ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⴰⵎⴻⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ
ⵜⴰⵖⴻⵍⵍⴰⵜ ⵜⴰⵎⴰⵡⴰⵙⵜ ⴻⴷⵀ ⵣⴻⵀⵔⴰ
جامعة سيدي محمد بن عبدالله
كلية العلوم ظهر المهراز
Université Sidi Mohamed Ben Abdellah
Faculté des Sciences Dhar Mahraz
MASTER WISD

Traitement d'un signal ECG en temps réel avec la carte Arduino et Android

Année universitaire: 2019/2020

SOMMAIRE

TABLE DE FIGURES

I. INTRODUCTION

Les maladies cardiaques deviennent l'un des plus grands problèmes depuis nombreuses années. Selon l'Organisation Mondiale de la Santé, 17.5 millions de décès dus aux maladies cardiaques ont été enregistrés en 2012. Par conséquence, des équipements de soins et de sante et les systèmes de surveillance sont conçus pour suivre cette maladie, en analysant le signal ECG pour que cette maladie soit peut-être prévenue.

Afin de remédier à ce problème, nous présentons donc ce projet, basé sur la carte Arduino. Le projet consistera en un capteur principal appelé '*XD-58C*', qui va nous permettre de lire le signal ECG et une application Android pour afficher les résultats de l'analyse.

1. L'électrocardiographie

L'électrocardiographie est un examen rapide qui ne prend que quelques minutes et qui vise à mesurer et à enregistrer l'activité électrique du cœur d'un patient. Il est indolore et non invasif, dénué de tout danger et Il peut être fait en cabinet de médecin, à l'hôpital, voire à domicile. Son interprétation reste cependant complexe et requiert une analyse méthodique et une certaine expérience du clinicien. Il permet de mettre en évidence diverses anomalies cardiaques et a une place importante dans les examens diagnostiques en cardiologie, comme pour la maladie coronarienne.

Ce type d'examen vous sera peut-être recommandé si vous êtes atteint d'arythmie ou que vous souffrez de douleurs à la poitrine ou de palpitations. Des résultats anormaux d'ECG permettent de détecter différents problèmes cardiaques.

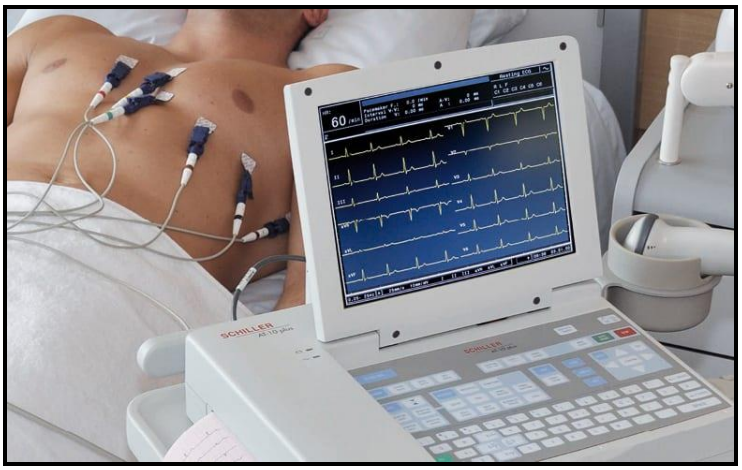


FIGURE 1 : TEST ELECTROCARDIOGRAPHE

2. L'électrocardiogramme (ECG)

L'électrocardiogramme (ECG) c'est un signal de tension que le cœur génère lorsqu'il bat. Chaque battement de cœur est une séquence de contractions et de relaxations, qui nécessitent un signal électrique traversant les muscles du cœur dans un ordre particulier. Ces signaux se propagent sur la peau où, avec des bons capteurs, peuvent être mesurés. Ils sont calculés avec la différence de tension entre deux points du corps.

Un ECG étudie le fonctionnement du cœur en mesurant son activité électrique. À chaque battement cardiaque, une impulsion électrique (ou « onde ») traverse le cœur. Cette onde fait contracter le muscle cardiaque afin qu'il expulse le sang du cœur. L'ECG mesure et enregistre l'activité électrique qui traverse le cœur.

II. BASES DE L'INTERPRETATION D'UN ECG

1. Lecture et interprétation d'un ECG

La lecture et l'interprétation d'un ECG requièrent une grande habitude qui ne peut être acquise que par une pratique régulière.

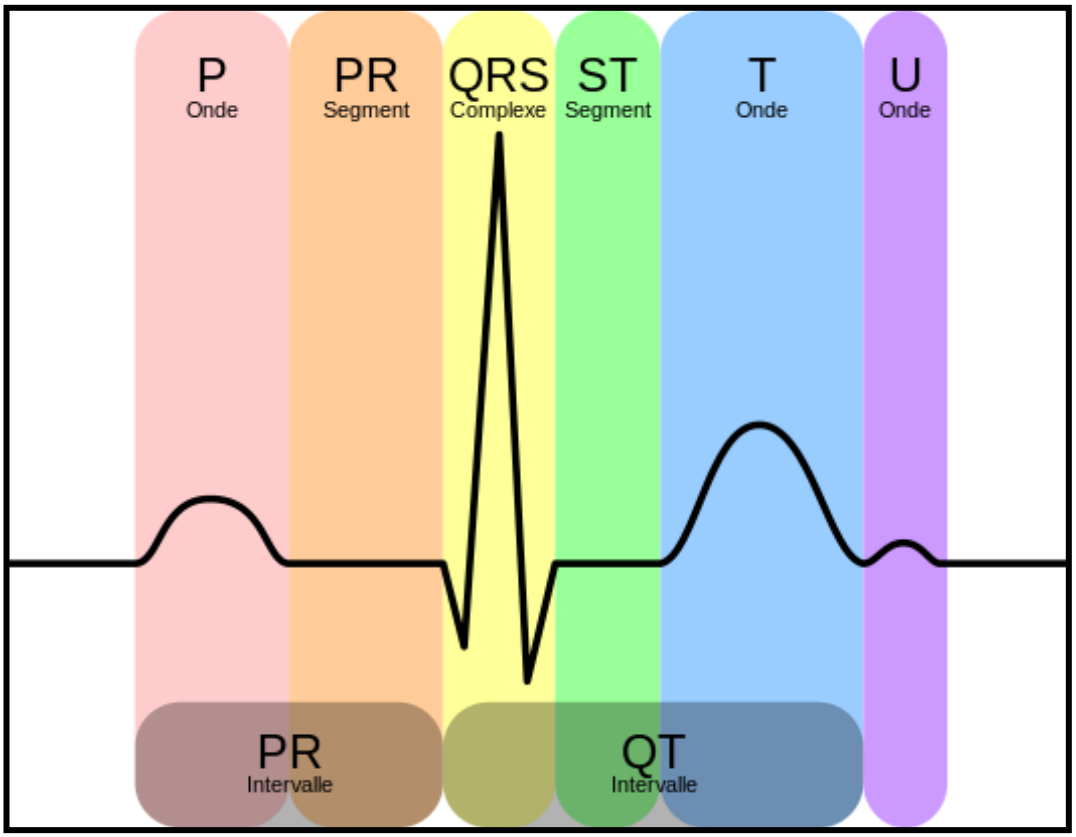


FIGURE 2 : REPRESENTATION D'UN ECG NORMAL

Le signal ECG va permettre d'enregistrer l'activité électrique liée à la dépolarisation des oreillettes droite et gauche, c'est la première onde appelée onde **P**, puis l'activité électrique liée à la dépolarisation des ventricules c'est le complexe **QRS**, enfin l'activité électrique liée à la repolarisation des ventricules c'est l'onde **T**. Il y a des valeurs normales qui permettent d'interpréter les résultats. Par exemple, si l'onde **P** est plus haute et plus large, cela traduit une anomalie au niveau de l'oreillette qui est dilatée. De la même manière, lorsque le complexe **QRS** est trop grand c'est un signe d'épaississement de la paroi du ventricule (hypertrophie). On analyse l'ensemble de ces complexes et la morphologie globale du complexe va nous orienter vers telle ou telle pathologie. Lorsque l'onde **P** normale disparaît, c'est qu'il y a un trouble du rythme ou de la conduction électrique.

Après les contrôles cités précédemment sur l'interopérabilité du tracé, l'analyse de l'ECG se poursuit par l'étude du rythme et de la fréquence cardiaque (nombre de QRS par unité de temps).

L’analyse doit être méthodique et systématique. Elle comporte l’étude de :

- La fréquence et le rythme
- L’onde **P** : durée et amplitude
- L’intervalle **PR** : durée
- Le complexe **QRS** : morphologie, durée, axe
- L’intervalle **ST** : morphologie, durée
- L’onde **T** : morphologie, amplitude, durée
- L’espace **QT** : durée et morphologie· l’onde **U**

Le tracé électrique comporte plusieurs accidents répétitifs appelés ‘*ondes’*, et différents intervalles entre ces ondes. Les principales mesures à effectuer lors de l'analyse d'un ECG sont celles de l'onde P, de l'espace PR, du complexe QRS, du délai d'inscription de la déflexion, du point J, de l'espace QT, du segment ST et enfin de l'onde T.

- **Onde P** : correspond à la dépolarisation (et la contraction) des oreillettes, droite et gauche. On analyse sa morphologie (positive, diphasique ou monophasique), sa durée (qui est de 0,08 à 0,1 seconde), son amplitude (inférieure à 2,5 mm en des déviations et 2 mm en autres), son axe (déterminé de la même façon que pour l'axe des QRS, normalement situé entre 0 et 90°, généralement vers 60°) et sa synchronisation avec l'onde QRS.
- **Intervalle PR** : (ou PQ) C’est le temps entre le début de P et le début du QRS. Il est le témoin du temps nécessaire à la transmission de l’activité électrique du nœud sinusal des oreillettes au tissu myocardique des ventricules (conduction auriculo-ventriculaire). Sa durée normale, mesurée du début de l'onde P au début du complexe QRS est de 0,12 à 0,20 seconde. La durée de l'espace PR diminue lorsque la fréquence cardiaque augmente. Il est normalement isoélectrique.
- **Onde QRS** : (appelée aussi complexe QRS) qui correspond à la dépolarisation (et la contraction) des ventricules, droit et gauche. L'onde Q est la première onde négative du complexe. L'onde R est la première composante positive du complexe. L'onde S est la deuxième composante négative. Suivant la dérivation et sa forme, on parle ainsi d'aspect « QS », « RS ». La forme et l'amplitude du QRS varient selon les dérivations et selon l'éventuelle pathologie du muscle cardiaque sous-jacent. Le complexe QRS a une durée normale inférieure à 0,1 seconde, le plus souvent inférieur à 0,08 s. L'axe des QRS normaux est compris entre 0 et 90°. La zone de transition correspondant à la dérivation précordiale dans laquelle les QRS sont isoélectriques.
- **Point J** : correspond au point de transition entre le complexe QRS et le segment ST. Il est normalement isoélectrique.
- **Segment ST** : correspond au temps séparant la fin de la dépolarisation ventriculaire représentée par le complexe QRS et le début de l'onde T. Le segment ST normal est isoélectrique du point J au début de l'onde T.
- **Intervalle QT** : mesuré du début du QRS à la fin de l'onde T correspond à l'ensemble de la dépolarisation et de la repolarisation ventriculaire. Sa durée varie en fonction de la fréquence cardiaque, il diminue quand la fréquence cardiaque augmente et augmente quand la fréquence cardiaque diminue. Son allongement est lié dans

certaines circonstances à l'apparition d'un trouble du rythme ventriculaire complexe nommé « torsades de pointes » potentiellement mortel. L'hypoxie cardiaque et les troubles de la concentration sanguine en calcium affectent cet intervalle.

- **Onde T** : correspond à la repolarisation (la relaxation) des ventricules. Sa durée est de 0,20 à 0,25 secondes, l'analyse de sa durée est comprise dans l'analyse de la durée de l'intervalle QT. L'axe normal de l'onde T, calculé de la même façon que l'axe des QRS, est compris entre - 10 et 70°, souvent autour de 40°. L'onde T est normalement pointue, asymétrique et ample dans la plupart des dérivations. Son amplitude dépend généralement de celle de l'onde R qui la précède, elle est comprise entre 1/8 et 2/3 de celle de l'onde R et ne dépasse pas le plus souvent 10 mm.
- **Onde T atriale** : est masquée par l'onde QRS et correspond à la repolarisation (la relaxation) des oreillettes. Mais celle-ci est négative.
- **Onde U** : est une petite déflexion parfois observée après l'onde T dans les dérivations précordiales. Elle est positive dans toutes les dérivations sauf en 'aVR'.

2. Caractéristiques d'un ECG normal

Pour qu'un ECG soit normal, plusieurs points doivent être réunis :

- **Le rythme et la fréquence** : Le rythme cardiaque normal (provenant du nœud sinusal situé dans l'oreillette droite) est un rythme régulier avec une fréquence cardiaque comprise entre 50 et 100 battements par minute (BPM). En dessous de 50 BPM, on parle de *bradycardie* et au-dessus de 100 BPM, on parle de *tachycardie*.
- **Onde P** : Il faut que sa durée est inférieure à 0,10 seconds. Et son amplitude inférieure ou égale à 2,5 mm maximale en DII et V1. L'onde P sinusale est toujours négative en aVR et positive en DI et DII.
- **Espace PR** : Entre le début de l'onde P et le début du complexe QRS. Il est isoélectrique et sa durée est comprise entre 0,12 et 0,20 s. Il diminue si la fréquence cardiaque augmente et augmente avec l'âge.
- **Complexes QRS** : Il faut que sa durée soit inférieure à 0,11 s et son délai d'apparition de la déflexion intrinsécoïde est inférieur à 0,04 s en V1 et 0,06 s en V6, son axe entre 0 et 90°
- **Repolarisation** : Point J et segment ST isoélectriques ; ondes T positives, asymétriques, d'axe proche de celui des QRS. Ondes U absentes ou inférieures aux ondes T. Intervalle QT prévu par la fréquence cardiaque.

Le tableau ci-dessous résume les valeurs normales pour un signal ECG :

Onde, intervalle	Valeurs normales
Fréquence cardiaque (Fréquence Cardiaque)	60–100 bpm
Durée de P	< 120 ms
Axe de P	60° (D2)
Amplitude de P	< 2,5 mm (en D2)
PR	120–200 ms
Durée de QRS	80–100 ms, en pratique 80–120 ms
Axe de QRS	– 45 à + 110°, en pratique – 30 à + 90°
Onde Q physiologique	< 1/3 amplitude QRS et < 40 ms de durée
QT (variable avec FC)	< 440 ms à 60 bpm

Tableau 1 : Résumé des valeurs normales.

3. La fréquence cardiaque (BPM)

La fréquence cardiaque est le nombre de battements cardiaques (ou pulsations) par unité de temps (généralement la minute), en anglais BPM (beats per minute). C'est une notion quantitative qui peut aussi se définir en nombre de cycles par seconde.

Quand on parle de fréquence cardiaque on parle en fait de la fréquence ventriculaire. Elle se mesure donc en comptant le nombre de complexes QRS par minute. Soit on peut s'aider d'une règle graduée, soit de manière plus approximative en divisant 300 par le nombre de grands carrés séparant 2 complexes QRS.

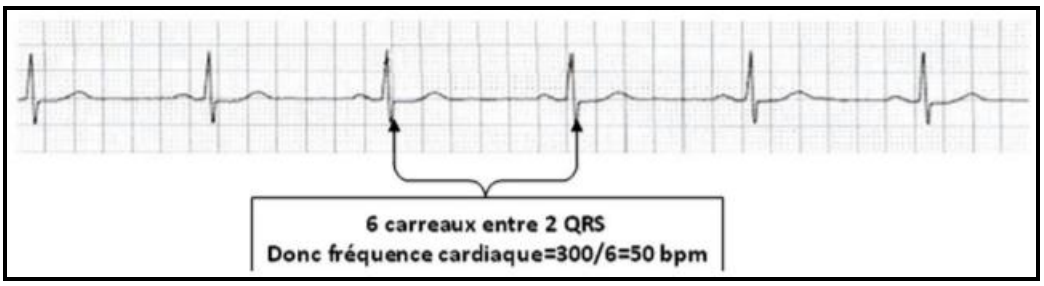


FIGURE 3 : CALCUL SIMPLIFIE DE LA FREQUENCE CARDIAQUE

La fréquence cardiaque au repos chez l'Homme est de 60 à 80 battements par minute, pour un débit de 4,5 à 5 litres de sang par minute.

Chez l'adulte en bonne santé, au repos, la fréquence cardiaque se situe entre 50 (sportif pratiquant l'endurance) et 80 pulsations par minute. Pendant un effort, la fréquence cardiaque maximale théorique est de 220 moins l'âge (exemple : 220 - 40 ans = 180).

La fréquence cardiaque au repos varie selon l'âge :

Age	Fréquence cardiaque au repos
à la naissance	90 à 180 bpm
à 1 an	80 à 160 bpm
à 4 ans	80 à 120 bpm
à l'adolescence	70 à 110 bpm

Tableau 2 : La fréquence cardiaque au repos en fonction de l'âge.

Les facteurs de stress influent sur la fréquence cardiaque grâce à trois mécanismes : les mécanismes nerveux, chimique et physique. On dispose d'un système nerveux autonome qui se distingue par un système nerveux sympathique et parasympathique. Le premier système a pour fonction d'augmenter la fréquence cardiaque ; le deuxième la diminue.

4. L'intervalle entre les battements (IBI)

L'intervalle entre les battements "*InterBeat Interval*" (IBI) est un terme scientifique utilisé en référence à l'intervalle de temps entre les battements individuels du cœur. L'intervalle de battement est abrégé "IBI" qui signifie "intervalle entre battements". Il est aussi parfois appelé intervalle "beat-to-beat".

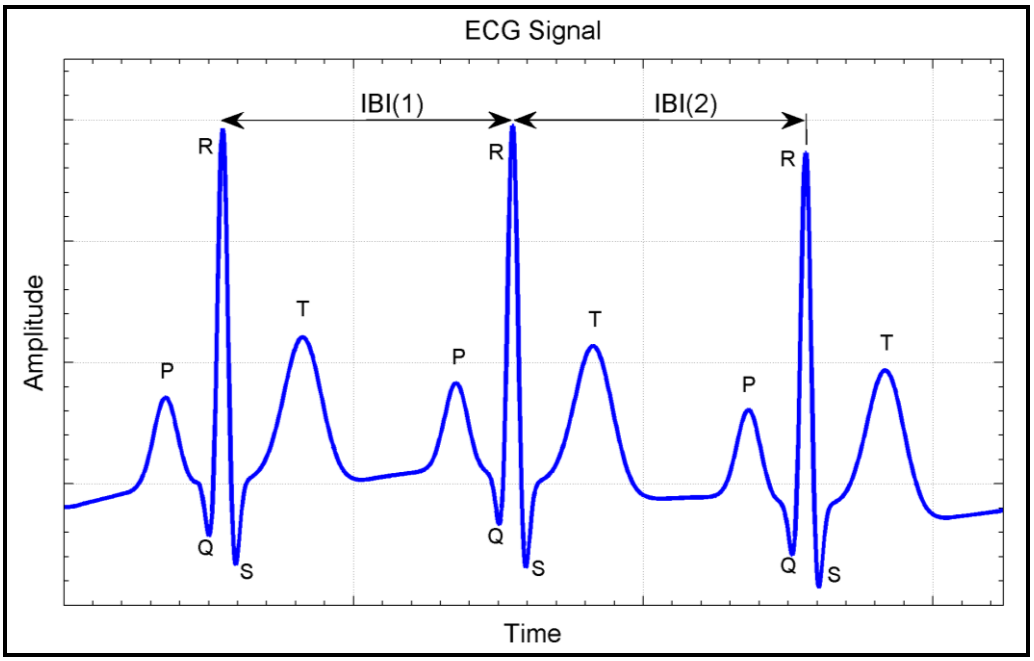


FIGURE 4 : REPRESENTATION D'UN IBI

L'IBI est généralement mesuré en unités de millisecondes. Dans la fréquence cardiaque normale, chaque valeur IBI varie d'un battement à l'autre. Cette variation naturelle est connue sous le nom de variabilité de la fréquence cardiaque (VRC). Cependant, certaines conditions cardiaques peuvent faire en sorte que les valeurs individuelles de l'IBI deviennent presque constantes, ce qui fait que le VRC est presque nul. Cela peut se produire, par exemple, pendant les périodes d'exercice lorsque la fréquence cardiaque augmente et que les battements deviennent réguliers. Certaines maladies peuvent entraîner une augmentation et une uniformisation de la fréquence cardiaque, par exemple lorsqu'un sujet est atteint d'une infection.

III. PARTIE MATERIEL ET FONCTIONNEMENT

1. Matériel et Composants

- **La carte Arduino UNO :**

Une carte Arduino (5,33 x 6,85 cm) est une petite carte électronique équipée d'un microcontrôleur. Le microcontrôleur permet, à partir d'événements détectés par des capteurs, de programmer et commander des actionneurs ; la carte Arduino est donc une interface programmable.

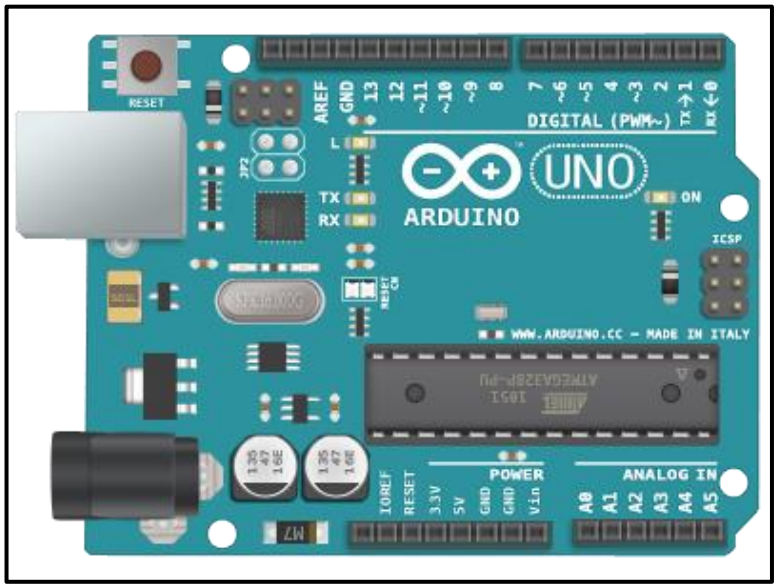


FIGURE 5 : LA CARTE ARDUINO

- **Alimentation / Programmation par USB**

La carte Arduino peut être alimentée avec un câble USB relié à votre ordinateur. Tout ce dont vous avez besoin, c'est de connecter votre carte Arduino à votre ordinateur avec le câble USB type A/B.

- **Microcontrôleur :** stocke le programme et l'exécute.

- **Les Broches (3.3, 5, GND, Vin) de la carte Arduino**

3.3V (6) – Broche d'alimentation de tension 3.3 Volts

5V (7) – Broche d'alimentation de tension 5 Volts

La plupart des composants destinés à fonctionner avec Arduino fonctionnent bien en 3.3 Volts ou 5 Volts.

GND (8) (Ground / Masse) – Il y a plusieurs broches de ce type présentes sur la carte Arduino, elles sont toutes communes et peuvent être utilisées comme masse (potentiel 0 Volts) pour vos circuits.

Vin (9) – Cette broche permet d'alimenter l'Arduino depuis une source de tension extérieure. Elle est reliée au circuit d'alimentation principale de la carte Arduino.

- **Broches analogiques de la carte Arduino**

L'Arduino UNO possède **5 broches d'entrées analogiques numérotée de A0 jusqu'à A5**. Ces broches permettent de lire un signal analogique d'un capteur comme un capteur d'humidité ou de température.

La carte Arduino utilise **un convertisseur analogique/numérique** (convertisseur CAN) pour permettre la lecture du signal par le microcontrôleur. Un signal sera converti sur **10 bits**. **La valeur pourra être lue sur une échelle 1024 points.**

- **Entrées/Sorties numériques de la carte Arduino**

La carte Arduino UNO possède **14 broches d'Entrées / Sorties numériques** (15), dont 6 peuvent fournir une sortie PWM (Pulse Width Modulation). Ces broches peuvent être configurées pour fonctionner comme des broches numériques d'entrée pour lire des valeurs logiques (0 ou 1) ou numériques. Elles peuvent également être utilisées comme des broches de sortie pour piloter différents modules comme des LEDs, des relais, etc. Les broches étiquetées "~" peuvent être utilisées pour générer des PWM.

- **<u>Module Bluetooth HC-05 :</u>**

Le module Bluetooth HC-05 est un module MAÎTRE / ESCLAVE. Par défaut, le réglage d'usine est ESCLAVE. Le rôle du module (maître ou esclave) peut être configuré uniquement par AT COMMANDS. Les modules esclaves ne peuvent pas établir de connexion avec un autre appareil Bluetooth, mais peut accepter des connexions. Le module maître peut initier une connexion à d'autres appareils. L'utilisateur peut l'utiliser simplement pour un remplacement de port série pour établir une connexion entre le MCU et le GPS, le PC à votre projet intégré, etc. En bref Ce module Bluetooth HC05 permet d'ajouter une liaison Bluetooth sur vos projets à microcontrôleur (communication via série TTL).

<u>FIGURE 6 : MODULE HC-05</u>

Le rôle des différentes broches de HC-05 :

- **Broche STATE :**

  La broche d'état est connectée à la LED intégrée, elle peut être utilisée comme rétroaction pour vérifier si Bluetooth fonctionne correctement.

- **Broche RX :**

  Recevoir des données série. Toutes les données série données à cette broche seront diffusées via Bluetooth.

- **Broche TX :**

  Transmet des données série. Tout ce qui est reçu via Bluetooth sera distribué par cette broche sous forme de données série.

- **Broche GND :**

  Broche de mise à la terre du module, connectez à la terre du système.

- **Broche 5V :**

  Alimente le module, Connectez à une tension d'alimentation de + 5V.

- **Broches-EN :**

  Cette broche est utilisée pour basculer entre le mode de données (réglé bas) et le mode de commande AT (réglé haut). Par défaut, il est en mode Données.

- **<u>Capteur de Rythme Cardiaque XD-58C :</u>**

Le module capteur d'impulsions XD-58C est utilisé pour mesurer la fréquence cardiaque.

Il est largement utilisé par les étudiants, les artistes, les athlètes, les inventeurs, les développeurs de jeux ou de terminaux mobiles pour développer des œuvres interactives liées à la fréquence cardiaque.

Vous pouvez porter le capteur sur votre doigt ou votre lobe d'oreille et le connecter à Arduino. Il dispose également d'un programme APP open source qui peut afficher votre fréquence cardiaque en temps réel dans un graphique. C'est un capteur de fréquence cardiaque intégré à un amplificateur optique et un circuit d'élimination du bruit.

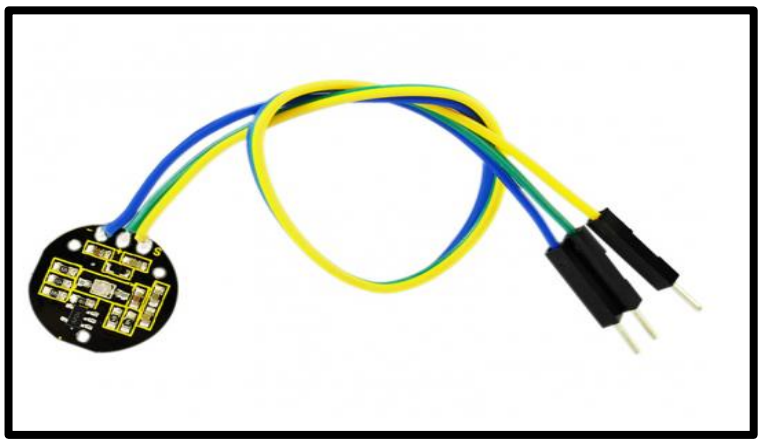


FIGURE 7 : PULSE SENSOR XD-58C

- **Indication par couleurs des différentes broches :**

(s) -> Sortie du signal : de type analogique.

(+) -> Alimentation: 3.3 V

(-) -> GND : Ground 0V

- **Zoom sur le capteur :**

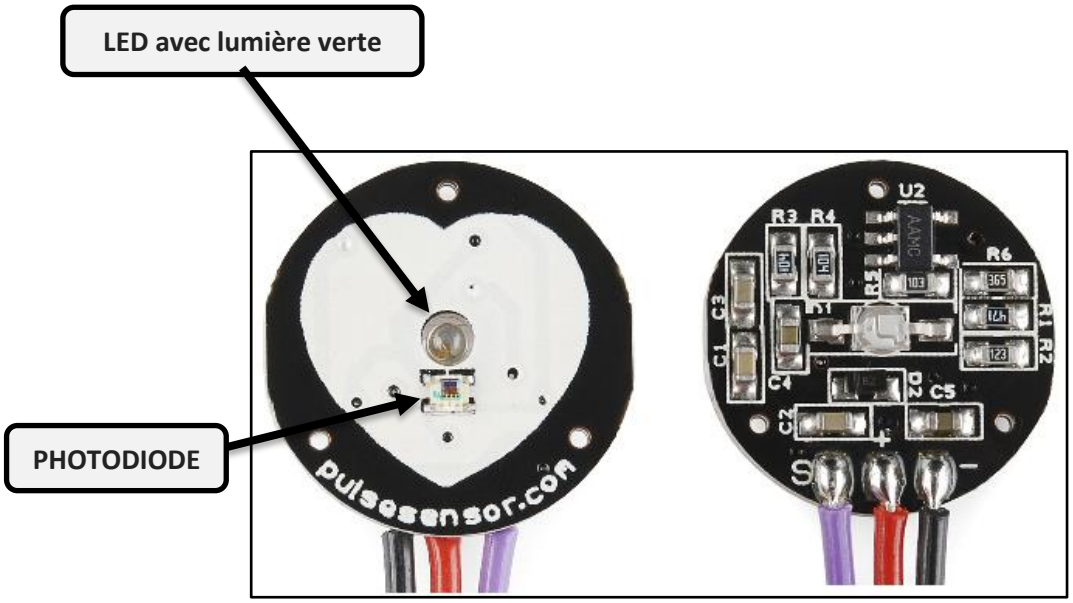


FIGURE 8 : ZOOM SUR LE CAPTEUR

Le capteur se compose d'une LED verte super brillante et d'un détecteur de lumière (PHOTODIODE).

2. Fonctionnement du capteur

Le capteur se compose d'une LED verte super brillante et détecteur de lumière. La LED doit être super brillant car la lumière maximale doit passer étalée dans le doigt et détectée par le détecteur. Maintenant, quand le cœur pompe une impulsion dans le vaisseau, le doigt devient légèrement plus opaque et donc moins de lumière a atteint le détecteur.

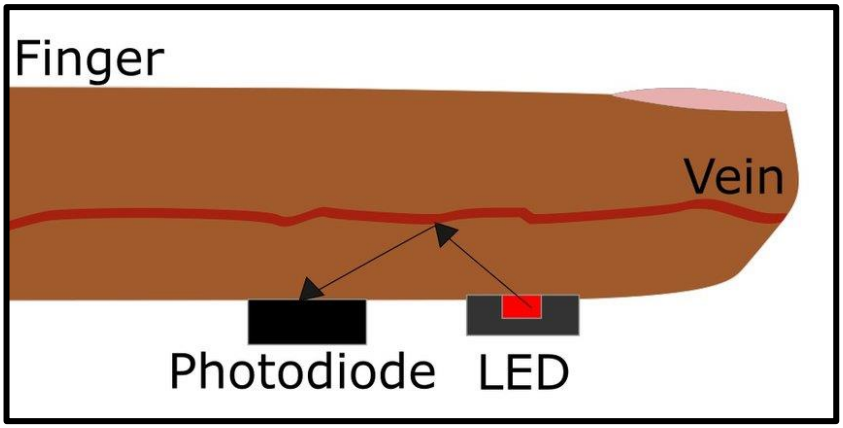


FIGURE 9 : POSITION DU DOIGT SUR LE CAPTEUR

Avec chaque pouls cardiaque, le signal du détecteur varie. Cette variation est convertie en impulsion électrique. Ce signal est amplifié et déclenché par un amplificateur qui délivre un signal de niveau logique + 5V. Le signal de sortie est également indiqué par une LED dans la carte Arduino qui clignote à chaque battement de cœur.

L'image suivante illustre cette opération d'une manière simple :

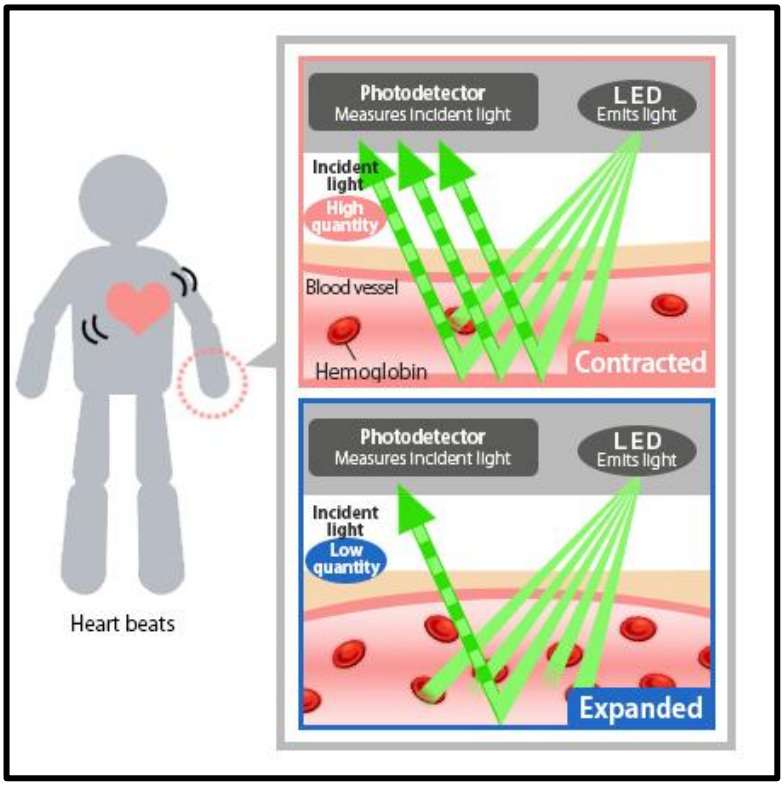


FIGURE 10 : PROPAGATION ET REFLEXION DE LA LUMIERE DANS LA VEINE

3. Montage des différents composants

Le montage final de la carte Arduino, le capteur ECG et la carte Bluetooth de notre projet se résume comme ceci :

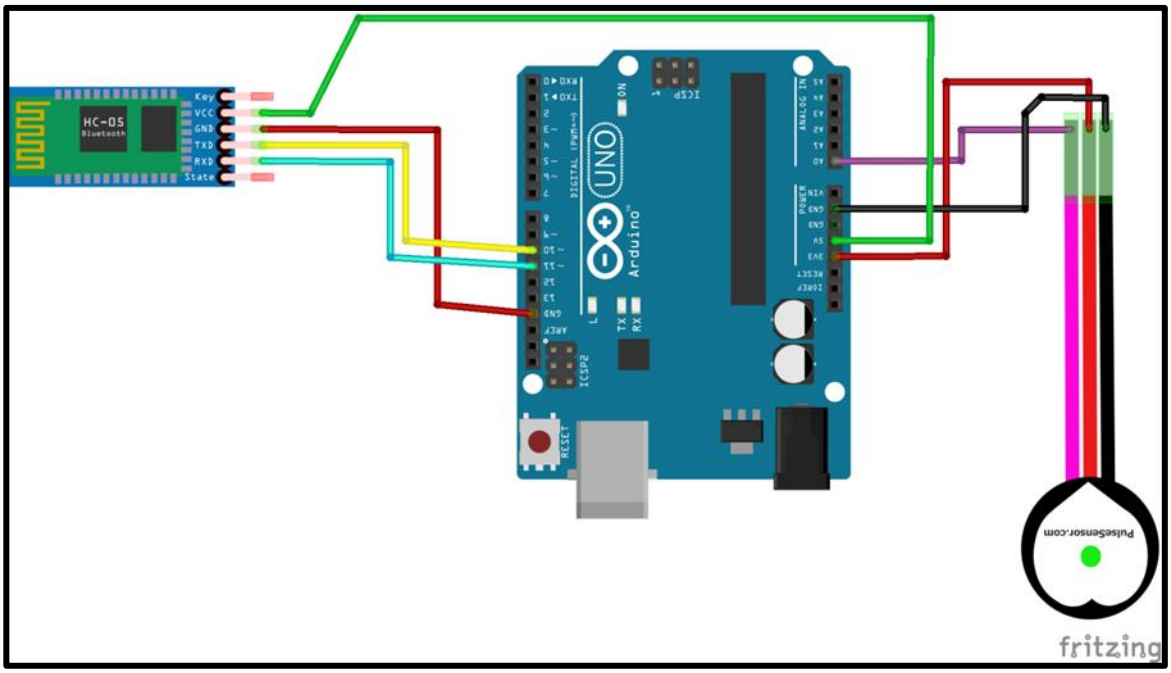


FIGURE 11 : MONTAGE FINAL

Dans cette partie on va lier chaque broche du module Bluetooth à la broche qui la convienne dans la carte Arduino.

- **Circuit carte Arduino / Bluetooth HC-05 :**

Broches Arduino	Broches Bluetooth
Digital broche 11	RX
Digital broche 10	TX
Power broche 5V	VCC
GND	GND

- **Circuit carte Arduino / Capteur XD-58C :**

Broches Arduino	Fonction	Broches XD-58C
Analogique Broche A0	**Sortie du signal**	**Signal OutPut (s)**
Power Broche 3.3V	**Alimentation**	**(+)**
GND	**Ground**	**GND (-)**

4. Programmation sur la carte Arduino

Dans cette partie en s'intéresse au côté programmation sur la carte Arduino et qui nécessite bien sûr un logiciel et un langage de programmation.

LOGO DE IDE ARDUINO

Arduino Integrated Development Environment (IDE) est une application multiplateforme (pour Windows, macOS, Linux) qui est écrite dans des fonctions de C et C ++. Il est utilisé pour écrire et télécharger des programmes sur des cartes compatibles Arduino.

Le code source de l'EDI est publié sous la licence publique générale GNU, version 2. L'IDE Arduino prend en charge les langages C et C ++ en utilisant des règles spéciales de structuration de code. L'Arduino IDE fournit une bibliothèque de logiciels du projet de câblage, qui fournit de nombreuses procédures d'entrée et de sortie courantes. Le code écrit par l'utilisateur ne nécessite que deux fonctions de base, pour démarrer l'esquisse et la boucle du programme principal, qui sont compilées et liées avec un stub main () de programme dans un programme exécutif cyclique exécutable avec la chaîne d'outils GNU, également inclus avec la distribution IDE. L'IDE Arduino utilise le programme Avrdude pour

convertir le code exécutable en un fichier texte au codage hexadécimal qui est chargé dans la carte Arduino par un programme de chargement dans le micro logiciel de la carte.

5. Code source Arduino et Explication :

Ce dernier titre dans cette partie Arduino est consacré au code source avec le langage Arduino et son explication ligne par ligne.

Dans cette partie de programmation sur carte Arduino on a utilisé 3 fichiers :

- Le 1er fichier **Ecg_signal.ino** :

C'est notre fichier principal qui contient les méthodes principales de chaque code Arduino (*setup()* et *loop()*) et c'est à partir duquel on fait appel aux différentes méthodes du fonctionnement du matériel avec le coté logiciel.

```
#include <stdio.h>
#include <SoftwareSerial.h> // c'est une bibliothèque qui permet de faciliter le transfer de données entre la carte arduino et HC-05.

#define SERIAL_PLOTTER  2  //Définition d'une Constante pour le Moniteur Série.
#define ANDROID  3         //Définition d'une Constante pour L'application android.

//Définir le Module bluetooth, et affectation des broches de la carte arduino aux broches de transmition TX et de réception RX du module Bluetooth
SoftwareSerial BTserial(10,11); // Arduino RX(Broche Digital 10) - Bluetooth (Broche TX) | Arduino TX(Broche Digital 11) - Bluetooth (Broche RX)

//  Variables
int pulsePin = 0;               // Fil violet du capteur d'impulsions connecté à la broche analogique 0 (A0)
int blinkPin = 13;              // broche pour clignoter à chaque battement

int fadePin = 5;                 //utilise pour la LED pour le clignotement
int fadeRate = 0;                //utilisé pour atténuer la LED avec PWM sur fadePin

// Ici on a des variables Volatiles Utilisées dans le Service de Routine D'Interruption (ISR) dans le fichier (interrupt.ino)

volatile int BPM;                   //Un int qui contient l'analogique brut en 0. mis à jour tous les 2 ms
volatile int Signal;                // contient les données brutes entrantes
volatile int IBI = 600;             //un int qui contient l'intervalle de temps entre les battements!
volatile boolean Pulse = false;     // "True" lorsque le rythme cardiaque en direct de l'utilisateur est détecté. "Faux" quand ce n'est pas un "battre en direct".
volatile boolean QS = false;        // Devient True si il y'a un battement du coeur capté par l'arduino

// decider si on veut envoyer les valeurs à l'application android ou bien au Traceur Série
//la variable statique utilisé dans le fichier (AllSerialHandling.ino)
static int outputType = ANDROID;
//static int outputType = SERIAL_PLOTTER;
```

```
void setup(){
  pinMode(blinkPin,OUTPUT);          // épingle qui clignotera à votre rythme cardiaque!
  pinMode(fadePin,OUTPUT);           // épingle qui clignotera à votre rythme cardiaque!
  Serial.begin(9600);               // Vitesse de reception de données sur le Moniteur Série en baud
  BTserial.begin(9600);            // Vitesse de transmition de données sur le Bluetooth HC-05
  interruptSetup();                //cette methode du fichier (interrupt.ino) est configuré pour lire le signal du capteur d'impulsions tous les 2 ms

}

//  Notre fonction boucle qui va s'occupeé des changements a chaque itération
void loop(){
  if (QS == true){     // Un battement de coeur a été trouvé
                       // BPM et IBI ont été déterminés
                       // Self quantifié "QS" vrai quand Arduino trouve un battement de coeur
       fadeRate = 255;         // Réalise l'effet de fondu LED
                               // Définissez la variable 'fadeRate' sur 255 pour atténuer la LED avec une impulsion

       serialOutputWhenBeatHappens(); //C'est une méthode dans le fichier (AllSerialHandling.ino)qu'on l'expliquera après, qui va etre appelé si il y a un battement du coeur.

       QS = false;                     // réinitialise l'indicateur Quantified Self pour la prochaine fois
  }

  serialOutput();//C'est une 2ème méthode dans le fichier (AllSerialHandling.ino) qu'on l'expliquera après aussi, qui va etre appelé si il y a un signal capté par le sensor.

  delay(20);  //c'est une méthode arduino pour prendre une pause de 20 ms après que la boucle entre dans une autre itération.

  ledFadeToBeat();                    // Réalise l'effet de fondu LED
}
```

```
//c'est la methode qui s'occupe des effets du LED
void ledFadeToBeat(){
    fadeRate -= 15;                         // définir la valeur de fondu des LED
    fadeRate = constrain(fadeRate,0,255);  // Empêche la valeur de fondu des LED de devenir négative!
    analogWrite(fadePin,fadeRate);          // LED s'estompe (lumiere faible)
  }
```

- Le 2ème fichier **AllSerialHandling.ino** :

C'est le fichier responsable de l'affichage des données sous différentes formes sur le Moniteur Série ou le Traceur Série de IDE Arduino, et aussi l'envoie de ces données vers l'application Android via le Module Bluetooth HC-05.

```
//cette premiere méthode est dedié pour visualiser les donées en sortie soit sur le moniteur série ou sur l'application android.
void serialOutput(){

  switch(outputType){ //la variable statique outputType est définié dans le fichier principale.

    case SERIAL_PLOTTER:  //Pour visaualiser les données dans le Traceur série de IDE Arduino
      Serial.print(BPM);   //les valeurs du BPM (Beat Per Minute), nombre de battements par minute
      Serial.print(",");   //virgule comme séparateur de données
      Serial.print(IBI);   //les valeurs de IBI (Inter Beats Interval) l'interval entre deux battements cardiaques
      Serial.print(",");  //virgule comme séparateur de données
      Serial.println(Signal); //les valeurs du signal ECG
      break;
```

```
    case ANDROID: //Pour visaualiser les données dans l'appliaction Android on les envoies sur Bluetooth

      //Pour envoyer ces données de l'arduino vers android on utilise le module bluetooth déja définie.
      BTserial.print('s');  //la lettre "s" qui precède chaque valeur de signal pour indiquer a l'app android qu'il sagit du signal
      //Pour poser ces données sur bluetooth on utilise la fct "print()"
      BTserial.print(floatMap(Signal,320,360,0,5),2); //la fct floatMap() est une fct définie par nous meme pour la conversion de données avant qu'elles sont envoyées sur bluetooth.
     //la fct floatMap()est de type float, c'est elle qui garantie le bon affichage du graphe ECG sur l'app android par la conversion de données envoyées.
     //voir le script de la fct ci-dessous pour plus détails.
      break;
    default:
      break;
  }

}
```

```
  // Tant que l'arduino recoie des battements de coeur, il fait des calculs spécifiques pour trouver les valeurs de BPM et IBI dans le fichier interrupt.
 // et puis il fait appel a cette fct pour choisir la façon de sortie de ces données soit sur le moniteur série soit l'app android.
//Alors pour effectuer cette derniere il doit envoyer ces données sur bluetooth.

void serialOutputWhenBeatHappens(){
 switch(outputType){
   case ANDROID:
       BTserial.print('b');   //la lettre "b" pour indiquer qu'elles sont des valeurs BPM.
       BTserial.print(BPM);   //la valeur BPM.
       BTserial.print(',');   //séparateur virgule.
       BTserial.print('i');   //la lettre "i" pour indiquer qu'elles sont des valeurs IBI.
       BTserial.print(IBI);   //la valeur IBI
       BTserial.print('e');   //fin
       break;
   default:
     break;
 }
}

//la fct floatMap() de type float dont on a déja parler.
//*************valeurs d'enter ****************
//le parametre "x" pour la valeur du Signal.
//le parametre "inMin" pour la valeur maximal de l'onde du signal.
//le parametre "inMax" pour la valeur minimal de l'onde du signal.
//*************valeurs de sortie ****************
//elles representent aussi l'intervalle de variation de valeurs converties dans notre app android.
//le parametre "outMin" pour la valeur minimal de l'onde du nouveau signal sur android.
//le parametre "outMax" pour la valeur maximal de l'onde du nouveau signal sur android.
```

```
  float floatMap(float x, float inMin, float inMax, float outMin, float outMax){
  return (x-inMin)*(outMax-outMin)/(inMax-inMin)+outMin;
}
```

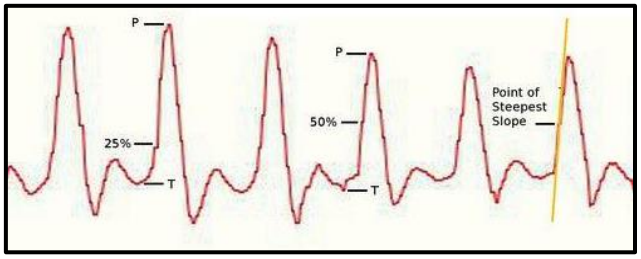
- Le 3ème fichier **Interrupt.ino :**

C’est le fichier le plus important dans cette partie, car c’est lui qui effectue les différents calculs pour avoir de bonnes valeurs de BPM et IBI, en plus de l’organisation de lecture des valeurs capté par le capteur, sur la carte Arduino.

Car il existe de bonnes raisons de ne pas baser l'algorithme de recherche de battements sur des phénomènes d'ondes arbitraires.

Nous voulons trouver le moment instantané du rythme cardiaque. Ceci est important pour le calcul précis du BPM, la variabilité de la fréquence cardiaque (HRV) et mesure du temps de transit pulsé (PTT).

Et c'est un défi digne! Les gens plus intelligents que nous soutiennent que le moment instantané du rythme cardiaque se produit à un moment donné au cours de cette montée rapide de la forme d'onde PPG.



Certains chercheurs en cardiologie disent que c'est lorsque le signal atteint 25% de l'amplitude, certains disent que c'est 50% de l'amplitude, et certains disent que c'est le moment où la pente est la plus raide pendant l'événement de montée.

Ce code du capteur de pouls est conçu pour mesurer l'IBI en chronométrant entre les moments où le signal traverse 50% de l'amplitude de l'onde pendant cette montée rapide. Le BPM est dérivé à chaque battement d'une moyenne des 10 temps IBI précédents.

Voici comment nous procédons! Tout d'abord, il est important d'avoir une fréquence d'échantillonnage régulière avec une résolution suffisamment élevée pour obtenir une mesure fiable de la synchronisation entre chaque battement. Pour ce faire, nous avons configuré Timer2, un minuteur matériel 8 bits sur l'ATmega328 (UNO), afin qu'il déclenche une interruption toutes les deux millisecondes. Cela nous donne un taux d'échantillonnage de 500 Hz et une résolution de synchronisation battement par battement de 2 ms. Cela désactivera la sortie PWM sur les broches 3 et 11.

```
volatile int rate[10];                      // tableau pour contenir les dix dernières valeurs IBI
volatile unsigned long sampleCounter = 0;   // utilisé pour déterminer la synchronisation des impulsions
volatile unsigned long lastBeatTime = 0;    // utilisé pour trouver IBI
volatile int P =360;                        // utilisé pour trouver le pic de l'onde de pouls
volatile int T = 360;                       // utilisé pour trouver un creux dans l'onde de pouls
volatile int thresh = 340;                  // utilisé pour trouver un moment instantané de battement de coeur
volatile int amp = 0;                       // utilisé pour maintenir l'amplitude de la forme d'onde d'impulsion
volatile boolean firstBeat = true;          // utilisé pour amorcer le tableau de taux, donc nous démarrons avec un BPM raisonnable
volatile boolean secondBeat = false;        // utilisé pour amorcer le tableau de taux, donc nous démarrons avec un BPM raisonnable
```

```
//Exemple de Calcul théorique du comptage:
//Le timer 2 sera utilisé en le faisant compter à la fréquence de 62500 Hz (fréquence d'horloge(16 MHz) divisée par 256).
//Un cycle d'horloge dure donc 16 µs (l'inverse de la fréquence). Pour avoir 500 ms (500000 µS),
//il faut compter 500 000 µs / 16 µs = 31250 impulsions pour que cela réalise 500 000 µs=0.5s. Cette valeur est décomposable en 125 * 250.
//Le timer doit compter 250 fois pour déborder ; il suffit de le faire partir de la valeur 6.
//Chaque fois qu'il déborde, une variable compteur est incrémentée ; quand cette variable atteint 125,
//on a bien 500 ms qui se sont écoulées et on agit sur la LED.
//************************************************************
//Dans le mode "CTC" ; le timer compte de 00 à TOP (qui vaut OCR2A soit 124) et lorsqu'il atteint cette valeur, il est remis à zéro et une interruption est générée,
//par le flag OCF2A en positionnant à 1 le bit OCIE2A qui est le bit 1 du registre TIMSK2.
//pour plus de details "https://www.locoduino.org/spip.php?article89"

void interruptSetup(){
// Initialise Timer2 pour lancer une interruption tous les 2 ms.
//les différents registres sur 8bits de contrôle associés au timer.
  TCCR2A = 0x02;     //DÉSACTIVER PWM ( Modulation par Largeur d'Impulsion )SUR LES BROCHES NUMÉRIQUES 3 ET 11 ET PASSER EN MODE CTC (Clear Timer on Compare)
  TCCR2B = 0x06;    // NE PAS FORCER COMPARER,diviser la fréquence de base (62500 Hz) par 256 grace au prédiviseur en microcontroleur .(0x06 est 256 en decimal)c'est le rapport de devision
  OCR2A = 0X7C;     // RÉGLEZ LE HAUT DU COMPTE À 124 POUR UN TAUX D'ÉCHANTILLON DE 500 Hz
  TIMSK2 = 0x02;   // ACTIVER L'INTERRUPTION SUR LE MATCH ENTRE TIMER2 ET OCR2A
  sei();          // ASSUREZ BIEN QUE LES INTERRUPTIONS GLOBALS SONT ACTIVÉES
}
```

Les paramètres d'enregistrement ci-dessus indiquent à Timer2 de passer en mode CTC et de compter jusqu'à 124 (0x7C) encore et encore et encore. Un pré définisseur de 256 est utilisé pour obtenir le bon timing de sorte qu'il faut 2 millisecondes pour compter jusqu'à 124. Un indicateur d'interruption est défini chaque fois que Timer2 atteint 124, et une fonction spéciale appelée routine de service d'interruption (ISR) que nous avons écrite est exécutée au tout prochain moment possible, peu importe ce que fait le reste du programme. sei () garantit que les interruptions globales sont activées.

La seule autre chose dont nous aurons besoin est le bon vecteur ISR à l'étape suivante. Ainsi, lorsque l'Arduino est alimenté et fonctionne avec un capteur de pouls amplifié branché sur la broche analogique 0, il lit constamment (tous les 2 ms) la valeur du capteur et recherche le rythme cardiaque. Voici comment cela fonctionne:

```
//Quand un événement interne (timer) ou bien externe (broche) demande une interruption au microcontrôleur .
//Celui-ci va s'interrompre dans son programme principal, puis va exécuter un sous-programme (on parle de routine d'interruption).
//Une fois que cette routine est exécutée, le microcontrôleur reprend son programme principal,
//là où il s'était arrêté, et continue son exécution.
//Pour cela, il a fallu que le microcontrôleur sauve différents registres sur une pile et les rétablisse en fin de routine d'interruption.

// C'EST LA ROUTINE DE SERVICE D'INTERRUPTION DE TIMER 2.
// Le Timer 2  s'assure que nous prenons une lecture toutes les 2 millisecondes

ISR(TIMER2_COMPA_vect){             // déclenché lorsque Timer2 compte jusqu'à 124
  cli();                           // désactiver les interruptions pendant que nous faisons cela
  Signal = analogRead(pulsePin);  // lire le capteur de pouls
  sampleCounter += 2;            // garde le temps en mS avec cette variable
  int N = sampleCounter - lastBeatTime;   // surveille le temps écoulé depuis le dernier battement pour éviter le bruit
```

Cette fonction est appelée toutes les 2 millisecondes. La première chose à faire est de prendre une lecture analogique du capteur de pouls. Ensuite, nous incrémentons la variable sampleCounter. La variable sampleCounter est ce que nous utilisons pour garder une trace du temps.

La variable N aidera à éviter le bruit plus tard. Ensuite, nous gardons une trace des valeurs les plus élevées et les plus basses de l'onde PPG (Photoplethysmography), pour obtenir une mesure précise de l'amplitude.

```
// trouver le pic et le creux de l'onde de pouls
  if(Signal < thresh && N > (IBI/5)*3){      // évite le bruit dichrotique en attendant 3/5 du dernier IBI
    if (Signal < T){                        // T est l'auge de l'onde
      T = Signal;                          // garde trace du point le plus bas de l'onde de pouls
    }
  }

// la condition de battement permet d'éviter le bruit
  if(Signal > thresh && Signal > P){     // P est le pic de l'onde cardiaque

    P = Signal;                         // garde la trace du point le plus haut de l'onde de pouls
  }
```

Les variables P et T contiennent respectivement les valeurs de crête et de creux. La variable Thresh est initialisée à 340 (milieu de la plage analogique) et change pendant l'exécution pour suivre un point à 50% d'amplitude comme nous le verrons plus loin. Il y a une période de 3/5 IBI qui doit s'écouler avant que T ne soit mis à jour pour éviter le bruit et les fausses lectures de l'encoche dichroïque.

Maintenant, vérifions et voyons si nous avons un pouls.

```
// MAINTENANT, IL EST TEMPS DE CHERCHER LES BATTEMENTS DU CŒUR

// le signal augmente en valeur chaque fois qu'il y a une impulsion
  if (N > 250){                                   // évite le bruit haute fréquence

    if ( (Signal > thresh) && (Pulse == false) && (N > (IBI/5)*3) ){

      Pulse = true;                               // définir l'indicateur Pulse quand on pense qu'il y a une impulsion
      digitalWrite(blinkPin,HIGH);               // allume la broche 13 LED
      IBI = sampleCounter - lastBeatTime;       // mesure le temps entre les battements en mS
      lastBeatTime = sampleCounter;            // garde le temps pour la prochaine impulsion
```

Avant même d'envisager de rechercher un battement de cœur, un minimum de temps doit s'écouler. Cela permet d'éviter le bruit à haute fréquence. 250 millisecondes minimum N place une limite supérieure de 250 BPM. Si vous vous attendez à avoir un BPM plus élevé, ajustez-le en conséquence et consultez un médecin. Lorsque la forme d'onde dépasse la valeur de seuil et que 3/5 du dernier IBI est passé, nous avons une impulsion! Il est temps de définir le drapeau Pulse et d'allumer la LED pulsePin. (Remarque: si vous voulez faire autre

chose avec la broche 13, commentez cette ligne et la dernière aussi). Ensuite, nous calculons le temps écoulé depuis le dernier temps pour obtenir l'IBI et mettons à jour le lastBeatTime.

Le bit suivant est utilisé pour s'assurer que nous commençons avec une valeur BPM réaliste au démarrage.

```
    if(secondBeat){                        // si c'est le deuxième battement, si secondBeat == TRUE
      secondBeat = false;                  // efface l'indicateur secondBeat
      for(int i=0; i<=9; i++){             // amorce le total cumulé pour obtenir un BPM réaliste au démarrage
        rate[i] = IBI;
      }
    }

    if(firstBeat){                         // si c'est la première fois que nous trouvons un battement, si firstBeat == TRUE
      firstBeat = false;                   // effacer l'indicateur firstBeat
      secondBeat = true;                   // définir le deuxième indicateur de temps
      sei();                               // réactiver les interruptions
      return;                              // La valeur IBI n'est pas fiable, alors jetez-la
    }
```

Le booléen firstBeat est initialisé comme true et secondBeat est initialisé comme false au démarrage, donc la toute première fois que nous trouvons un beat et que nous arrivons aussi loin dans l'ISR, nous sommes expulsés par le retour; dans le premier temps conditionnel. Cela finira par jeter la première lecture IBI, car c'est moche. La deuxième fois, nous pouvons faire confiance (plus ou moins) à l'IBI et l'utiliser pour amorcer le tableau rate [] afin de commencer avec un BPM plus précis. Le BPM est dérivé d'une moyenne des 10 dernières valeurs IBI, d'où la nécessité de semer. Permet de calculer le BPM!

```
// conserve un total cumulé des 10 dernières valeurs IBI
      word runningTotal = 0;                  // effacer la variable runningTotal

      for(int i=0; i<=8; i++){                // déplacer les données dans le tableau des taux et supprimez la valeur IBI la plus ancienne
        rate[i] = rate[i+1];
        runningTotal += rate[i];              // additionne les 9 valeurs IBI les plus anciennes
      }

      rate[9] = IBI;                          // ajoute le dernier IBI au tableau des taux
      runningTotal += rate[9];                // ajouter la dernière IBI à runningTotal
      runningTotal /= 10;                     // moyenne des 10 dernières valeurs IBI
      BPM = 60000/runningTotal;               // combien de temps peut tenir une minute? c'est la valeur BPM!
      QS = true;                              // définir l'indicateur d'auto quantifié (Quantified Self)
                                              // L' INDICATEUR QS N'EST PAS DÉGAGÉ À L'INTÉRIEUR DE CET ISR
    }
  }
```

Tout d'abord, nous saisissons une grande variable, **runningTotal**, pour collecter les IBI, puis le contenu de **rate []** est déplacé et ajouté à **runnungTotal**. L'IBI le plus ancien (il y a 11 temps) tombe hors de la position 0, et l'IBI frais est mis en position 9. Ensuite, c'est un processus simple pour faire la moyenne du tableau et calculer le BPM. La dernière chose à faire est de définir le drapeau QS (abréviation de Quantified Self, de super supporters du kickstarter!) Afin que le reste du programme sache que nous avons trouvé le rythme. C'est tout pour les choses à faire quand on trouve le rythme.

Il y a quelques autres extrémités lâches qui doivent être attachées avant que nous ayons fini, comme trouver le non-beat.

```
  if (Signal < thresh && Pulse == true){    // lorsque les valeurs baissent, le rythme du battement est terminé
    digitalWrite(blinkPin,LOW);            // éteindre la broche 13 LED
    Pulse = false;                         // réinitialise le drapeau Pulse pour que nous puissions recommencer
    amp = P - T;                           // obtenir l'amplitude de l'onde d'impulsion
    thresh = amp/2 + T;                    // régler le seuil à 50% de l'amplitude
    P = thresh;                            // réinitialise ces derniers (P , T) pour la prochaine fois
    T = thresh;
  }
```

L'impulsion a été déclarée vraie lors de l'augmentation du signal du capteur de pouls lorsque nous avons trouvé le battement, ci-dessus, donc lorsque le signal franchit un seuil descendant, nous pouvons comprendre que l'impulsion est terminée. Un peu de ménage dans la compensation de **pulsePin** et du **Pulse** booléen. Ensuite, l'amplitude de l'onde qui vient de passer est mesurée et le seuil est mis à jour avec la nouvelle marque de 50%. P et T sont réinitialisés au nouveau seuil. L'algorithme est maintenant amorcé et prêt à trouver le rythme suivant.

Il y a une autre question à poser avant que l'ISR soit terminé. Et s'il n'y a pas de battements?

```
  if (N > 2500){                           // si 2,5 secondes passent sans un battement
    thresh = 340;                          // définir le seuil par défaut 340
    P = 360;                               // définir P par défaut 360
    T = 360;                               // définir T par défaut 360
    lastBeatTime = sampleCounter;          // mettre à jour le lastBeatTime
    firstBeat = true;                      // régler (firstBeat et secondeBeat) pour éviter le bruit quand on reprend le rythme cardiaque
    secondBeat = false;
  }

  sei();                                   // activer les interruptions lorsque vous avez terminé!
}// fin isr
```

S'il n'y a pas d'événement de battement pendant 2,5 secondes, les variables utilisées pour trouver le rythme cardiaque sont réinitialisées aux valeurs de démarrage. C'est la fin de l'ISR!

En utilisant l'interruption Timer2, notre algorithme de recherche de battements s'exécute en arrière-plan et met à jour automatiquement les valeurs des variables. Voici une liste de variables utiles et la fréquence à laquelle elles sont mises à jour.

Nom de variable	Fréquence de rafraîchissement	Signification
Signal	2 ms	signal brut du capteur d'impulsions
IBI	chaque battement	temps entre les battements cardiaques en ms.
BPM	chaque battement	battements par minute
QS	rendre vrai à chaque battement	doit être effacé par l'utilisateur
Impulsion	rendre vrai à chaque battement	autorisé par ISR

IV. PARTIE ANDROID

1. Connexion de l'application et l'Arduino via le module de Bluetooth

Dans ce projet, on a utilisé le module de Bluetooth HC-05 qui va lier entre l'application Android et la carte Arduino. L'établissement d'une connexion via le Bluetooth est basé sur le modèle client-serveur. Le serveur créer une socket-serveur, qui attend les requêtes de connexion, après le client crée une socket client qui va envoyer une requête de connexion au serveur. Le serveur accepte cette requête est génère un deuxième socket qui va être utilise pour communiquer avec le client. Et maintenant on est capable d'envoyer et de recevoir les données dans les deux sens. La figure ci-dessous montre ce processus.

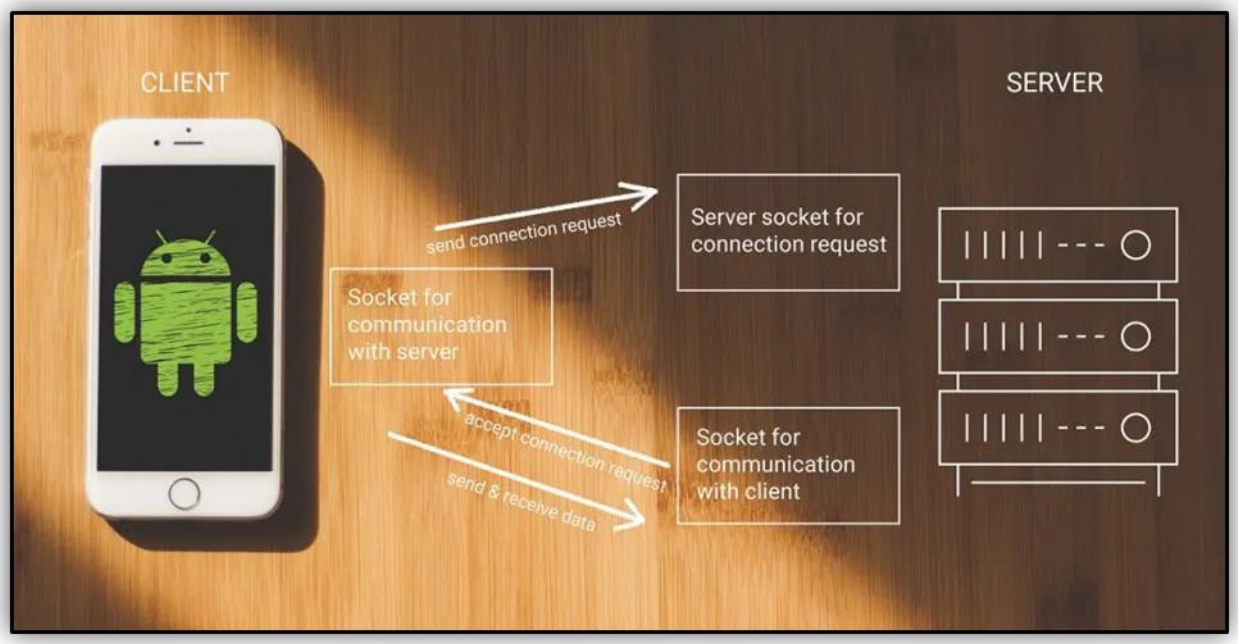


FIGURE 12 : L'ETABLISSEMENT D'UNE CONNEXION BLUETOOTH

Maintenant, pour établir une connexion entre deux dispositifs on doit les coupler pour une seule fois. Le processus est le suivant : on met le dispositif en mode découvrable, dans notre cas le HC-05 est automatiquement en mode découvrable. Puis, on scan pour trouver les diapositifs désormais. Lorsqu'on détecte le Bluetooth souhaiter on l'envoie une demande d'appairage.

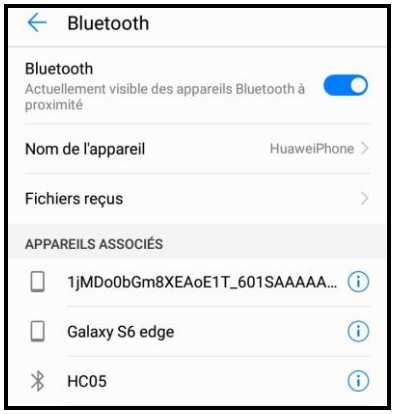


FIGURE 13 : APPAIRAGE DE MODULE BLUETOOTH HC-05

Maintenant, dans le projet d'Android Studio on ajoute les permissions nécessaires à l'application.

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

Comme on voit aussi dans le fichier '*AndroidManifest.xml*' que la première activité qui se lance est '*MainActivity*'.

```
<application
    android:label="ECG Application"
    android:icon="@drawable/logo"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".MainActivity"
        >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

L'interface graphique de cette activité est présentée ci-dessous :

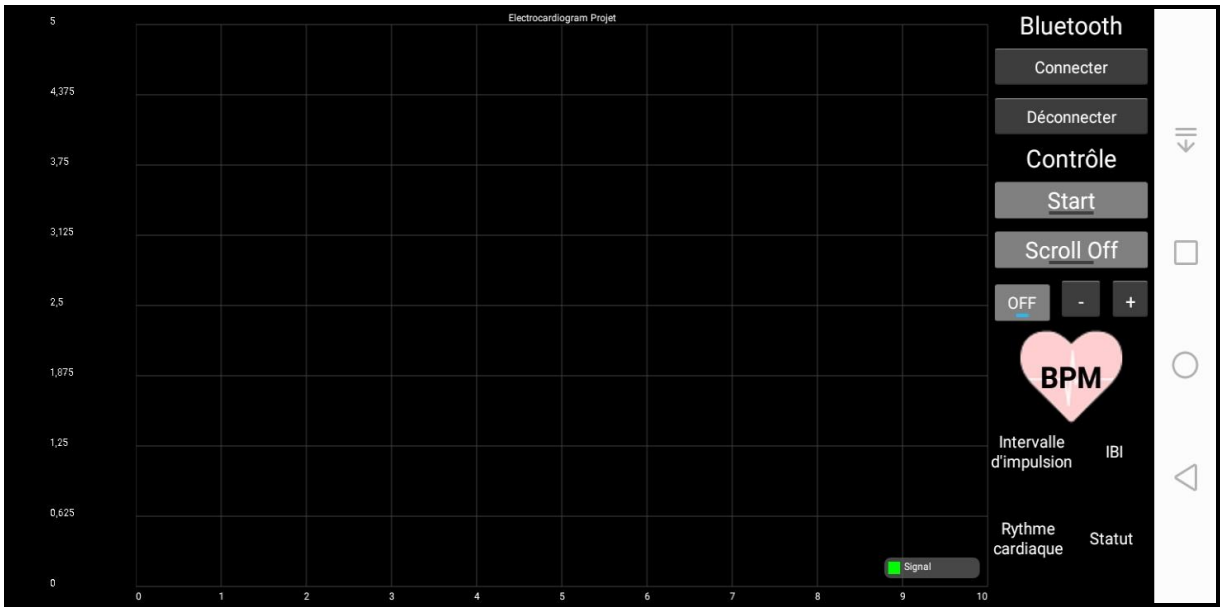


FIGURE 14 : INTERFACE GRAPHIQUE DE L'APPLICATION

Pour commencer la connexion via le Bluetooth avec la carte Arduino, on clique sur le bouton '*Connecter*'. Ce bouton est associé à un écouteur.

```
//Initialisation de bouton Connect
bConnect = (Button)findViewById(R.id.bConnect);
bConnect.setOnClickListener(this);
```

A l'aide d'une '*switch case*' on va déterminer quel bouton a été cliqué.

```
@Override
public void onClick(View v) {
    // TODO Auto-generated method stub
    switch(v.getId()){
        // Le cas ou on clique sur le bouton Connect
        case R.id.bConnect:
            startActivity(new Intent( action: "android.intent.action.BT1"));
            break;
        // Le cas ou on clique sur le bouton Disconnect
        case R.id.bDisconnect:
```

Au sein de ce cas, on a un traitement à faire, c'est de lancer les activités qui répond à l'intention avec l'action 'android.intent.action.BT1' (BT1 est une simple chaine de caractère on peut la changer comme on veut). Dans le fichier 'AndroidManifest.xml' on voit que l'activité 'Bluetooth' répondre à ce type d'actions.

```
    <activity
        android:name=".Bluetooth"
        android:label="Bluetooth" >
        <intent-filter>
            <action android:name="android.intent.action.BT1" />

            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>
</application>
```

Alors, l'activité 'Bluetooth' se lance, la fonction 'onCreate()' commence à s'exécuté dont on effectue toutes les initialisations nécessaire (tous les widgets qui on a créé pour former l'interface de l'activité), ainsi que le 'Broadcast Receiver' qui sert à identifier l'action extraite de l'objet intent, est se basant sur cette action on exécute un traitement. Ensuite, on effectue des tests pour vérifier est ce que le Bluetooth est supporté ou non et on active le Bluetooth avec la fonction 'turnOnBT()' s'il n'est pas activé. Puis, on exécute la fonction 'getPairedDevices()' pour avoir la liste des dispositifs appairé avec le téléphone. Ensuite, on exécute la fonction 'startDiscovery()' pour scanné et détecté les dispositifs Bluetooth disponible autour de téléphone.

```
    //On test est ce que le bluetooth est supporté ou non par le téléphone
    if (btAdapter==null){
        Toast.makeText( context: this, text: "No bluetooth detected", Toast.LENGTH_SHORT).show();
        finish();
    }else{ // Si il est supporté on test est ce qu'il est activer ou non
        if (!btAdapter.isEnabled()){
            // S'il n'est pas activé on exécute la fonction turnOnBT() pour l'activé
            turnOnBT();
        }
        // On cherche les dispositifs appairé avec notre téléphone
        getPairedDevices();
        // On met
        startDiscovery();
    }
```

Ici, le 'Broadcast Receiver' écoute de telle sorte que lorsque la fonction 'startDiscovery()' a émet un intente avec l'action 'ACTION_DISCOVERY_STARTED' le 'Broadcast Receiver' capte cet objet la et exécute un 'Toast' avec le message 'Searching for devices'.

```
// Lorsque la fonction startDiscovery() a émet un intente avec l'action 'ACTION_DISCOVERY_STARTED'
}else if (BluetoothAdapter.ACTION_DISCOVERY_STARTED.equals(action)){
    Toast.makeText(getApplicationContext(), text: "Searching for devices", Toast.LENGTH_SHORT).show();
```

Aussi le 'Broadcast Receiver' écoute de telle sorte que lorsque la fonction 'startDiscovery()' a détecté un nouveau dispositif donc il envoie une action de type 'ACTION_FOUND' est le 'Broadcast Receiver' capte cet objet intente et récupère les informations de ce dispositif Bluetooth et il l'ajout vers la liste 'listview' qui contient la liste des dispositifs Bluetooth appairé avec le téléphone et sont disponible pour la connexion.

```
// Lorsque la fonction startDiscovery() a détecté un nouveau dispositif
if (BluetoothDevice.ACTION_FOUND.equals(action)){
    BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
    devices.add(device);
    String s = "";
    for(int a=0;a<pairedDevices.size();a++){
        if(device.getName()!=null && pairedDevices.get(a)!=null){
            if (device.getName().equals(pairedDevices.get(a))){
                //append
                s = "(Paired)";
                listAdapter.add(device.getName()+" "+s+" "+"\n"+device.getAddress());
            }
        }
    }
```

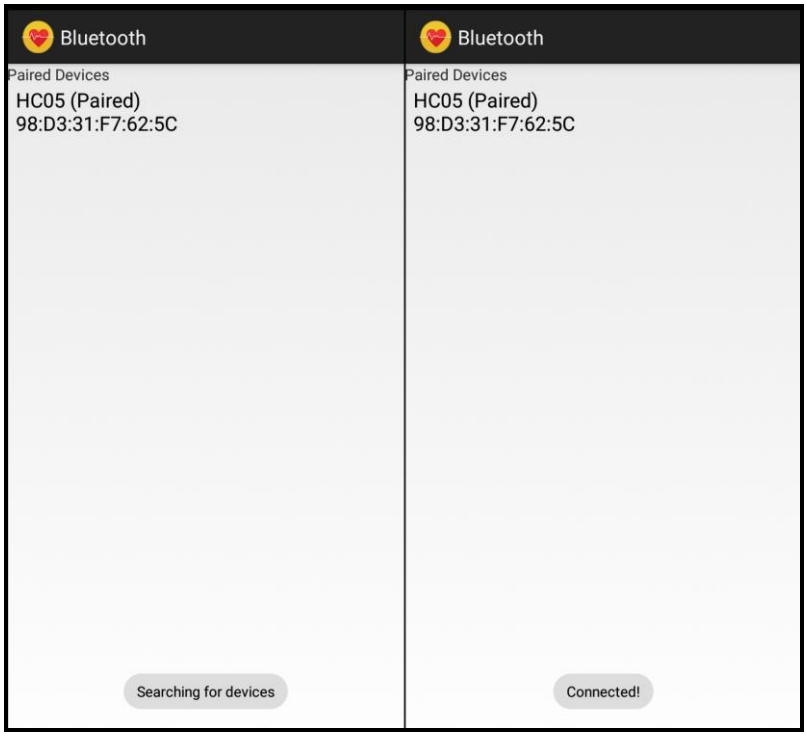


FIGURE 15 : CONNEXION AU MODULE HC-05

Maintenant, la liste des dispositifs disponible est affiché, on clique sur le module 'HC-05' pour connecter via Bluetooth avec la carte Arduino. Lorsqu'on clique sur un dispositif la fonction 'OnItemClick()' commence à s'exécuté. On test est ce que le dispositif qui on a cliqué sur est appairé avec le téléphone ou non, si oui on passe le dispositif sélectionner au constructeur de Thread 'ConnectThread' est on commence l'exécution de ce Thread.

```
@Override
public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
    // TODO Auto-generated method stub
    if (btAdapter.isDiscovering()){
        btAdapter.cancelDiscovery();
    }
    if (listAdapter.getItem(arg2).contains("(Paired)")){

        BluetoothDevice selectedDevice = devices.get(arg2);
        ConnectThread connect = new ConnectThread(selectedDevice);
        connect.start();
    }else {
        Toast.makeText( context: this,  text: "device is not paired", Toast.LENGTH_SHORT).show();
         }
}
```

On obtient un socket client pour effectuer la connexion via le bluetooth en utilisant la méthode ‘createRfcommSocketToServiceRecord(MY_UUID)’ et dans de la fonction ‘run()’ de Thread on effectue le connexion via la fonction ‘connect()’.

Maintenant, le graphe qui va afficher les valeurs lues par le capteur est créé dans l’activité ‘MainActivity’ et la connexion de Bluetooth et la récupération des valeurs est effectuer dans l’activité ‘Bluetooth’, donc comment on va afficher les valeurs dans le graphe ? Pour remédier à ce problème on utilise un ‘Handler’ qui sert à envoyer des messages entre le Thread est une activité spécifier.

Lorsque la connexion est établie avec succès, en utilise un handler pour envoyer un message vers l’activité ‘MainActivity’. Le message envoyer contient un message que la connexion a était établie avec succès et le socket client ‘mmSocket’ qui sert a utilisé pour recevoir les données d’après le dispositif connecter avec.

```
        }
        // On affiche un Toast dans MainActivity et on lance un autre thread via un Handler
        mHandler.obtainMessage(SUCCESS_CONNECT, mmSocket).sendToTarget();
    }
```

Maintenant, on maigre vers l’activité ‘MainActivity’ ou on a initialisé l’objet ‘mHandler’. Au sein de la fonction ‘handleMessage(Message msg)’, on récupère le message envoyer depuis l’activité ‘Bluetooth’ et on effectue des tests à l’aide d’une ‘switch case’.

Dans le cas où le message envoyer correspond a ‘SUCCESS_CONNECT’ c’est-à-dire que la connexion a été établie avec succès, on affiche un Toast informant que la connexion a été établie avec succès et on commence l’exécution d’un deuxième Thread, qui va gérer les données envoyer dans les deux sens, en passant au constructeur le socket client récupérer grâce au handler.

```
Handler mHandler = new Handler(){
    @TargetApi(Build.VERSION_CODES.ECLAIR)
    @RequiresApi(api = Build.VERSION_CODES.ECLAIR)
    @Override
    public void handleMessage(Message msg) {
        // TODO Auto-generated method stub
        super.handleMessage(msg);
        switch(msg.what){
            case Bluetooth.SUCCESS_CONNECT:
                Bluetooth.connectedThread = new Bluetooth.ConnectedThread((BluetoothSocket)msg.obj);
                Toast.makeText( context: MainActivity.this, text: "Connected!", Toast.LENGTH_SHORT).show();
                String s = "successfully connected";
                Bluetooth.connectedThread.start();
                break;
```

2. Envoie et récupération des données par l'application

Une fois la connexion via le Bluetooth est effectuée on est donc capable d'envoyer les données dans les deux sens. Dans l'application on va récupérer les données envoyer par le capteur et les afficher dans un graphe. Pour cela, on a utilisé un Thread qui gère les données envoyer/récupérer.

On va commencer l'exécution de Thread 'connectedThread' en utilisant la fonction 'Bluetooth.connectedThread.start()'. Dans le constructeur on recupere l'objet socket client qui on va utiliser pour recevoir les valeurs de BPM. On utilise ' InputStream' pour contrôler les transmissions par le client socket. Et on obtient les données en utilisant la fonction 'bytes = mmInStream.read(buffer); '

Et enfin en envoie un message vers l'activité 'MainActivity' via le handler avec le message 'MESSAGE_READ' et l'objet 'bytes' qui contient les valeurs.

```
public void run() {

    byte[] buffer;  // buffer pour stoquer le flot des donnees
    int bytes; //  bytes returner par la methode read()
    // Ecoute de Inputstream jusqu'a une exception est leve
    while (true) {
        try {
            try {
                sleep( millis: 30);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            buffer = new byte[1024];
            // Lire depuis InputStream
            bytes = mmInStream.read(buffer);
            // Envoyer les bytes recuperer vers MainActivity
            mHandler.obtainMessage(MESSAGE_READ, bytes,  arg2: -1, buffer).sendToTarget();
        } catch (IOException e) {
            break;
        }
    }
```

Dans l'activité 'MainActivity', on récupère l'objet bytes, et on crée une chaine de caractères. Donc la chaine de caractères 'strIncom' contient la valeur envoyer par le capteur,

la chaine de caractères ‘bpm’ contient la valeur de BPM (Beats Per Minute) et la chaine de caractères ‘IBI’ contient la valeur de IBI (Interbeat Interval).

```
case Bluetooth.MESSAGE_READ:
    if(tbStream.isChecked()){
        byte[] readBuf = (byte[]) msg.obj;
        String strIncom = new String(readBuf, offset: 0, length: 5);  // On cree une chaine de caracteres a partir du table bytes
        String bpm=new String(readBuf, offset: 0, length: 80);
        String ibi=new String(readBuf, offset: 0, length: 80);
```

3. L’affichage des valeurs de capteur ECG dans un graphe

Jusqu’à maintenant, on est capable d’effectuer une connexion via le module d Bluetooth, récupérer les valeurs obtenues par le capteur ECG, mais on est besoin d’un graphe qui va afficher les valeurs. On a utilisé le package ‘*jjoe64/GraphView*’

Premièrement on ajoute le package dans le projet. Dans ‘*MainActivity.java*’ on va instancier toutes les composantes, les buttons et le graphe. Puis on va customiser le graphe.

Le variable ‘GraphView’ c’est le widget qui représente le graph dans l’interface graphique. L’objet ‘series’ représente l’ensemble des points qui constituent le graphe. On va utiliser cet objet pour ajouter les points lus au graph.

On spécifie que la couleur de graphe est verte, le titre est ‘Electrocardiogram Projet’ et que l’épaisseur de la ligne de graphe égale à 2. Et on ajoute l’objet ‘series’ dans l’objet ‘graphView’ et enfin on affiche le graph en ajoutant ‘graphView’ dans ‘GraphView’.

```
void init(){
    Bluetooth.gethandler(mHandler);

    //init de graphview
    GraphView = (LinearLayout) findViewById(R.id.Graph);
    Series = new GraphViewSeries( description: "Signal",
            new GraphViewStyle(Color.GREEN, thickness: 3),//couleur et l'épaisseur de la ligne de graphe
    new GraphViewData[] {new GraphViewData( valueX: 0, valueY: 0)});
    graphView = new LineGraphView(
            context: this // context
            , title: "Electrocardiogram Projet" // Titre
    );
    graphView.setViewPort( start: 0, Xview);
    graphView.setScrollable(true);
    graphView.setScalable(true);
    graphView.setShowLegend(true);
    graphView.setLegendAlign(LegendAlign.BOTTOM);
    graphView.setManualYAxis(true);
    graphView.setManualYAxisBounds( max: 5, min: 0);
    graphView.addSeries(Series); // data
    GraphView.addView(graphView);
}
```

Maintenant, pour chaque valeur lue de message contenue dans le ‘mHandler’ et contenue dans ‘strIncom’ on va tester est ce que la chaine contient un ‘.’ (point) a la position 2 et qu’elle contient ‘s’au début. Avec ce test, on confirme qu’il s’agit bien d’une valeur d’envoyer par le capteur, car dans le code d’Arduino, les valeurs sont envoyées précédé par un ‘s’.

```
case ANDROID:
  BTserial.print('s');
  BTserial.print(floatMap(Signal,0,1023,0,5),2);
```

Si le test au-dessus est vérifier, on remplace le caractère ‘s’ par un espace et on converti la valeur en Double. On ajoute la valeur par l’instanciation de classe ‘GraphViewData’ qui supporte deux valeurs, la première pour l’axe des abscisses, et la deuxième pour l’axe d’ordonne qui correspond à la valeur envoyer par le capteur. La valeur ‘graph2LastXValue’ d’axe des abscisses et incrémentée dès qu’en veut ajouter un nouveau point au graphe. La variable ‘graph2LastXValue’ est initialement initialiser par 0. Et vers la fin, on ajoute le nouveau point au variable ‘series’ en utilisant la fonction ‘appendData()’. Mais ce processus est fait si est seulement on la condition ‘tbStream.isChecked()’ est vérifier, c’est-à-dire on a cliqué sur le bouton ‘start’ pour commencer l’affichage des valeurs dans le graphe.

```
case Bluetooth.MESSAGE_READ:
    if(tbStream.isChecked()){
        byte[] readBuf = (byte[]) msg.obj;
        String strIncom = new String(readBuf, offset: 0, length: 5);  // On cree une chaine de caracteres a partir du tab
        String bpm=new String(readBuf, offset: 0, length: 80);
        String ibi=new String(readBuf, offset: 0, length: 80);

        //Log.d("strIncom", strIncom);

        if (strIncom.indexOf('.')==2 && strIncom.indexOf('s')==0){
            strIncom = strIncom.replace( target: "s", replacement: "");
            if (isFloatNumber(strIncom)){
                Series.appendData(new GraphViewData(graph2LastXValue,Double.parseDouble(strIncom)),AutoScrollX);
```

Maintenant, on passe pour ajouter les valeurs de ‘BPM’ a l’activité, pour ajouter cette valeur, on crée deux variables temporaires : ‘temp1’ et ‘temp2’. Ces deux variables servent à mémoriser la place de caractère ‘b’ pour la première variable, et l’indice de ‘,’ (virgule) pour la deuxième variable. On effectue ce test pour vérifier est ce que l’indice de ‘b’ égale ou supérieur à 0, et que l’indice de ‘,’ est supérieur à ce de ‘temp1’. Ces tests son nécessaire car la valeur de ‘BPM’ est envoyée par la carte Arduino sous un format bien préciser, la valeur de BPM est précédée par un ‘b’ et se termine par une ‘,’.

```
BTserial.print('b');
BTserial.print(BPM);
BTserial.print(',');
```

Après ce test, on soustrait la valeur de BPM, on l’associe à la valeur de champ de texte ‘BPM’ et on associe une image pour l’arrière-plan.

```
int temp1,temp2;
temp1=bpm.indexOf('b');
temp2=bpm.indexOf(',');
if(temp1>=0 && temp2>=temp1){
    bpm = bpm.substring(temp1+1,temp2);
    TextView BPM = (TextView) findViewById (R.id.BPM);
    BPM.setText(bpm);
    BPM.setBackgroundResource(R.drawable.heartbeat);
```

On a aussi programmé que lorsque on a une ‘heart beat’ on va voir un son. Pour cela, on va utiliser le classe ‘ToneGenerator’ pour générer un son lorsqu’on a le bouton ‘on’ activé et on a une valeur de BPM inferieur a la valeur 100.

```
//beep
int beepBPM = Integer.parseInt(bpm);
if(beep.isChecked()){
    if(beepBPM < 100){ // NE COMPTER QUE LES VALEURS DE BPM INF A 100
        try {
            ToneGenerator toneG = new ToneGenerator(AudioManager.STREAM_ALARM, 70);
            toneG.startTone(ToneGenerator.TONE_CDMA_ALERT_CALL_GUARD, 200);
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

A l'aide de la valeur de 'BPM' (Beat Per Minute) on peut tester est ce que le rythme de cœur est régulier ou non. Pour cela, on test est ce que la valeur de BPM est inférieur à 100 ou non, si oui donc on conclut que le rythme de cœur est régulier.

```
if(beepBPM > 100){
    TextView arrythmia = (TextView) findViewById (R.id.arrythmiaStatus);
    arrythmia.setText("irrégulier");
    arrythmia.setTextColor(Color.BLACK);
    arrythmia.setBackgroundResource(R.color.red);

}else{
    TextView arrythmia = (TextView) findViewById (R.id.arrythmiaStatus);
    arrythmia.setText("régulier");
    arrythmia.setTextColor(Color.BLACK);
    arrythmia.setBackgroundResource(R.color.white);
}
```

Il reste d'affecter la valeur de 'IBI' au champ de texte correspondant. On utilise encore une fois, les variables temporaire 'temp1' et 'temp2' pour effectuer des tests. La valeur 'IBI' est envoyée par la carte Arduino sous ce format :

```
BTserial.print('i');
BTserial.print(IBI);
BTserial.print('e');
```

Si le test est vérifié on ajoute la valeur au champs correspondant.

```
temp1=ibi.indexOf('i');
temp2=ibi.indexOf('e');

if(temp1>=0 && temp2>=temp1){
    ibi = ibi.substring(temp1+1,temp2);
    old_interval=new_interval;
    new_interval=Integer.parseInt(ibi);
    TextView IBI = (TextView) findViewById (R.id.IBI);
    IBI.setText(ibi);
```

Et voilà le résultat final de l'application :

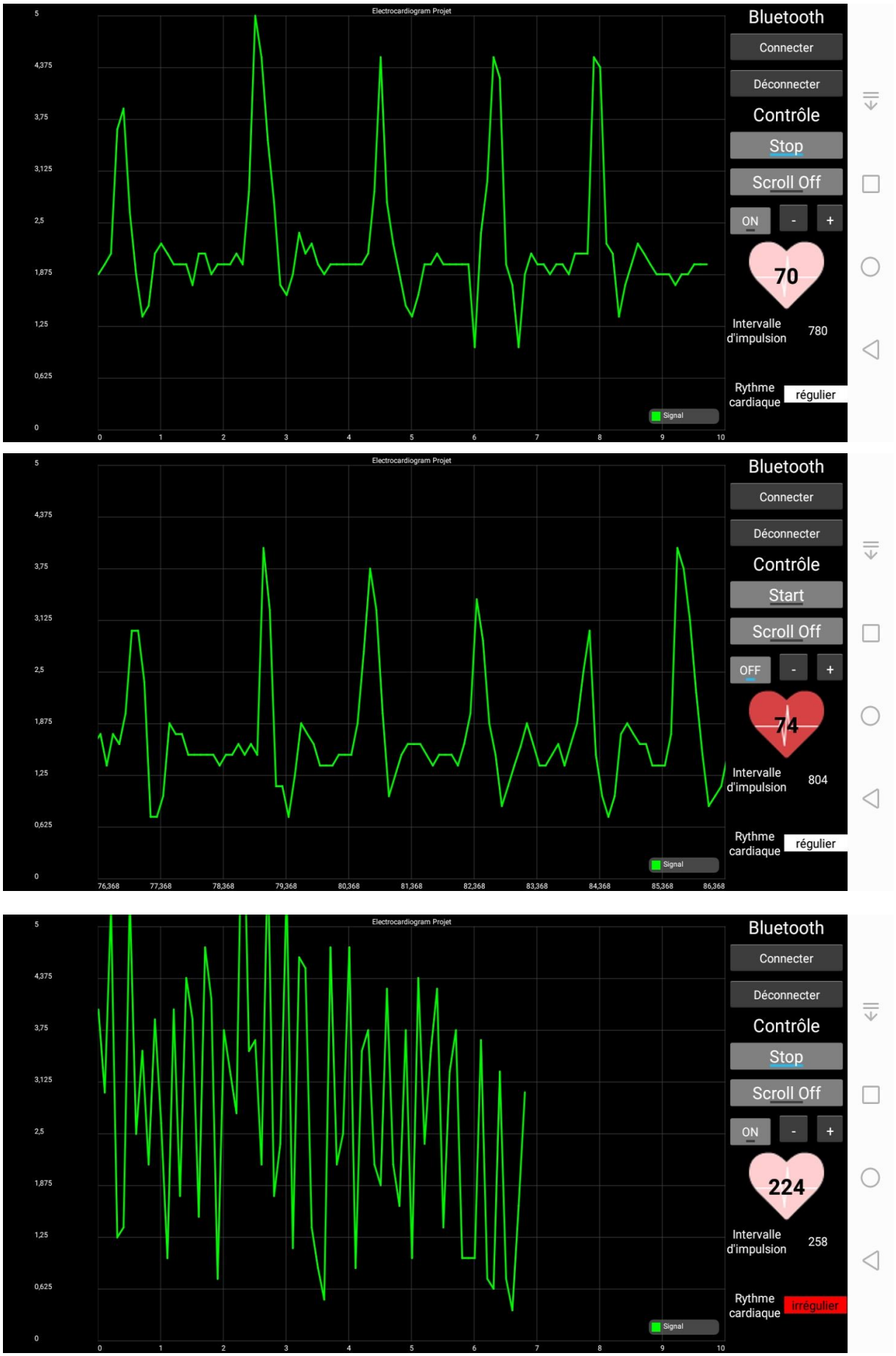


FIGURE 16 : AFFICHAGE FINALE DE L'APPLICATION

V. CONCLUSION

Ce travail, par ces multiples aspects nous a permis de nous rendre compte sur le traitement d'un signal ECG et l'affichage de ce signal dans une application Android en utilisons la carte Arduino et un capteur de signal ECG.

Il a fallu d'abord réfléchir sur la problématique du sujet et effectuer une recherche bibliographique avant la mise en place de notre travail. Cette recherche nous a été utile aussi bien pour le choix d'un bon signal ECG. Cette recherche bibliographique nous a guidées aussi bien pour les méthodes de détection de pics R de signal et aussi la détection des BPM du cœur ainsi que les IBI.

Ce projet nous a permis de nous familiariser avec la carte Arduino et améliorer nos compétences en Android avec lequel nous avons implémenté notre interface de traitement.

Ainsi nous avons pu appréhender les différentes étapes nécessaires à la conception et à la mise en place d'un système de traitement de données envoyé par la carte Arduino et capturé par l'application Android en utilisant le module Bluetooth HC-05.

Ce projet peut être amélioré par la détection d'autres caractéristiques qu'un signal ECG présente, et aussi par l'utilisation des notions de Deep Learning et de Réseau de Nouerons pour faire une classification de ce signal et aussi pour analyser et classer ces données dans le but d'avoir un diagnostic judicieux.

À la fin de ce rapport, nous tenons à remercier notre cher professeur **Mr. Jamal RIFFI** pour la passion qu'il nous a communiqué pour ce travail et nous espérons être à la hauteur de ses attentes.

VI. BIBLIOGRAPHIE ET WEBOGRAPHIE

- Andrew R. HOUGHTON et David GRAY. 2003. Editions MASSON : Maîtriser l'ECG de la théorie à la clinique.
- Dale DUBIN.1999.Editions MALOINE : Lecture accélérée de l'ECG.
- Référentiel : Collège National des Enseignants de Cardiologie.
- https://fr.wikipedia.org/wiki/%C3%89lectrocardiographie
- https://pulsesensor.com/
- https://github.com/WorldFamousElectronics/PulseSensor_Amped_Arduino/blob/master/PulseSensorAmped_Arduino_1.5.0/Timer_Interrupt_Notes.ino
- https://github.com/jjoe64/GraphView/wiki