pi RC} = \frac{1}{2\pi \times 1M\Omega \times 1\mu F} = 0{,}16\ Hz$$

<u>Coupure des hautes fréquences :</u>

$$f_{c2} = \frac{1}{2\pi RC} = \frac{1}{2\pi \times 22k\Omega \times 47nF} = 154\ Hz$$

On obtient ainsi les bonnes fréquences de coupure.

Finalement, le signal ECG a une partie négative et une partie positive. Si la référence était à la masse, alors on perdrait la partie négative du signal. Pour éviter cela, on fixe la référence à Vdd/2 = 1,5 V. Pour cela, on utilise un diviseur de tension et un amplificateur opérationnel en suiveur afin d'avoir une impédance très faible sur la tension de référence.

Ainsi le circuit obtenu est le suivant :

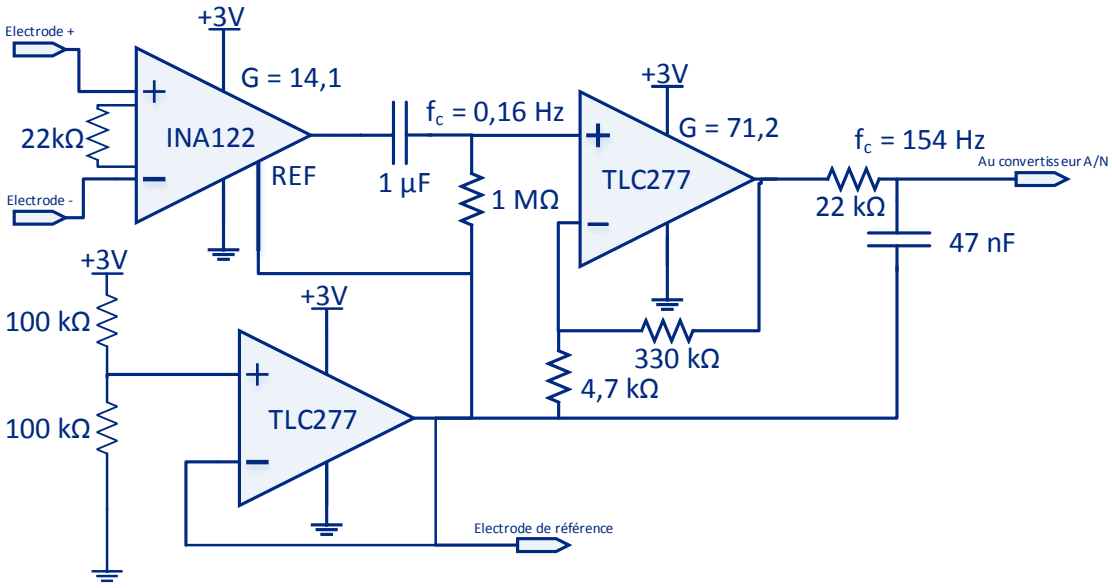


Figure 6 - Schéma du circuit d'amplification analogique

Avec ce circuit et les électrodes, on regarde avec un oscilloscope si on peut obtenir un signal en sortie. Le graphique obtenu est le suivant :

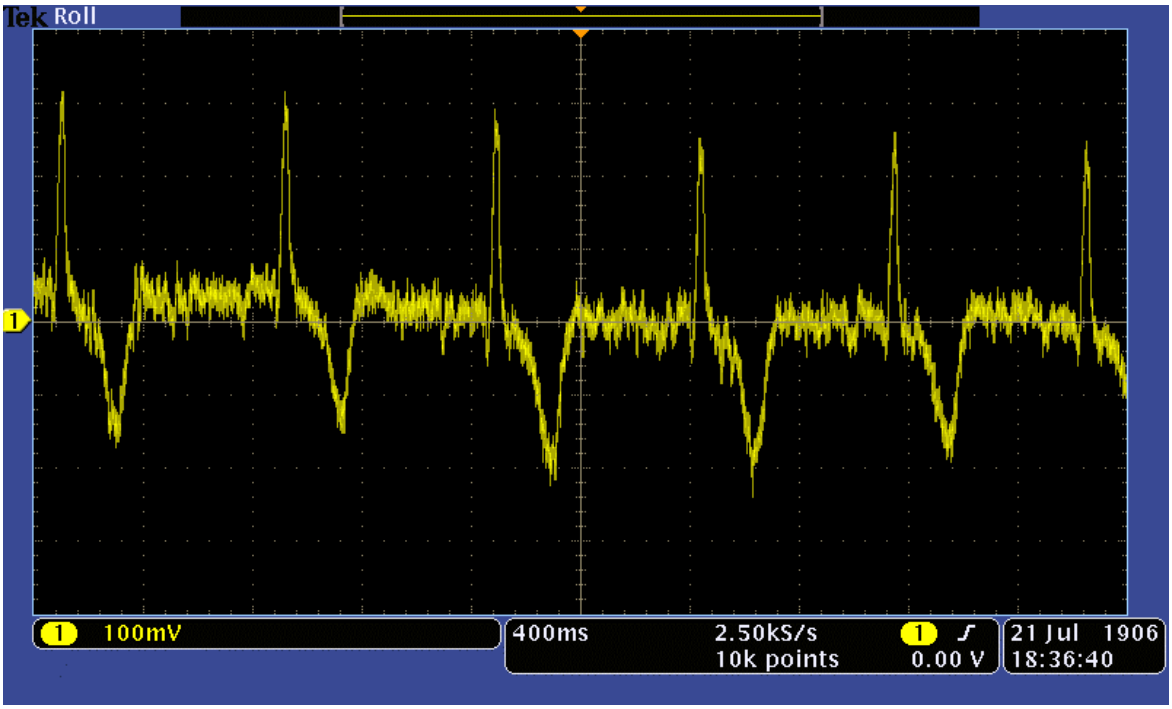


Figure 7 - ECG sur l'oscilloscope

Le signal obtenu ressemble à un ECG (les différentes vagues composants un ECG sont présentes). L'amplitude du signal est aussi adaptée. Ce signal peut donc être échantillonnée par le convertisseur analogique/numérique.

Choix du microcontrôleur et développement du code

Il faut tout d'abord choisir le microcontrôleur. Afin de minimiser le cout, il faudrait un microcontrôleur avec un convertisseur analogique/numérique et une antenne *Bluetooth Low Energy* intégrés. Le microcontrôleur doit aussi pouvoir fonctionner sur une alimentation de 3V. Le RFduino, utilisant le RFD22102 de *Nordic Semiconductor* répond à toutes ces exigences. De plus, il se programme avec l'environnement Arduino, connu pour sa simplicité d'utilisation. Le temps de développement sera donc réduit. Il possède une antenne et une librairie *Bluetooth Low Energy*, ainsi que sept convertisseurs analogique/numérique de 12 bits.

Le code simplifié du microcontrôleur est montré ci-dessous :

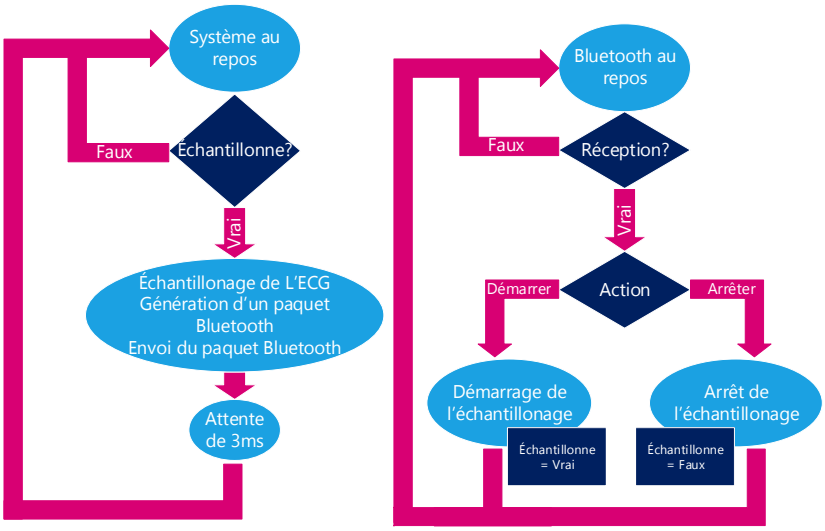


Figure 8 - Code simplifié du microcontrôleur

Lorsque le microcontrôleur est repos, il se met dans un état de faible consommation. S'il reçoit l'ordre de démarrage de la part du téléphone, il se met à échantillonner l'ECG, générer des paquets *Bluetooth Low Energy*, et les envoyer a une fréquence de 300 Hz. S'il reçoit l'ordre d'arrêt, il se remet dans l'état de repos pour diminuer sa consommation en courant. Le code final du microcontrôleur est montré en annexe.

Développement de l'application Android

Pour l'application, on utilise le kit de développement logiciel *Android*. Étant donné que l'ECG doit être tracé en temps réel, on utilise une librairie à source ouverte appelée *AndroidPlot* qui permet de travailler à plus haut niveau et donc de réduire le temps de développement.

Le code simplifié de l'application *Android* est montré ci-dessous :

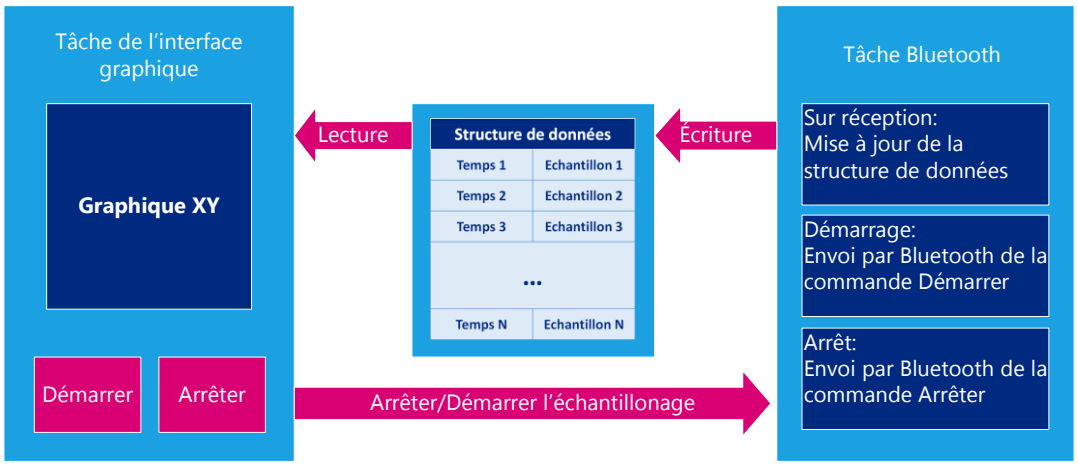


Figure 9 - Code simplifié de l'application Android

L'application est composée de deux taches (thread), une pour l'interface graphique et l'autre pour le *Bluetooth*. Lorsqu'il y a réception de paquet de la part du microcontrôleur, la tache *Bluetooth* vient lire le paquet et mettre à jour la structure de données. La tâche de l'interface graphique utilise ensuite les données contenues dans la structure de données pour générer son graphique. Sur l'interface graphique, il y a deux boutons « Démarrer » et « Arrêter ». Lorsque que le bouton « Démarrer » est appuyé, la tache *Bluetooth* envoie un paquet au microcontrôleur lui demandant de commencer l'échantillonnage et la transmission. Lorsque que le bouton « Arrêter » est appuyé, la tache *Bluetooth* envoie un paquet au microcontrôleur lui demandant d'arrêter la transmission et de se mettre dans un mode basse consommation en attendant un nouvel ordre.

Le code final de l'application *Android* est montré en annexe.

Réalisation du prototype

Tous les modules ont été conçus. Le système est donc assemblé sur *breadboard* pour voir les résultats obtenus. Le design sur *breadboard* permet de facilement changer le design au besoin, et donc de rapidement développement une solution fonctionnelle. Le premier prototype est montre à la figure suivante :

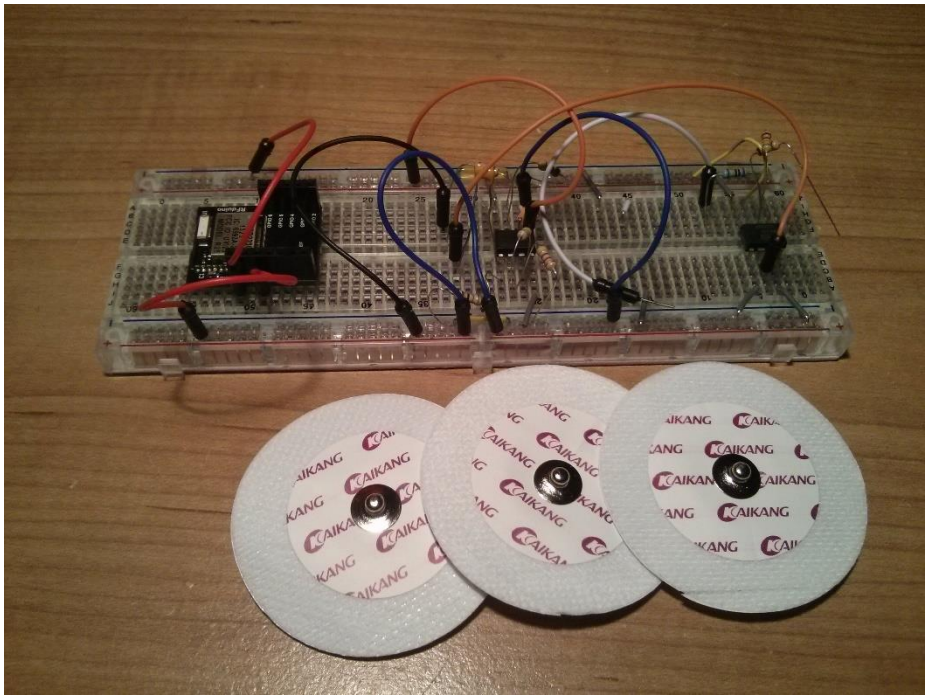


Figure 10 - Photographie du premier prototype

Résultats préliminaires

Avec ce prototype, on obtient l'électrocardiogramme suivant :

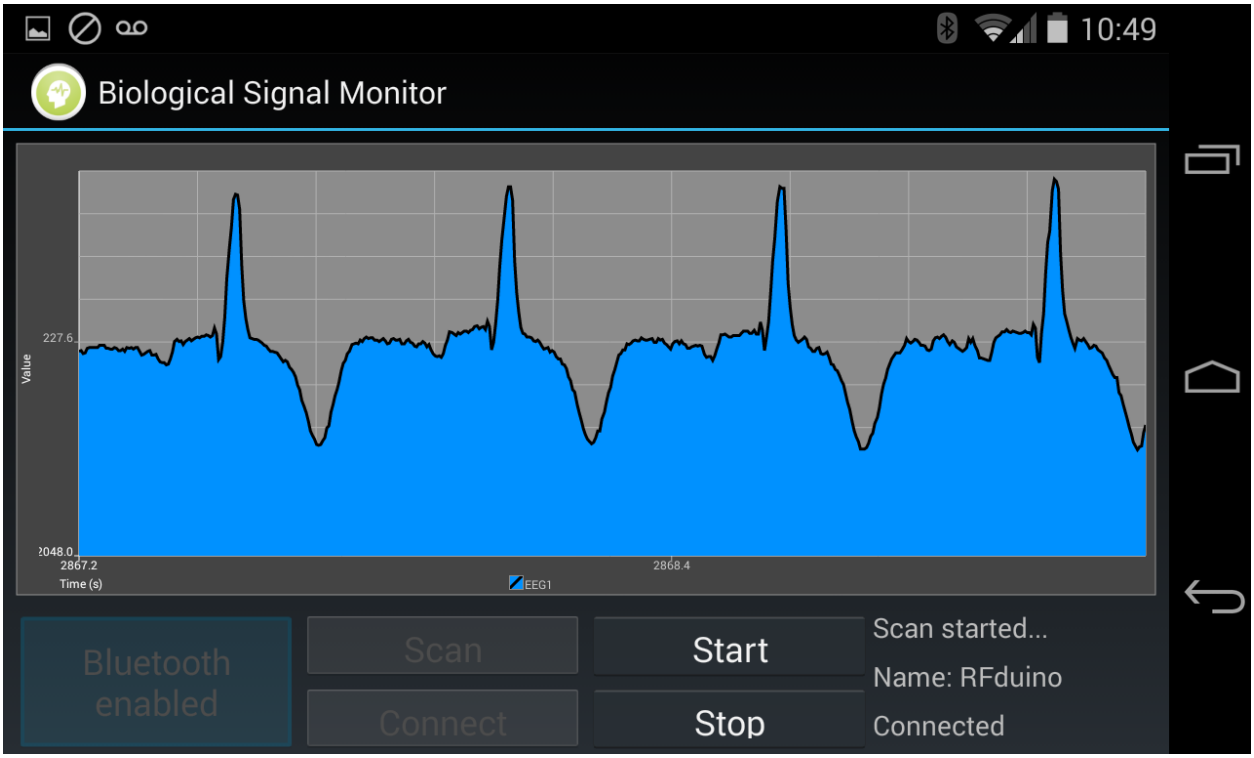


Figure 11 - Capture d'écran du téléphone lors de la prise de l'ECG

Qualitativement, cet ECG a la même allure que les ECG obtenus professionnellement. Cela nous donne une bonne indication que le design est fonctionnel. Il faut cependant vérifier cela à l'étape de *Validation*, qui est détaillée dans ce rapport.

Itérations du prototype

La première itération du design est fonctionnelle. Cependant, elle ne respecte pas les spécifications de dimension. On réalise donc une deuxième itération du prototype, en le soudant sur *PerfBoard*. Le nouveau prototype obtenu est le suivant :

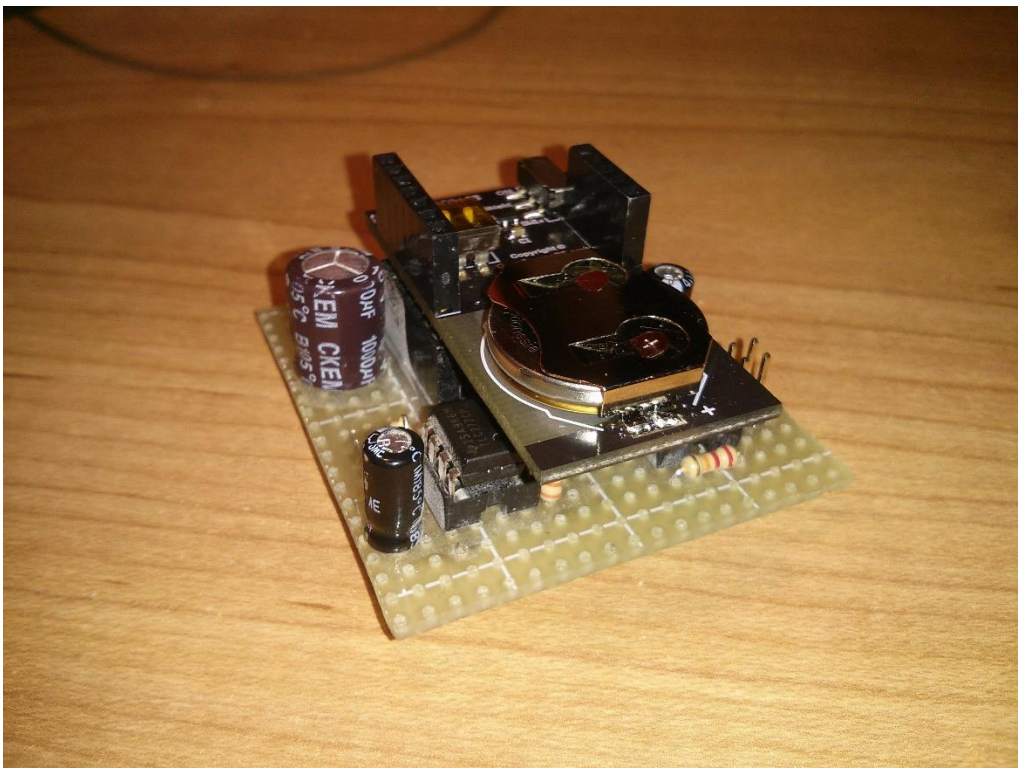


Figure 12 - Photographie du prototype final

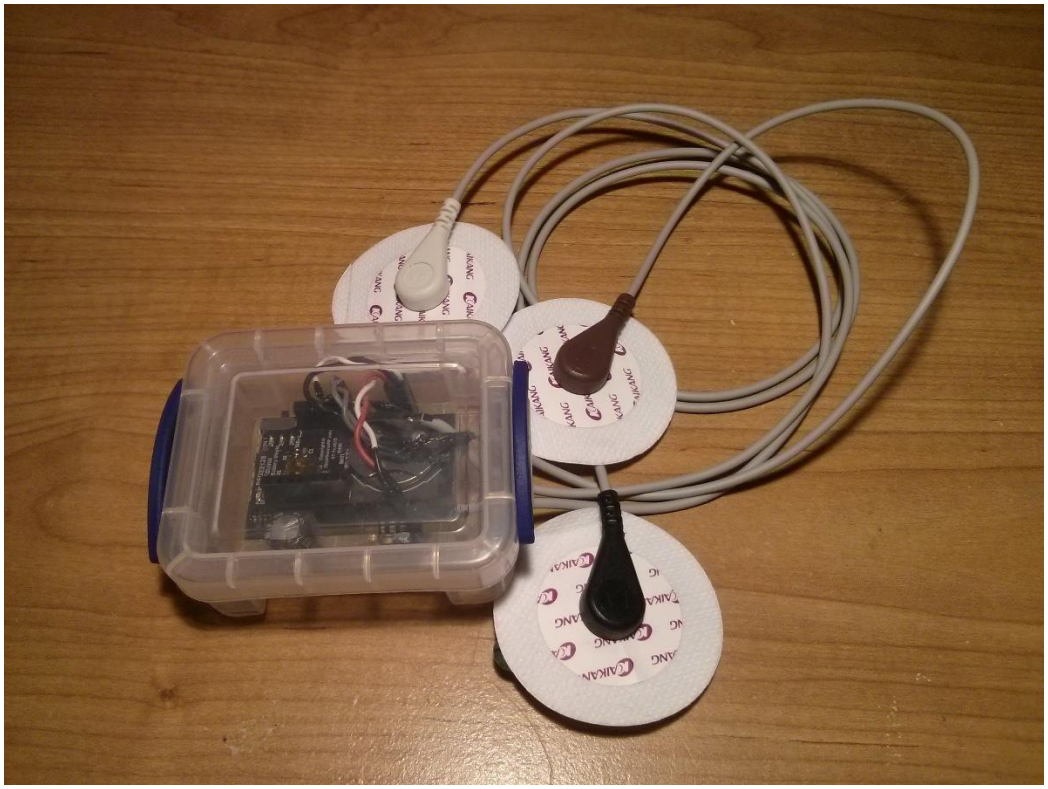


Figure 13 - Photograpie du prototype final dans sa boite

Ce prototype est beaucoup plus compact que le premier. Il mesure 7 cm x 5 cm x 3 cm ce qui le rend très portable. Il respecte donc les spécifications de dimensions, et peut facilement être mis dans une poche. On effectue les tests de validation sur ce nouveau prototype (ces tests sont expliqués dans la section *Validation*).

Validation et Vérification

Une étape très importante lors du développement du système a été l'étape de la validation. Sans cette étape, on ne peut s'assurer que le projet est robuste et qu'il respecte les spécifications. C'est ainsi que environ la moitié du temps a été passé à l'étape de validation et vérification.
Pour cette étape, les tests sont d'abord effectués sur des sous-modules du système. Cela permet d'isoler les problèmes et les régler plus facilement. Des tests sont ensuite effectués sur l'ensemble du système afin de régler les derniers problèmes liés à l'intégration. De cette façon, on s'assure d'obtenir un système extrêmement fiable.

Tout d'abord, sur la partie d'amplification analogique, on effectue un test de réponse en fréquence (on cherche à obtenir le diagramme de Bode). Pour cela, on applique une différence de tension entre les deux entrées égales à un sinus de fréquence et d'amplitudes connues. On relève l'amplitude en sortie et on obtient ainsi le gain. En effectuant cela sur des fréquences allant de 0,01 Hz à 10000 Hz, on obtient le diagramme de Bode suivant :

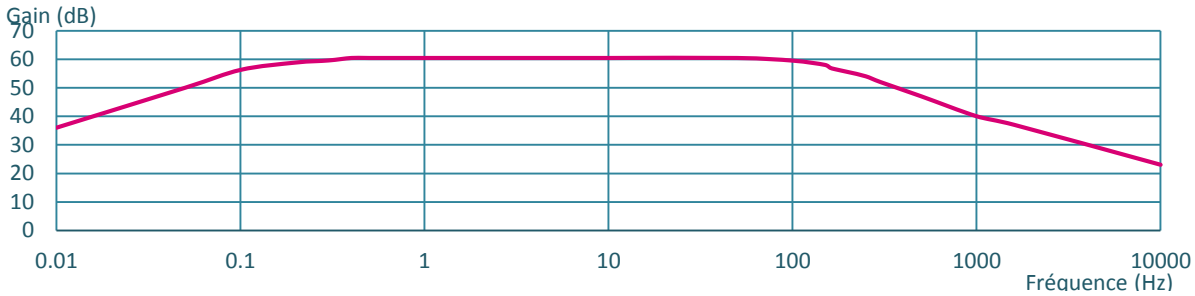


Figure 14 - Diagramme de Bode

On voit que les fréquences de coupures respectent les spécifications. Le gain aussi est bon (60 dB, soit 1000 V/V dans la bande passante).

On effectue aussi un test de bruit. On met en entrée un sinus d'une fréquence de 2 Hz et d'amplitude 1 mV. Avec une transformée de Fourier Rapide obtenue à l'oscilloscope, on regarde les différentes fréquences qui composent le signal en sortie. On obtient le rapport signal sur bruit de la manière suivante :

$$R_{Signal/Bruit} = \frac{A_{signal}}{A_{Bruit}} = \frac{1{,}045\ V}{0{,}088\ V} = 11{,}875 = 21{,}5\ dB$$

Cela respecte donc les spécifications initiales. Le signal contient un niveau raisonnable de bruit. Le signal en sortie représentera donc l'ECG avec fidélité.

On effectue ensuite des tests sur le code du microcontrôleur. Pour cela, on branche le microcontrôleur sur l'ordinateur par USB. Cela nous permet d'utiliser l'UART pour afficher des valeurs sur la console de l'ordinateur. Ceci est donc un outil très utile pour déterminer le code. La méthode pour utiliser l'UART est *Serial.println(string message)*. On fixe le taux de transfert à 115200 bauds afin que la communication UART ne prenne pas beaucoup de temps. Cela est très important car pour notre application, le temps d'exécution est critique (le système doit échantillonner à une fréquence d'au moins 300 Hz).

En utilisant cette méthode, on insert la fonction *Serial.println* à des endroits stratégiques dans le code (appel de fonction, changement d'état *Bluetooth*, valeur des échantillons). On met en entrée un signal de tension connue. On lance le programme et on obtient le texte suivant sur la console :

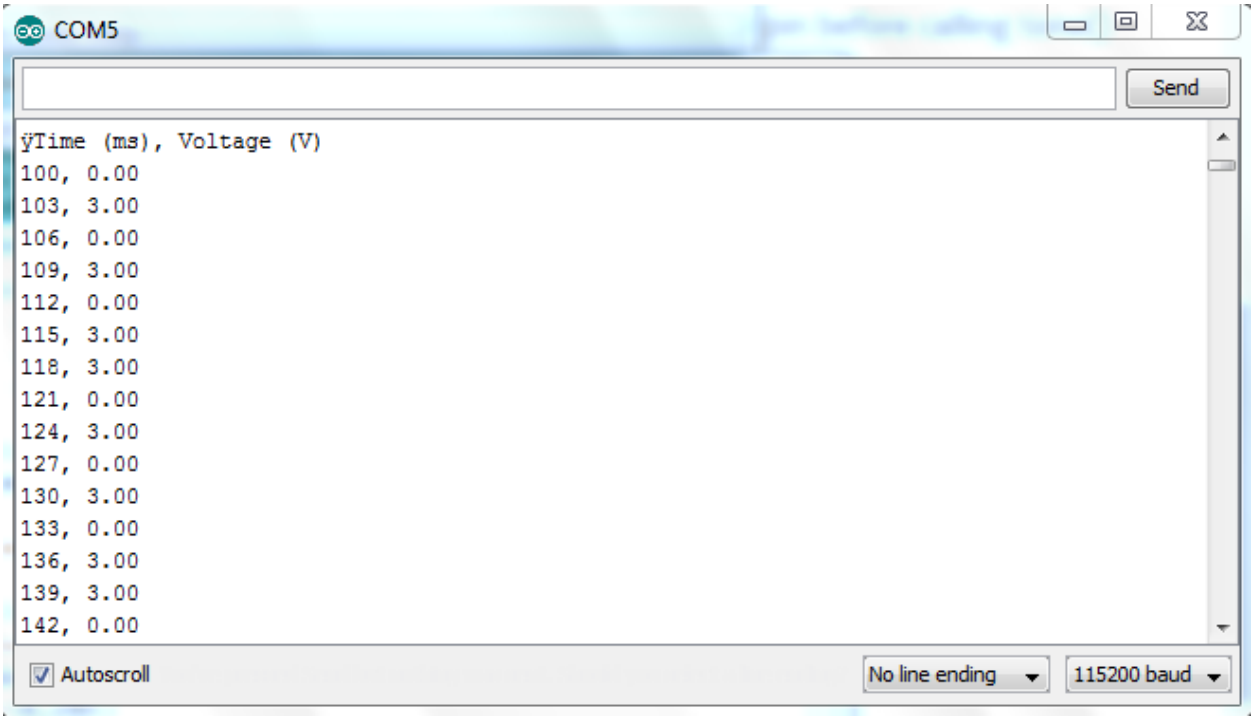


Figure 15 - Console de déverminage pour le microcontrôleur

On observe ici les valeurs des échantillons avec une onde carrée de fréquence 150 Hz en entrée. On s'assure ainsi de la robustesse et du bon fonctionnement du système. Le temps entre chaque échantillon est de 3 millisecondes. On obtient ainsi une fréquence d'échantillonnage de :

$$f = \frac{1}{T} = 333\ Hz$$

Cela est donc supérieur à 300 Hz, et la spécification d'échantillonnage est donc respectée.

De la même façon, on détermine le code de l'application Android. Une fonction appelée *System.out.println* permet d'afficher des messages sur la console du kit de développement (console appelée *LogCat*). En insérant cette fonction à chaque changement d'état de connexion et à chaque réception de paquet, on peut s'assurer du bon fonctionnement de l'application. Dans la figure suivante, on observe la réception de chaque paquet :

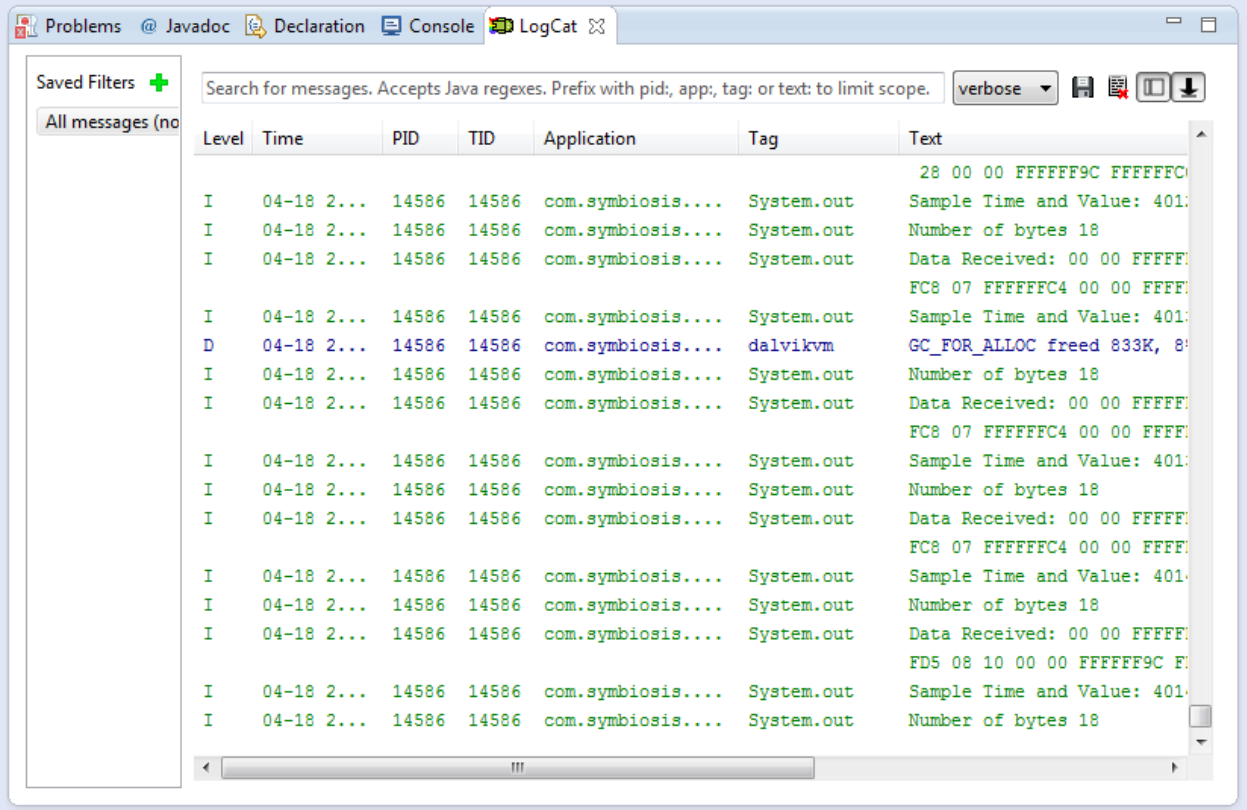


Figure 16 - Capture d'écran de LogCat

En utilisant ceci, on s'assure que l'application est sans bug. On essaie aussi des cas non triviaux, comme le cas où l'utilisateur quitte l'application et la relance. Après avoir effectué ce test, on est sûr que l'application et la connexion *Bluetooth Low Energy* sont robustes.

Maintenant que tous les sous modules ont été validés, on procède à des tests de validation au niveau système. On monte donc le système intégralement avec électrodes, et on obtient les ECG suivants :

Figure 17 - ECG obtenu au repos

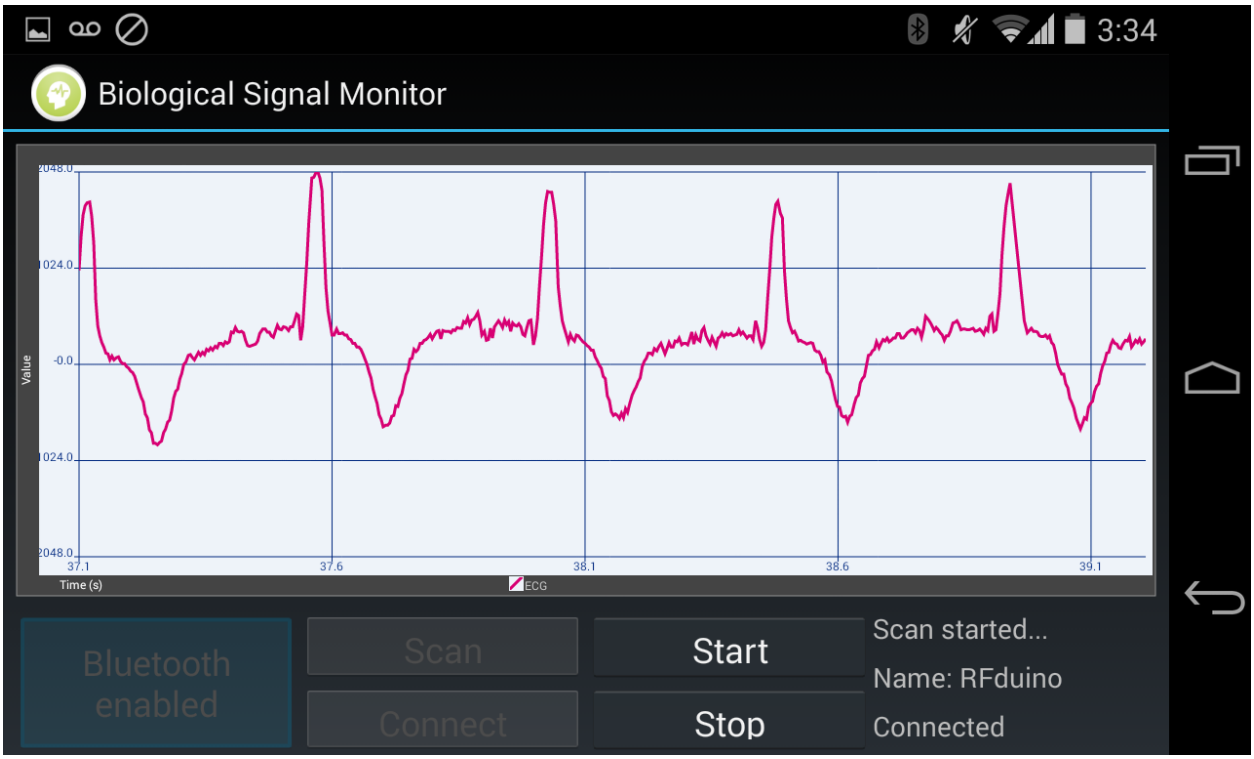
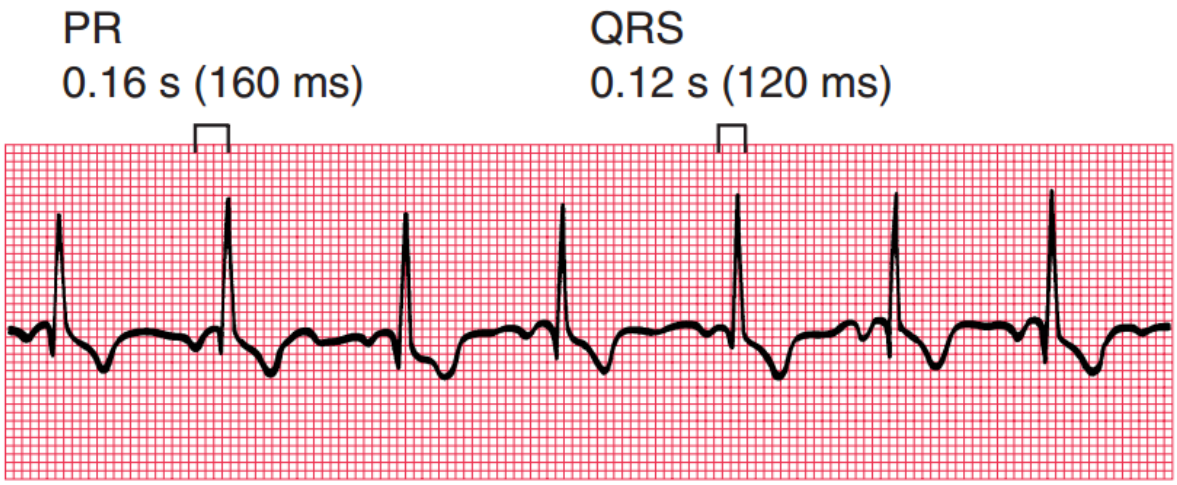


Figure 18 - ECG obtenu après un effort physique

Idéalement, il faudrait comparer avec un système professionnel. Cela permettrait une comparaison quantitative. Cependant, n'ayant pas accès à un tel système, on doit effectuer cette comparaison avec un ECG de référence, de manière qualitative. Voici l'ECG en question, obtenu lors de la revue de documentation :



[3]Figure 19 - ECG de référence

En comparant la Figure 17 avec la Figure 19 (ECG de référence), on remarque que les deux ont la même allure. En effet, qualitativement parlant, ils possèdent tous les deux un complexe QRS (pic fin), suivi d'un creux puis d'un plateau (avec quelques bosses). Cela indique que l'ECG obtenu avec le système développé a du sens d'un point de vue qualitatif.

De plus, si on compare l'ECG obtenu au repos (Figure 17) avec celui obtenu après un effort (Figure 18), on voit qu'ils ont la même allure. Cependant, la fréquence obtenue au repos est beaucoup plus faible que la fréquence obtenue après un effort. En effet, la fréquence au repos est de 75 battements par minute, alors que la fréquence après un effort physique est de 92 battements par minute. Ceci est normal (le cœur bat plus vite lors d'un effort physique) ce qui nous suggère que les ECG obtenus sont justes.

On effectue d'autres tests pour vérifier que les spécifications ont été respectées. Tout d'abord, on effectue un test de consommation. Pour cela on utilise un ampèremètre et l'on mesure le courant tiré de la batterie. On multiplie ensuite par la tension d'alimentation pour obtenir la consommation en puissance. On obtient 2,06 mA x 3V = 6,2 mW au repos, lorsqu'il n'y a aucune transmission, et 4,07 mA x 3V = 12,2 mW lors des transmissions. Cela respecte donc les spécifications initiales. A partir de cela on peut on déduire la durée de vie. Le système tourne sur une batterie CR2032 de capacité 225 mAh. Ainsi, la durée de vie se situe entre :

$$D_{min} = \frac{225\ mAh}{4{,}07\ mA} = 55\ heures$$

[3] Auteur inconnu, «What is the ECG about». [En ligne]. Disponible: https://www.us.elsevierhealth.com/media/us/samplechapters/9780443068171/ECG%20Made%20Easy%201-40.pdf. [Accès le 18 Avril 2014].

$$D_{max} = \frac{225\ mAh}{2{,}06\ mA} = 109\ heures$$

La durée de vie est donc incluse entre 55 heures et 109 heures, dépendamment de la proportion de temps où le système est en mode transmission.

On calcule aussi le prix total du système (téléphone exclu) :

Composant	Quantité	Coût unitaire ($)	Coût total ($)
RFD22301	1	25,38	25,38
INA122PA	1	7,58	7,58
TLC277CP	1	3,19	3,19
Résistances	7	0,0014	0,01
Condensateurs	4	0,0025	0,01
Pile CR2032	1	0,40	0,40
Etui à CR2032	1	0,32	0,32
Electrodes	3	0,658	1,97
Câbles	3	2,53	7,59
Total	-	-	46,45

Tableau 1 - Calcul du prix total

On obtient un cout total de 46,45$. Cela est un prix raisonnable, ce qui permet au système d'être abordable. Le dispositif pourrait donc être déployé a grande échelle car son prix le rend accessible à tous les patients. Les spécifications ont donc été respectées au niveau du prix. D'autres tests ont été effectués. Les résultats sont compilés dans le tableau ci-dessous :

Critère	Résultat
Consommation	6,2 mW au repos 12,2 mW en transmis-sion
Durée de vie	55 ~ 109 heures
Dimensions	7 cm x 5 cm x 3 cm
Bande passante	0,16 Hz – 154 Hz
Echantillonnage	~ 300 Hz
Coût	46,45 $

Tableau 2 - Tableau des spécifications

Limite du projet

Le projet est fonctionnel, portatif, et respecte toutes les spécifications. Il possède cependant certaines limites. Tout d'abord, il ne fournit qu'un seul canal. Cela peut être amélioré en mettant en parallèle plusieurs modules analogiques d'amplification. Le microcontrôleur peut ainsi supporter un maximum de 7 canaux en parallèle. Cela peut même être augmente en utilisant des convertisseurs analogique/numérique en série suivant le protocole SPI.

De plus, le système utilise le protocole *Bluetooth Low Energy*. Seul 5% des téléphones intelligents supportent actuellement ce protocole. Il faudrait donc utiliser le protocole *Bluetooth*, qui est utilisé par la quasi-totalité des téléphones, pour pénétrer plus facilement le marché avec un tel dispositif. Le protocole *Bluetooth Low Energy* est actuellement en voie d'adoption, et devrait représenter une plus grande part du marché d'ici quelques années.

Aussi, le système a une moins bonne précision que les systèmes professionnels. Cela est dû au bruit généré par le microcontrôleur sur le circuit analogique. En effet, lors de la transmission, le microcontrôleur utilise beaucoup de courant ce que fait varier la tension d'alimentation, ce qui crée du bruit sur le signal analogique. Il faudrait donc séparer les alimentations digitales et analogiques. Cela diminuerait grandement le bruit du système mais ajouterait des composantes, augmentant ainsi le prix.

Finalement, le système pourrait être bien plus compact. En réalisant un PCB avec des composants montés en surface, on pourrait atteindre une surface équivalente à celle d'une pièce de monnaie.

Apprentissage réalisé durant le projet

Ce projet fut particulièrement intéressant, car il m'a permis de m'initier à une multitude de domaines. Cela inclue :

- L'acquisition de signaux biologiques. Avant ce projet, je n'avais aucune connaissance dans ce domaine. Je comprends maintenant les différentes étapes nécessaires afin d'obtenir un ECG propre, quelles sont les principales difficultés pour son obtention (exemple : interférences du 60/50 Hz, rejet du mode commun, etc.).
- L'amplification analogique. J'ai eu l'occasion d'utiliser des amplificateurs opérationnels lors de mon projet intégrateur deux. Cependant, je n'avais jamais utilisé d'amplificateur d'instrumentation (AI). Cela a donc été nouveau pour moi et m'a donner de nouvelles connaissances, notamment sur les critères de performance des amplificateurs d'instrumentation, et les différentes applications des AI.
- La programmation micro-contrôleur. N'ayant pas encore suivi le cours ELE3312, l'apprentissage de la programmation d'un microcontrôleur a été nouveau pour moi. Cependant, j'avais déjà des bases en programmation grâce a des cours suivis, ce qui m'a beaucoup aidé.
- La programmation Android. Grace aux tutoriaux trouvés sur internet et en utilisant le kit de développement logiciel Android, j'ai été en mesure de développer ma première application Android. Cela se fait par Java et XML, que je ne connaissais pas avant.
- Le protocole *Bluetooth Low Energy*. Le microcontrôleur et le téléphone doivent communiquer en utilisant ce protocole. Il a donc fallu que je comprenne le fonctionnement de ce protocole et j'ai donc lu la fiche technique mise à disposition par le *Bluetooth Special Interest Group*.

Le projet a donc été l'occasion d'essayer pleins de domaines. Cela a constitué une difficulté supplémentaire, mais a finalement été très enrichissant vu les connaissances acquises. Finalement, j'ai compris l'importance de définir des spécifications et de planifier afin de mener un projet à bien.

Conclusion

Pour conclure, un système d'acquisition sans-fil d'électrocardiogramme a été conçu. Comme son nom l'indique, il permet d'acquérir l'électrocardiogramme d'un individu, et de l'envoyer de manière sans-fil à un téléphone intelligent Android, qui l'affiche en temps réel. Le système est composé de trois électrodes, d'un circuit d'amplification analogique, d'un microcontrôleur et d'un téléphone intelligent. Le téléphone intelligent Android et le microcontrôleur communique par le biais du protocole *Bluetooth Low Energy*. Le prototype obtenu mesure 7 cm x 5 cm x 3 cm et coute 46,45$. Il peut tenir entre 55 heures et 109 heures sur une batterie CR2032. Il permet d'étendre le domaine d'application des ECG a la surveillance en continue des personnes à risque cardiaque. Il permet aussi leur application dans le domaine du fitness, en créant des exercices personnalisés.
Le prototype obtenu est fonctionnel et respecte toutes les spécifications. Cependant, il possède certaines limites. En effet, il ne fournit qu'un seul canal. Ce nombre pourrait être augmente dans des itérations de prototype futures. Aussi, le système nécessite *Bluetooth Low Energy*, qui n'est pas encore très répandu pour les téléphones intelligents. Cependant, cela risque de changer dans les prochaines années. Finalement, le système est moins précis que les systèmes professionnels actuels. Plusieurs solutions existent afin d'améliorer la précision du système, au détriment du coût total du système.

Le système présente beaucoup de similitudes avec le produit qu'une compagnie développe actuellement. En effet, Carré Technologies développe actuellement Hexoskin, qui est un t-shirt incluant plusieurs capteurs notamment l'ECG et l'oxymétrie.

De nombreuses pistes existent pour pousser le projet plus loin. Tout d'abord, le développement d'applications pour *Android* et *iOS* serait intéressant. Des applications pour le domaine biomédical et pour le domaine du fitness. Il faudrait aussi programmer des algorithmes de reconnaissances pour identifier certaines caractéristiques dans un ECG.

Par la suite, on pourrait penser ajouter d'autres capteurs. La seule partie à changer est la partie des électrodes et d'amplification analogique. Tout le reste est réutilisable. Il sera donc facile de rajouter d'autres capteurs comme les EEG, l'oxymétrie ou encore la spectroscopie proche-infrarouge. Cela ouvrirait encore plus d'applications possibles pour le système.

Bibliographie

Auteur inconnu, «What is the ECG about». [En ligne]. Disponible: https://www.us.elsevierhealth.com/media/us/samplechapters/9780443068171/ECG%20Made%20Easy%201-40.pdf. [Accès le 18 Avril 2014].

Texas Instruments, «INA122 Datasheet». [En ligne]. Disponible: *http://www.ti.com/lit/ds/symlink/ina122.pdf*. [Accès le 18 Avril 2014].

Texas Instruments, «TLC277 Datasheet». [En ligne]. Disponible: *http://www.ti.com/lit/ds/symlink/tlc277.pdf*. [Accès le 18 Avril 2014].

Auteur inconnu, «Datasheet RFD22102 RFduino DIP». [En ligne]. Disponible: *http://www.rfduino.com/wp-content/uploads/2014/03/rfduino.datasheet.pdf*. [Accès le 18 Avril 2014].

Réjean Plamondon, «Définition des spécifications fonctionnelles». [En ligne]. Disponible: *https://moodle.polymtl.ca/mod/resource/view.php?id=164303*. [Accès le 18 Avril 2014].

Annexes

Annexe 1 : Code du microcontrôleur

```
#include <RFduinoBLE.h>

// Constants
const char DEVICE_NAME[] = "RFduino";
const char ADVERTISEMENT_DATA[] = "ECG1";
const int ADVERTISEMENT_INTERVAL = 250;
const int TX_POWER_LEVEL = 0;
const int BAUD_RATE = 9600;
const int SAMPLING_RATE = 300;
const int SETUP_TIME = 100;
const int ELECTRODE_PIN = 2;
const int SLEEP_TIME = 1;
const int SAMPLE_SIZE = 6; // 6 bytes
const int SAMPLES_PER_PACKET = 3;

// Actions
const uint8_t STOP_SAMPLING = 0x00;
const uint8_t START_SAMPLING = 0x01;

// Attributes
bool isSampling;
long startSamplingTime;
int sampleNumber;

void setup()
{
  // Attributes Initialization
  isSampling = false;
  sampleNumber = 0;
  analogReadResolution(12);

  // Serial Initialization
  Serial.begin(BAUD_RATE);

  // Bluetooth Initialization
  RFduinoBLE.deviceName = DEVICE_NAME;
  RFduinoBLE.advertisementData = ADVERTISEMENT_DATA;
  RFduinoBLE.advertisementInterval = MILLISECONDS(ADVERTISEMENT_INTERVAL);
  RFduinoBLE.txPowerLevel = TX_POWER_LEVEL;  // (-20dBm to +4 dBm)
  RFduinoBLE.begin();

  delay(SETUP_TIME); // Time for set up
}
```

```
void loop()
{
 if(isSampling)
 {
  // Sample value from electrodes
  unsigned long sampleTime = millis() - startSamplingTime;
  unsigned int sampleValue = analogRead(ELECTRODE_PIN);

  // Send Data
  char data[SAMPLE_SIZE*SAMPLES_PER_PACKET];
  data[sampleNumber*SAMPLE_SIZE] = (char) ((sampleTime >> 24) & 0xFF);
  data[sampleNumber*SAMPLE_SIZE+1] = (char) ((sampleTime >> 16) & 0xFF);
  data[sampleNumber*SAMPLE_SIZE+2] = (char) ((sampleTime >> 8) & 0xFF);
  data[sampleNumber*SAMPLE_SIZE+3] = (char) ((sampleTime) & 0xFF);
  data[sampleNumber*SAMPLE_SIZE+4] = (char) (sampleValue >> 8);
  data[sampleNumber*SAMPLE_SIZE+5] = (char) (sampleValue % 256);
  sampleNumber++;

  if(sampleNumber == SAMPLES_PER_PACKET)
  {
  while(RFduinoBLE.radioActive);
   {
     RFduinoBLE.send(data, SAMPLE_SIZE*SAMPLES_PER_PACKET);
   }
   sampleNumber = 0;
  }

  // Wait until next sample
  RFduino_ULPDelay(SECONDS(1.0/SAMPLING_RATE));
 }
 else
 {
  RFduino_ULPDelay(SECONDS(SLEEP_TIME)); // Wait for BLE interrupt.
 }
}

void RFduinoBLE_onAdvertisement(bool start)
{
 Serial.println("Advertising..."); // For debugging purpose.
}

void RFduinoBLE_onConnect()
{
 Serial.println("Connected to Radio..."); // For debugging purpose.
}

void RFduinoBLE_onDisconnect()
{
```

```
  Serial.println("Disconnected from Radio..."); // For debugging purpose.
  isSampling = false;
  sampleNumber = 0;
}

void RFduinoBLE_onReceive(char *data, int len)
{

  if(len > 0 )
  {
    uint8_t action = data[0];
    if(action == START_SAMPLING)
    {
      Serial.println("Turning sampling on"); // For debugging purpose.

      isSampling = true;
      startSamplingTime = millis();
      sampleNumber = 0;
    }
    else if(action == STOP_SAMPLING)
    {
      Serial.println("Turning sampling off"); // For debugging purpose.
      sampleNumber = 0;
      isSampling = false;
    }
  }
}
```

Annexe 2 : Code de l'application Android

AndroidManifest.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.symbiosis.eegmonitor">
    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="19" />

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

    <uses-feature android:name="android.hardware.bluetooth_le"
android:required="true" />

    <application android:theme="@android:style/Theme.Holo"
        android:icon="@drawable/eegmonitor" android:label="Biological Signal Monitor"
android:debuggable="true">
        <activity android:name="com.symbiosis.eegmonitor.MainActivity"
            android:screenOrientation="landscape"
                   android:configChanges="keyboardHidden|orientation|screenSize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
      <service android:name=".RFduinoService" />
    </application>
</manifest>
```

layout_main.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:orientation="horizontal"
            android:layout_marginLeft="0px"
            android:layout_marginRight="0px"
            android:layout_marginTop="0px"
            android:layout_weight="2"
            >

        <com.androidplot.xy.XYPlot
            android:id="@+id/dynamicPlot"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:layout_marginLeft="10px"
            android:layout_marginRight="10px"
            android:layout_marginTop="10px"
            android:layout_weight="1"
            />


             </LinearLayout>

        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:orientation="horizontal"
            android:layout_marginLeft="10px"
            android:layout_marginRight="10px"
            android:layout_marginTop="10px"
            android:layout_weight="6"
            >

            <Button
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="Enable Bluetooth"
            android:id="@+id/enableBluetooth"
            android:enabled="true"
            android:layout_weight="1" />

              <LinearLayout
                  android:layout_width="fill_parent"
                  android:layout_height="fill_parent"
                  android:orientation="vertical"
                  android:layout_marginLeft="0px"
                  android:layout_marginRight="0px"
```

```
        android:layout_marginTop="0px"
        android:layout_weight="1"
        >

        <Button
            android:id="@+id/scan"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="Scan"
         android:layout_weight="1"
               android:layout_gravity="center_horizontal"
               android:enabled="false" />
            />

        <Button
            android:id="@+id/connect"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="Connect"
         android:layout_weight="1"
         android:layout_gravity="center_horizontal"
         android:enabled="false"
            />

    </LinearLayout>

      <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical"
        android:layout_marginLeft="0px"
        android:layout_marginRight="0px"
        android:layout_marginTop="0px"
        android:layout_weight="1"
        >

        <Button
            android:id="@+id/start"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="Start"
         android:layout_weight="1"
            />

        <Button
            android:id="@+id/stop"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="Stop"
         android:layout_weight="1"
            />
    </LinearLayout>

      <LinearLayout
        android:layout_width="fill_parent"
```

```
                android:layout_height="fill_parent"
                android:orientation="vertical"
                android:layout_marginLeft="0px"
                android:layout_marginRight="0px"
                android:layout_marginTop="0px"
                android:layout_weight="1"
                >

                        <TextView
                            android:layout_width="fill_parent"
                            android:layout_height="fill_parent"
                            android:id="@+id/scanStatus"
                            android:layout_weight="1" />

                        <TextView
                            android:layout_width="fill_parent"
                            android:layout_height="fill_parent"
                            android:id="@+id/deviceInfo"
                            android:layout_weight="1" />

                        <TextView
                            android:layout_width="fill_parent"
                            android:layout_height="fill_parent"
                            android:id="@+id/connectionStatus"
                            android:layout_weight="1" />

            </LinearLayout>
        </LinearLayout>
</LinearLayout>
```

MainActivity.java :

```
package com.symbiosis.eegmonitor;

import java.util.Arrays;
import java.util.UUID;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.graphics.Color;
import android.graphics.LinearGradient;
import android.graphics.Paint;
import android.graphics.Shader.TileMode;
import android.os.Bundle;
import android.os.IBinder;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import com.androidplot.Plot;
import com.androidplot.xy.BoundaryMode;
import com.androidplot.xy.LineAndPointFormatter;
import com.androidplot.xy.PointLabelFormatter;
import com.androidplot.xy.SimpleXYSeries;
import com.androidplot.xy.XYPlot;
import com.androidplot.xy.XYStepMode;

public class MainActivity extends Activity implements BluetoothAdapter.LeScanCallback {


   private XYPlot dynamicPlot;
   private SimpleXYSeries eeg1;
   final private static int HISTORY_SIZE = 512;
   private double mostRecentTime;

   // State machine
   final private static int STATE_BLUETOOTH_OFF = 1;
   final private static int STATE_DISCONNECTED = 2;
   final private static int STATE_CONNECTING = 3;
   final private static int STATE_CONNECTED = 4;

         private static final int SAMPLE_PER_PACKET = 3;
```

```
        private static final int SAMPLE_SIZE = 6;
        private static final float SAMPLE_CORRECT = 2;
        private static final float TOTAL_GAIN = (float) (SAMPLE_CORRECT*14.1*71.2*4096/3000);
        private int state;

private boolean scanStarted;
private boolean scanning;

private BluetoothAdapter bluetoothAdapter;
private BluetoothDevice bluetoothDevice;

private RFduinoService rfduinoService;

private Button enableBluetoothButton;
private TextView scanStatusText;
private Button scanButton;
private TextView deviceInfoText;
private TextView connectionStatusText;
private Button connectButton;
private Button startSamplingButton;
private Button stopSamplingButton;

private final BroadcastReceiver bluetoothStateReceiver = new BroadcastReceiver() {
   @Override
   public void onReceive(Context context, Intent intent) {
      int state = intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, 0);
      if (state == BluetoothAdapter.STATE_ON) {
         upgradeState(STATE_DISCONNECTED);
      } else if (state == BluetoothAdapter.STATE_OFF) {
         downgradeState(STATE_BLUETOOTH_OFF);
      }
   }
};

private final BroadcastReceiver scanModeReceiver = new BroadcastReceiver() {
   @Override
   public void onReceive(Context context, Intent intent) {
      scanning = (bluetoothAdapter.getScanMode() != BluetoothAdapter.SCAN_MODE_NONE);
      scanStarted &= scanning;
      updateUi();
   }
};

private final ServiceConnection rfduinoServiceConnection = new ServiceConnection() {
   @Override
   public void onServiceConnected(ComponentName name, IBinder service) {
      rfduinoService = ((RFduinoService.LocalBinder) service).getService();
      if (rfduinoService.initialize()) {
```

```
            if (rfduinoService.connect(bluetoothDevice.getAddress())) {
                upgradeState(STATE_CONNECTING);
            }
        }
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        rfduinoService = null;
        downgradeState(STATE_DISCONNECTED);
    }
};

private final BroadcastReceiver rfduinoReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        if (RFduinoService.ACTION_CONNECTED.equals(action)) {
            upgradeState(STATE_CONNECTED);
        } else if (RFduinoService.ACTION_DISCONNECTED.equals(action)) {
            downgradeState(STATE_DISCONNECTED);
        } else if (RFduinoService.ACTION_DATA_AVAILABLE.equals(action)) {
            addData(intent.getByteArrayExtra(RFduinoService.EXTRA_DATA));
        }
    }
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(com.symbiosis.eegmonitor.R.layout.layout_main);

    mostRecentTime = 0;

    // Plot
    LineAndPointFormatter formatter = new LineAndPointFormatter(Color.BLACK, null, Color.rgb(0,
145, 255), (PointLabelFormatter) null);
    Paint lineFill = new Paint();
    lineFill.setAlpha(175);
    lineFill.setShader(new LinearGradient(0, 0, 0, 624, Color.rgb(0, 145, 255), Color.WHITE,
TileMode.CLAMP));

    formatter.setFillPaint(lineFill);

    dynamicPlot = (XYPlot) findViewById(com.symbiosis.eegmonitor.R.id.dynamicPlot);
    eeg1 = new SimpleXYSeries("ECG");
    dynamicPlot.setRangeBoundaries(-2048/TOTAL_GAIN, 2048/TOTAL_GAIN, BoundaryMode.FIXED);
    dynamicPlot.setDomainBoundaries(0, 1, BoundaryMode.AUTO);
```

```
dynamicPlot.addSeries(eeg1, formatter);
dynamicPlot.setDomainStep(XYStepMode.INCREMENT_BY_VAL, 0.5);
dynamicPlot.setRangeStep(XYStepMode.SUBDIVIDE, 5);

dynamicPlot.getGraphWidget().getBackgroundPaint().setColor(Color.TRANSPARENT);
dynamicPlot.getGraphWidget().getGridBackgroundPaint().setColor(Color.TRANSPARENT);
dynamicPlot.getGraphWidget().getRangeSubGridLinePaint().setColor(Color.WHITE);
dynamicPlot.getGraphWidget().getDomainSubGridLinePaint().setColor(Color.WHITE);
dynamicPlot.getGraphWidget().getDomainGridLinePaint().setColor(Color.WHITE);
dynamicPlot.getGraphWidget().getRangeGridLinePaint().setColor(Color.WHITE);
dynamicPlot.getGraphWidget().getDomainOriginLinePaint().setColor(Color.WHITE);
dynamicPlot.getGraphWidget().getRangeOriginLinePaint().setColor(Color.WHITE);
dynamicPlot.getGraphWidget().getCursorLabelBackgroundPaint().setColor(Color.WHITE);
dynamicPlot.getGraphWidget().getCursorLabelPaint().setColor(Color.WHITE);
dynamicPlot.getGraphWidget().getRangeLabelPaint().setColor(Color.WHITE);
dynamicPlot.getGraphWidget().getDomainLabelPaint().setColor(Color.WHITE);
dynamicPlot.getGraphWidget().getDomainOriginLabelPaint().setColor(Color.WHITE);
dynamicPlot.getGraphWidget().getRangeOriginLabelPaint().setColor(Color.WHITE);
dynamicPlot.setBorderStyle(Plot.BorderStyle.NONE, null, null);
dynamicPlot.getLegendWidget().getTextPaint().setColor(Color.WHITE);

dynamicPlot.setDomainLabel("Time (s)");
dynamicPlot.getDomainLabelWidget().pack();
dynamicPlot.setRangeLabel("Voltage (mV)");
dynamicPlot.getRangeLabelWidget().pack();


bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

// Bluetooth
enableBluetoothButton = (Button) findViewById(com.symbiosis.eegmonitor.R.id.enableBluetooth);
enableBluetoothButton.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      enableBluetoothButton.setEnabled(false);
      enableBluetoothButton.setText(
            bluetoothAdapter.enable() ? "Enabling bluetooth..." : "Enable failed!");
   }
});

// Find Device
scanStatusText = (TextView) findViewById(com.symbiosis.eegmonitor.R.id.scanStatus);

scanButton = (Button) findViewById(com.symbiosis.eegmonitor.R.id.scan);
scanButton.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      scanStarted = true;
```

```
            bluetoothAdapter.startLeScan(
                    new UUID[]{ RFduinoService.UUID_SERVICE },
                    MainActivity.this);
        }
    });

    // Device Info
    deviceInfoText = (TextView) findViewById(com.symbiosis.eegmonitor.R.id.deviceInfo);

    // Connect Device
    connectionStatusText = (TextView) findViewById(com.symbiosis.eegmonitor.R.id.connectionStatus);

    connectButton = (Button) findViewById(com.symbiosis.eegmonitor.R.id.connect);
    connectButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            v.setEnabled(false);
            connectionStatusText.setText("Connecting...");
            Intent rfduinoIntent = new Intent(com.symbiosis.eegmonitor.MainActivity.this,
com.symbiosis.eegmonitor.RFduinoService.class);
            bindService(rfduinoIntent, rfduinoServiceConnection, BIND_AUTO_CREATE);
        }
    });

    startSamplingButton = (Button) findViewById(com.symbiosis.eegmonitor.R.id.start);
    startSamplingButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            while(eeg1.size() != 0)
            {
                    eeg1.removeFirst();
            }
            dynamicPlot.redraw();
            mostRecentTime = 0;
            rfduinoService.send(new byte[]{1});
        }
    });

    stopSamplingButton = (Button) findViewById(com.symbiosis.eegmonitor.R.id.stop);
    stopSamplingButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            rfduinoService.send(new byte[]{0});
        }
    });
  }

  @Override
```

```
    protected void onStart() {
        super.onStart();

        registerReceiver(scanModeReceiver, new
IntentFilter(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED));
        registerReceiver(bluetoothStateReceiver, new
IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED));
        registerReceiver(rfduinoReceiver, RFduinoService.getIntentFilter());

        updateState(bluetoothAdapter.isEnabled() ? STATE_DISCONNECTED : STATE_BLUETOOTH_OFF);
    }

    @Override
    protected void onStop() {
        super.onStop();

        bluetoothAdapter.stopLeScan(this);

        unregisterReceiver(scanModeReceiver);
        unregisterReceiver(bluetoothStateReceiver);
        unregisterReceiver(rfduinoReceiver);
    }

    private void upgradeState(int newState) {
        if (newState > state) {
            updateState(newState);
        }
    }

    private void downgradeState(int newState) {
        if (newState < state) {
            updateState(newState);
        }
    }

    private void updateState(int newState) {
        state = newState;
        updateUi();
    }

    private void updateUi() {
        // Enable Bluetooth
        boolean on = state > STATE_BLUETOOTH_OFF;
        enableBluetoothButton.setEnabled(!on);
        enableBluetoothButton.setText(on ? "Bluetooth enabled" : "Enable Bluetooth");
        scanButton.setEnabled(on);

        // Scan
```

```
        if (scanStarted && scanning) {
            scanStatusText.setText("Scanning...");
            scanButton.setText("Stop Scan");
            scanButton.setEnabled(true);
        } else if (scanStarted) {
            scanStatusText.setText("Scan started...");
            scanButton.setEnabled(false);
        } else {
            scanStatusText.setText("");
            scanButton.setText("Scan");
            scanButton.setEnabled(true);
        }

        // Connect
        boolean connected = false;
        String connectionText = "Disconnected";
        if (state == STATE_CONNECTING) {
            connectionText = "Connecting...";
        } else if (state == STATE_CONNECTED) {
            connected = true;
            connectionText = "Connected";
        }
        connectionStatusText.setText(connectionText);
        connectButton.setEnabled(bluetoothDevice != null && state == STATE_DISCONNECTED);

        // Send
        startSamplingButton.setEnabled(connected);
                        stopSamplingButton.setEnabled(connected);
    }

    private void addData(byte[] data) {

            for(int i = 0; i < SAMPLE_PER_PACKET*SAMPLE_SIZE; i += SAMPLE_SIZE)
            {
                        // Retrieve data
                        long sampleTime = HexAsciiHelper.byteArrayToInt(Arrays.copyOfRange(data, i, i+4));

                        int sampleValue = HexAsciiHelper.byteArrayToSample(Arrays.copyOfRange(data, i+4,
i+6));
                        double realSampleTime = sampleTime;
                        float realSampleValue = (float) SAMPLE_CORRECT*(sampleValue - 2048)/TOTAL_GAIN;
                        if(realSampleValue > 2047/TOTAL_GAIN) {realSampleValue = 2047/TOTAL_GAIN;} else
if(realSampleValue < -2047/TOTAL_GAIN) {realSampleValue = -2047/TOTAL_GAIN;}
                        System.out.println("Number of bytes " + data.length);
                        System.out.println("Data Received: " + HexAsciiHelper.bytesToHex(data));
                        System.out.println("Sample Time and Value: " + realSampleTime + ", " +
realSampleValue);
```

```
                    // Add to plotData
                    if(realSampleTime > mostRecentTime ) // Packet is valid
                    {
                            mostRecentTime = realSampleTime;
                        // get rid the oldest sample in history:
                        if (eeg1.size() > HISTORY_SIZE) {
                           eeg1.removeFirst();
                        }
                        // add the latest history sample:
                        eeg1.addLast(realSampleTime/1000.0, realSampleValue);
                    }

            }

        // UpdateUI
            dynamicPlot.redraw();

    }

    @Override
    public void onLeScan(BluetoothDevice device, final int rssi, final byte[] scanRecord) {
        bluetoothAdapter.stopLeScan(this);
        bluetoothDevice = device;

        MainActivity.this.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                deviceInfoText.setText(
                        BluetoothHelper.getDeviceInfoText(bluetoothDevice, rssi, scanRecord));
                updateUi();
            }
        });
    }

}
```

// D'autres fichiers sont nécessaires, notamment *RFduinoService.java*, *BluetoothHelper.java*, *EditData.java* et *HexAsciiHelper.java*.