

Rapport de projet – MEA3 S5

PROJET ACTIMETRIE



Projet réalisé par :

Benjamin SAIGNE
Adam MABROUQUE
Aurélien GOUMAIN
Abel DIGUET
Zyed KAMIL
Pierre-Marius CAMASSE
Alexandre DE JESUS MARTINS

Projet encadré par :

M. Guy CATHEBRAS
M. Michel FACERIAS
M. Arnaud VENA
MME. Karen GODARY DEJEAN
M. François MIRABEL
M. Fabien SOULIER



SOMMAIRE

A. PRÉSENTATION DU PROJET.....	2
B. PRÉSENTATION DE L'ÉQUIPE.....	3
C. OBJECTIF DU PROJET.....	7
C.1. OBJECTIF GLOBAL.....	7
C.2. OBJECTIFS TECHNIQUES PRINCIPAUX.....	8
C.3. OBJECTIFS TECHNIQUES SECONDAIRES.....	9
D. L'ARCHITECTURE GLOBALE.....	10
D.1. MICROCONTRÔLEUR.....	10
D.2. DÉTECTION DES MOUVEMENTS.....	11
D.3. COMMUNICATION.....	11
D.4. GESTION DES DONNÉES.....	12
<i>D.4.1. Le choix de la taille mémoire.</i>	14
D.5. ECRAN.....	14
D.6. INTERFACE GRAPHIQUE.....	15
D.7. DÉVELOPPEMENT DES SCHÉMAS ÉLECTRONIQUES POUR LE MONTAGE.....	16
D.8. DÉVELOPPEMENT DES CODES ARDUINO.....	18
<i>D.8.1. Analyse et structuration d'un code Arduino.</i>	18
<i>D.8.2. Limite technologique à prendre en compte.</i>	19
<i>D.8.3. Test des composants séparément.</i>	19
<i>D.8.4. Elaboration d'un schéma fonctionnel du programme.</i>	20
<i>D.8.5. Essais et amélioration.</i>	21
D.9. GESTION ÉNERGÉTIQUE.....	21
D.10. ERGONOMIE ET DESIGN.....	23
D.11. APPLICATION WINDOWS.....	24
E. DÉROULEMENT DU PROJET.....	26
E.1. PLANIFICATION DU PROJET.....	26
E.2. RÉPARTITION DES TÂCHES.....	26
E.3. COORDINATION DES TÂCHES.....	28
E.4. COMPOSANTS.....	30
E.5. PROBLÈMES RENCONTRÉS.....	31
F. RÉSULTATS.....	33
F.1. TESTS DE PERFORMANCE.....	33
F.2. RÉSULTATS SUR L'INTERFACE UTILISATEUR.....	34
F.3. APPLICATION.....	34
F.4. CONCLUSIONS ET PERSPECTIVES.....	36
F.5. PRÉSENTATION DU PROTOTYPE FONCTIONNEL.....	37
G. MODE D'EMPLOI.....	40
H. RETOUR D'EXPÉRIENCE.....	48
I. WEBOGRAPHIE.....	52
J. ANNEXES.....	56

A. Présentation du projet

En France, environ 140 000 personnes sont victimes d'un accident vasculaire cérébral chaque année, selon le site Ameli.fr [1]. Dans 85% des cas, l'AVC est dit « ischémique ». C'est-à-dire qu'un vaisseau sanguin se bouche dans le cerveau, le plus souvent à cause d'un caillot, ce qui bloque l'apport en oxygène et engendre des lésions dans la zone non irriguée. Les 15 % des AVC restants sont « hémorragiques » et résultent de la rupture d'une artère cérébrale.

On estime qu'environ 800 000 des victimes survivent, mais que 40% gardent des séquelles plus ou moins importantes. Les séquelles d'un accident vasculaire cérébral sont multiples : séquelles motrices, trouble des sens et de la compréhension, séquelles psychologiques, perte d'autonomie.

Parmi les séquelles motrices les plus fréquentes, on peut nommer [2] :

- Une paralysie ou une faiblesse d'un côté du corps, d'un bras...
- Une paralysie faciale
- Des tremblements et des mouvements anormaux

À la suite d'un AVC, s'entame, au plus vite, une longue phase de rééducation. Elle est en général motrice et orthophonique. Elle débute à l'hôpital puis se poursuit à domicile ou en centre spécialisé, selon les cas. La rééducation permet d'éviter l'apparition de complications et assure une récupération maximale des différentes fonctions : marche, langage, usage du bras et de la main etc. En effet, sans entraînement, un raidissement progressif des membres paralysés peut survenir et aggraver le handicap.

Après un retour au domicile, un suivi médical à vie est indispensable tel que des consultations avec le médecin traitant et des bilans avec d'autres professionnels de la santé. La victime de l'AVC doit également continuer la rééducation de son bras dans la vie de tous les jours. Ces personnes ont besoin d'un dispositif leur permettant de vérifier facilement si elles bougent assez leur membre atteint. C'est ainsi qu'en tant que futur ingénieur, nous avons pour objectif de créer un dispositif médical numérique de monitoring en post-AVC pour simplifier la phase de rééducation à domicile, nommé l'« ApplaWatch ».

B. Présentation de l'équipe

Notre équipe est composée de personnalités variées et complémentaires, issues de parcours divers comme le PEIP, le BUT ou encore la prépa. Afin de mettre cette richesse en avant, nous avons choisi de nous présenter individuellement.

Aurélien GOUMAIN



Bonjour, je m'appelle Aurélien et je suis bien évidemment en 3ème année du cycle ingénieur à Polytech Montpellier, spécialisé en Microélectronique et Automatique (MEA). Passionné par l'espace, l'aéronautique et les sciences en général, la curiosité et la soif de connaissance m'animent au quotidien. Après mon baccalauréat spécialisé en mathématiques et en physique-chimie je me suis dirigé vers une CUPGE à l'université Paul Sabatier de Toulouse pendant deux ans, pour ensuite intégrer Polytech ! J'ai eu l'opportunité de travailler l'été dernier chez PINK MOBILITY une entreprise de scooters électriques où j'ai pu voir la gestion de projet à grande échelle. Fort de mon parcours académique et de mes expériences précédentes, j'ai acquis des connaissances solides en sciences, développé de bonnes méthodologies de travail et renforcé mes compétences en relations humaines qui me seront utiles pour ce projet.

Zyed KAMIL



Je m'appelle Zyed Kamil, j'ai 20 ans et je suis actuellement en 3ème année du cycle ingénieur dans la spécialité Microélectronique et Automatique au sein de Polytech Montpellier. J'ai intégré le cycle préparatoire en Parcours d'École d'ingénieurs à Polytech (PEIP) directement après avoir obtenu mon baccalauréat spécialité Mathématiques et Physique-Chimie. Durant ma PEIP, j'ai poursuivi de nombreux projets notamment un projet en MEA qui m'a permis de prendre goût à l'électronique et à l'automatique. De plus, depuis petit, je me suis grandement intéressé à tout ce qui touchait à l'électronique. Toutes les innovations telles que les premiers téléphones tactiles, les robots me faisaient énormément d'œil jusqu'au point où je cherchais comment tout cela fonctionnait. Mes points forts au cours de ce projet sont le design, la rédaction et l'organisation.

Benjamin SAIGNE

Je m'appelle Benjamin et je suis actuellement en 3ème année du cycle ingénieur à Polytech Montpellier, spécialisé en Microélectronique et Automatique (MEA). Passionné par l'électronique depuis la fin du collège, j'ai développé un grand intérêt pour la conception et la programmation de systèmes électroniques. Cette passion m'a conduit à choisir un baccalauréat spécialisé en mathématiques et physiques, avec l'option sciences de l'ingénieur. À la suite de mon bac, j'ai intégré le cycle préparatoire PEIP à Polytech Montpellier en vue d'intégrer la spécialité MEA mais également pour renforcer mes bases scientifiques. Durant ces années, j'ai eu l'opportunité de travailler sur de nombreux projets, tant en classe qu'en milieu professionnel lors de stage, touchant à la fois l'électronique et l'informatique. J'espère mettre à contribution mes compétences en Arduino.



Adam MABROUQUE



Je m'appelle Adam MABROUQUE, j'ai 19 ans et je suis actuellement en 3ème année du cycle ingénieur dans la spécialité Microélectronique et Automatique au sein de Polytech Montpellier. J'ai intégré le cycle préparatoire en Parcours d'École d'ingénieurs à Polytech (PEIP) directement à Tours après avoir obtenu le diplôme du baccalauréat avec comme spécialité Mathématiques et Physique-Chimie. Durant ma PEIP, j'ai pu travailler chez STMicroelectronics, ce qui m'a permis de prendre goût à la spécialité microélectronique enseignée ici même à Polytech Montpellier. J'ai ainsi pu développer quelques compétences en informatique, mais aussi en physique, et par conséquent en électronique. Cela me permet donc de comprendre et de participer à plusieurs tâches sur ce projet. Enfin, j'ai toujours aimé rédiger et cela fait de moi un atout.

Alexandre DE JESUS MARTINS

Je m'appelle Alexandre, j'ai 21 ans et je suis actuellement en 3^{ème} année de cycle ingénieur à Polytech Montpellier dans le département Microélectronique et Automatique (MEA). Avant cela, j'ai passé un BUT en Mesures Physique parcours Techniques d'Instrumentation au cours duquel je me suis rendu compte que je préférais l'électronique et l'informatique à la physique, ce qui a motivé ma décision de venir en MEA. Entre la 2^{ème} et 3^{ème} année de BUT j'ai eu la chance d'effectuer une alternance au sein de l'entreprise Sonaxis dans leur pôle électronique. Mes missions principales étaient d'améliorer un logiciel de pilotage d'instruments en programmant de nouvelles fonctionnalités, et de concevoir des circuits imprimés sur ordinateur. Mon point fort au sein du groupe est la programmation.



Pierre-Marius CAMASSE



Je m'appelle Pierre-Marius CAMASSE, mon intérêt pour les sciences m'a poussé à opter pour un bac technologique. Après l'obtention de mon diplôme, j'ai choisi d'intégrer une classe préparatoire. Ce choix m'a permis d'approfondir mes connaissances en mathématiques, en physique et sciences de l'ingénieur. J'ai décidé de m'orienter vers une école d'ingénieurs spécialisée dans le secteur de l'électronique pour allier mon intérêt pour les nouvelles technologies et mon envie de contribuer aux avancées dans le domaine de la médecine. Mon parcours d'études m'a permis de développer des compétences en informatique, d'acquérir des connaissances en électronique et de m'investir dans des activités de groupe pour mettre au point de nouveaux projets.

Abel DIGUET

Je m'appelle Abel Diguet, j'ai toujours été intéressé par le domaine des sciences, c'est pour cela qu'après mon baccalauréat spécialités physique-chimie / Mathématiques je me suis orienté vers une CPGE, d'abord en MPSI puis en MP afin d'intégrer Polytech' Montpellier dans la spécialité MEA, qui était celle pour laquelle j'avais le plus d'intérêt. Pour ce qui est des compétences, mes deux années de prépa m'ont permis d'avoir un niveau équilibré dans les compétences requises pour ce projet et une certaine rigueur dans mon travail, même si durant ma CPGE je n'ai que très peu eu l'occasion de travailler en groupe, je m'adapte assez bien et suis à l'écoute de mes camarades.



C. Objectif du projet

C.1. Objectif global

L'objectif de notre projet est de concevoir un dispositif médical numérique de monitoring en post-AVC pour aider le patient dans sa rééducation ainsi que le médecin dans le suivi médical.

Nous avons choisi dans le cadre de ce projet de nous concentrer uniquement sur un dispositif adapté aux bras. En revanche, dans le cadre de projets ultérieurs, le dispositif pourrait être adapté à d'autres membres du corps.

Ainsi, nous avons projeté de mettre en place un dispositif composé d'un bracelet associé à une montre qui sera le cerveau du système. Chacun des deux systèmes mesurerait les mouvements du bras qui lui est associé, puis les données des deux systèmes seraient comparées.

La personne n'aurait plus qu'à regarder l'écran de la montre pour savoir si elle bouge assez ou non, son bras atteint.

Le schéma ci-dessous (schéma 1) récapitule cet objectif global :

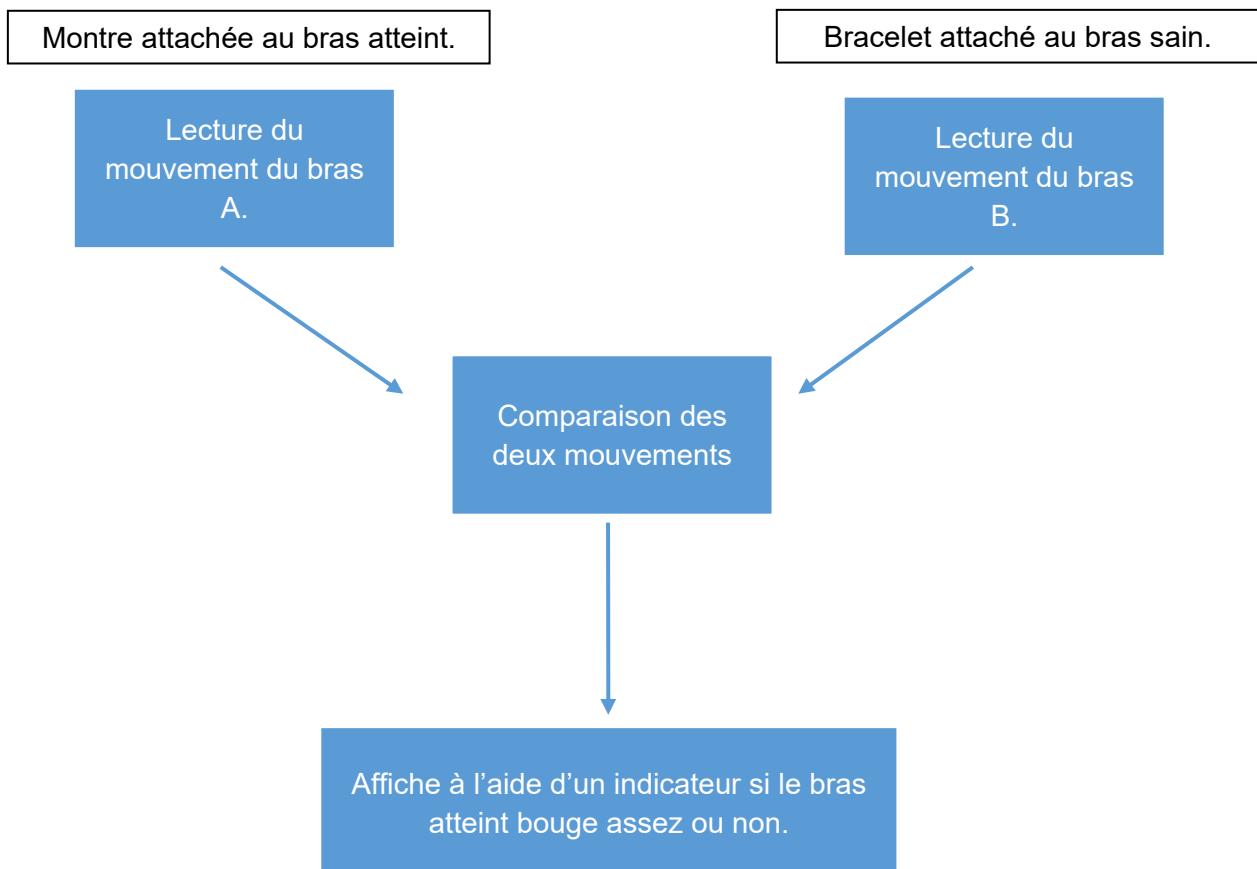


Schéma 1 : Schéma récapitulatif de l'objectif global

D'un point de vue un peu plus technique nous avons imaginé le dispositif suivant (schémas 2 et 3) :

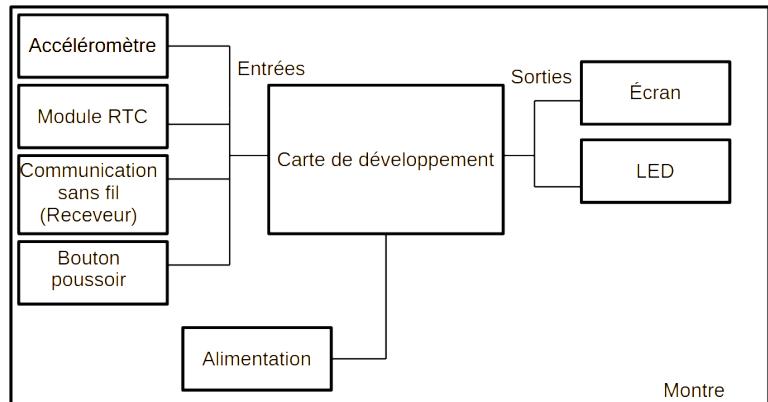


Schéma 2 : Schéma technique de la montre

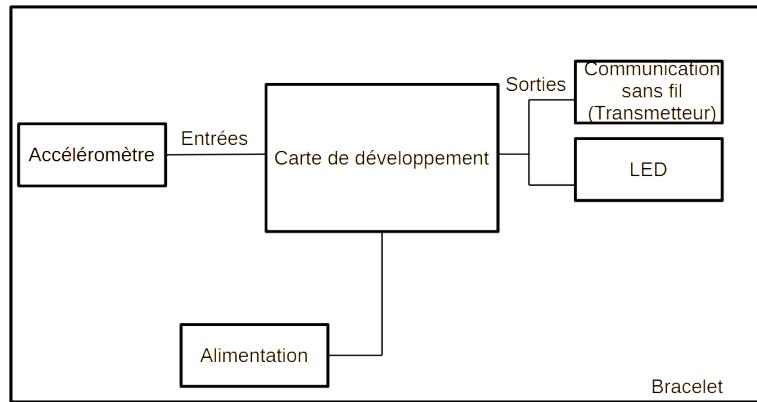


Schéma 3 : Schéma technique du bracelet

C.2. Objectifs techniques principaux

Pour y parvenir, nous avons donc défini les objectifs techniques principaux suivants :

- 1) Mesure des mouvements : Intégrer un dispositif de détection précis et performant des mouvements.
- 2) Communication sans fil : Concevoir et mettre en place un système de transmission des données, fiable et sans fil, entre la montre et le bracelet.
- 3) Interface « virtuelle » avec l'utilisateur : Concevoir une interface graphique permettant à l'utilisateur de facilement évaluer sa rééducation. L'interface doit être adaptée au public concerné.
- 4) Interface « physique » avec l'utilisateur : Concevoir un système physique permettant à l'utilisateur de mettre en veille et/ou d'allumer la montre.
- 5) Gestion énergétique : Etude d'une alimentation électrique adaptée.
- 6) Développement d'une application : Conception d'une application accessible permettant le transfert des données du dispositif vers le médecin.
- 7) Impression d'un boîtier en 3D : Conception et modélisation de deux boîtiers 3D. Un premier boîtier pour la montre et un second pour le bracelet.

C.3. Objectifs techniques secondaires

Nous avons également défini des objectifs secondaires :

- 1) Alimentation : Durée de vie de la batterie d'au moins 2 ans, rechargeable tous les 3 jours.
- 2) Résistance aux chocs et à la poussière.
- 3) Miniaturisation du dispositif avec un poids inférieur à 300g.
- 4) Utilisations de matériaux variés, non allergisants, et adaptés à une utilisation quotidienne et continue.

Pour remplir les objectifs cités précédemment, il faut respecter des enjeux économiques et les délais imposés. En effet, le prix ne doit pas dépasser une certaine somme, un prix faible est un synonyme de grand public. De plus le projet débute le mercredi 11 septembre 2024 et se termine le mercredi 22 janvier 2025. Afin de terminer ce projet ambitieux à temps, il est important de correctement le gérer et de le tenir à jour grâce aux outils de gestion. Nous pourrons voir en détail ces différents outils, qui nous ont aidé, dans une partie ultérieure.

D. L'architecture globale

D.1. Microcontrôleur

Il existe aujourd’hui une multitude de microcontrôleurs disponibles sur le marché. Les familles les plus connues sont les Arduino, les ESP32, les STM32 et les Raspberry Pi.

Les microcontrôleurs qui constitueront notre dispositif doivent répondre à plusieurs critères :

- Avoir une taille petite de l’ordre de 4cm de longueur et de 2cm de largeur, pour nous permettre de faire un prototype facilement tout en prenant le moins de place possible.
- Une consommation de l’ordre des 100 mA. A ce stade du projet, cette valeur est un ordre d’idée fondée sur aucun calcul. En revanche, cette valeur sera déterminée et affinée lors de l’étude énergétique.
- Un IDE (Environnement de développement) accessible.

Nous avons en premier lieu étudié la famille des ESP32 [3], malheureusement ceux fournis par le bâtiment 14 sont équipés de modules wifi intégrés. N’ayant pas besoin de wifi dans notre dispositif, nous avons choisi de ne pas utiliser ce microcontrôleur pour ne pas ajouter de modules inutiles qui pourraient consommer beaucoup de courant inutilement.

Puis nous avons étudié la famille des Raspberry Pi [4], les modèles tels que le Rasberry Pi 3,4 et 5 sont des modèles offrant un trop grand nombre de fonctionnalités inutiles à notre projet (port Ethernet, port USB, connexion SSD etc) mais la famille Rasberry Pi propose également des modèles plus simples et compacts tel que le Rasberry Pi pico 2.

De même la famille des STM32[5] propose des modèles trop évolués pour notre dispositif tel que le Nucleo-64 ou le Nucleo-144 mais aussi des modèles plus compacts tels que le Nucleo-32.

Nous avons fait le choix de nous orienter vers la famille des Arduino. En particulier nous avons choisi l’Arduino Nano Every [6] [7] (une évolution de l’Arduino Nano plus stable et puissante) qui propose des fonctionnalités similaires à ses concurrents mais un IDE plus accessible et de nombreux tutoriels en ligne. En voici une image (Fig. 1) ci-dessous.



Figure 2: Illustration de l'Arduino Nano Every

D.2. Détection des mouvements

Pour détecter le mouvement des bras, la méthode la plus simple est de mesurer son accélération selon les 3 axes (x, y, z). Un module nommé « accéléromètre » existe sur le marché et permet de mesurer l'intensité de pesanteur. L'intensité de pesanteur (notée « g ») est la force par unité de masse subie par un objet soumis à une accélération matérielle. Elle est directement proportionnelle à l'accélération d'un objet et sa valeur est de 9.81 m. s^{-2} .

Il existe de nombreux accéléromètres : à 3 axes, à 6 axes, capables de mesurer différentes accélérations : $\pm 2 \text{ g}$ $\pm 4 \text{ g}$ $\pm 8 \text{ g}$ voire $\pm 16 \text{ g}$.

Les accéléromètres à 6 axes intègrent des gyroscopes qui ne seront pas nécessaires dans la précision des mouvements pour notre dispositif.

D'après M.Facerias, un bras peut subir une accélération de près de 6g. Nous avons donc fait le choix d'intégrer un accéléromètre à 3 axes ayant une plage de mesure allant de $\pm 2 \text{ g}$ à $\pm 8 \text{ g}$.

L'accéléromètre MMA8451 [8] [9] dont les caractéristiques sont détaillées ci-dessous (Tab. 1) correspond à nos critères. De plus, sa consommation est très faible et sa précision (14 bits) est suffisante. Voici également une image de cet accéléromètre (Fig. 2).

Caractéristiques	Détails
Précision	14 bits
Plage de valeur	$\pm 2\text{g}$ / $\pm 4\text{g}$ / $\pm 8\text{g}$
Tension	5V
Courant	165 μA
Adresse	0x1D

Tableau 1 : Caractéristiques de l'accéléromètre MMA8451

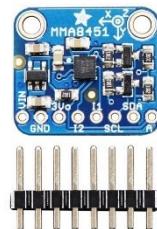


Figure 3: Image du MMA8451

D.3. Communication

Le réseau cellulaire, le Bluetooth, le wifi ou encore le protocole LoRaWan sont différents moyens de communication sans fil. Les deux systèmes composant le dispositif (la montre et le bracelet) doivent communiquer entre eux à l'aide de l'un de ses modes de communication. Ceci permettra l'échange des données entre le cerveau du système et le bracelet.

Nous avons tout d'abord émis l'hypothèse d'intégrer une connexion au réseau cellulaire, cette solution nécessite un abonnement chez un opérateur. Elle est donc trop coûteuse. Ensuite nous avons réfléchi à intégrer le wifi dans le dispositif. Le wifi n'étant pas disponible dans la rue, cette solution n'est pas viable.

De même, le protocole LoRaWan [10] est principalement lié au déploiement d'un environnement IoT avec de multiples objets connectés.

Le Bluetooth est une technologie radio sans fil à courte portée sur la fréquence des 2,4 GHz. Il existe sur le marché de l'électronique deux types de modules utilisant les fréquences du Bluetooth :

- Les modules tels que le HC-05 ou le HC-06 [11] sont capables d'être appairés avec un appareil IOS ou Android par exemple.
- Les modules équipés de la puce nRF24 [12]. La puce nRF24 est un circuit intégré de chez Nordic Semiconductor [13]. Ce type de module permet uniquement une communication entre des modules équipés de la puce nRF24. Son avantage est qu'il se pilote très facilement.

Nous avons choisi la seconde solution car, en effet, nous n'avons pas besoin d'appairer le dispositif à un quelconque appareil. Pour cela nous avons choisi le module NRF24L01+ de Sparkfun [14]. Ce module d'émission et de réception radio opère dans la bande 2,400 à 2,4835 GHz grâce à une antenne intégrée.

Il est facile à prendre en main grâce à sa programmation via le port SPI. En voici une illustration ci-contre (Fig. 3).



Figure 4: Illustration du NRF24L01+

Les caractéristiques du NRF24L01 [14] sont recensées ci-dessous :

Tableau 2 : Caractéristiques du NRF24L01

Caractéristiques	Détails
Fréquence de transmission	2,400 à 2,4835 GHz
Vitesse de transmission des données	250 kbps / 1 Mbps / 2 Mbps
Valeurs de rapport de puissance	-18 dBm / -12 dBm / -6 dBm / 0 dBm
Mémoire	32 octets max par message
Tension	5 V
Courant	11.3 mA

D.4. Gestion des données

Les données relevées par les 2 accéléromètres seront traitées par le cerveau du système (l'Arduino de la montre). Une fois le traitement fait, une seule valeur (sous forme d'entier) sera stockée dans la RAM : c'est l'écart de mouvement entre les deux accéléromètres.

Afin d'évaluer dans le futur les variations des mouvements entre les deux bras, il est nécessaire de stocker toutes les 30 minutes l'écart de mouvement mesuré entre les deux accéléromètres, ainsi que l'heure.

L'Arduino Nano Every est constitué de 3 types de mémoire :

- La mémoire flash (48 KB) [15] : C'est une mémoire rapide capable de réaliser près d'un million de cycles. Elle permet de stocker les programmes puisqu'elle perdure après la coupure de l'alimentation. Malheureusement, il est impossible de stocker des données dessus durant l'exécution du code.
- La mémoire SRAM (6 KB) [15] : Cette mémoire très rapide sert à stocker les données temporaires (les variables du programme). C'est une mémoire volatile qui oublie tout après une coupure de l'alimentation.
- La mémoire EEPROM (Electrically-Erasable Programmable Read-Only Memory) (256 bytes) [15] : Cette mémoire s'use très rapidement. Elle est capable de réaliser moins de 100 000 cycles d'écriture. Cette mémoire permet de stocker quelques données persistantes et importantes au programme.

Les mesures prélevées par l'accéléromètre doivent être stockées sur une mémoire extérieure à l'Arduino une fois la SRAM pleine. En effet, aucune des 3 mémoires composant l'Arduino n'est capable de réaliser cette tâche.

Les solutions de stockage disponibles aujourd'hui sont les SSD, les disques durs, les cartes SD/micro-SD, et les mémoires flash.

Les SSD, une solution de stockage numérique et non volatile des données, sont trop gourmands en énergie pour notre projet. Les disques durs sont, eux, trop lourds, trop imposants et également trop gourmands.

Donc nous avons, dans un premier temps, procédé à des tests avec une carte micro-SD.

La carte micro-SD fonctionne en utilisant la technologie de la mémoire flash. Elle permet de stocker les données de manière permanente comme indiqué auparavant.

Malheureusement, nous avons choisi de ne pas opter pour cette solution. En effet, ce type de mémoire est trop énergivore à l'écriture. Nous détaillerons les calculs et l'étude dans la partie gestion énergétique du projet.

Nous avons donc opté pour les mémoires flash. En voici une illustration ci-contre (Fig. 4).



D.4.1. Le choix de la taille mémoire

Les deux valeurs que nous stockons dans la mémoire sont des entiers non signés : 32 bits + 32 bits = 64 bits.

Dans une journée, il y a $24 \times 60 = 1440$ minutes.

Dans une semaine, il y a $1440 \times 7 = 10\ 080$ minutes.

Dans un mois, il y a $10\ 080 \times 4 = 40\ 320$ minutes.

Dans un an, il y a $40\ 320 \times 12 = 483\ 840$ minutes.

En 3 ans, il y a $483\ 840 \times 3 = 1\ 451\ 520$ minutes.

En enregistrant les 2 valeurs toutes les 15 minutes pendant 3 ans, on stockera
 $\frac{1\ 452\ 520}{15} \times 64 = 6\ 193\ 152$ bits de données.

Les mémoires flash que nous avons trouvées sont disponibles en plusieurs tailles : 2MB, 8 MB et 16 MB.

Conversion Mégabit en Bit (Tab. 3) :

1 Mégabit	1 000 000 bits
2 Mégabits	2 000 000 bits
8 Mégabits	8 000 000 bits
16 Mégabits	16 000 000 bits

Tableau 3 : Conversion de Mégabits en Bits

La mémoire adaptée à notre projet est une mémoire flash de 8 MB. Elle permettra de stocker des données pendant 3 ans assurant un long suivi après l'AVC.

D.5. Ecran

L'écran permet à l'utilisateur d'avoir un retour en temps réel de sa rééducation. Son choix est crucial dans l'accessibilité de la montre (taille, couleur, luminosité).

Il existe aujourd’hui plusieurs technologies d’afficheur : LCD, 7 segments (Led), Oled, E-Ink.

- L’afficheur LCD [16], à cristaux liquides utilise des pixels accompagnés d’un rétro-éclairage donnant une image standard.
- L’afficheur 7 segments [16] permet d’afficher des nombres. Or nous souhaitons afficher une barre de progression pour améliorer l’accessibilité. Cette technologie d’affichage ne convient donc pas.
- La technologie OLED [16] permet d’afficher une grande quantité d’informations aussi bien sous forme de texte, que d’images, ou éléments graphiques. Cette technologie existe en version monochrome ou multicolore permettant d’afficher des millions de couleurs. Cet écran permet donc une grande accessibilité pour notre projet.
- L’écran E-Ink [17] est un écran à encre électronique. Il consomme peu et l’affichage perdure même lorsque l’alimentation est débranchée. Malheureusement, sa fréquence de rafraîchissement (1 à 10s) ne permet pas un affichage en temps réel pour notre projet.

Nous avons donc choisi d’utiliser un écran OLED pour augmenter l’accessibilité du dispositif. C’est la solution la plus adaptée au public concerné (qui se trouve être pour les personnes âgées).

Notre choix s’est porté sur l’écran suivant (Tab. 4) :

 "Adafruit 1.3" and 1.54" 240x240 Wide Angle TFT LCD Displays"	<u>Caractéristiques :</u> <ul style="list-style-type: none">• Poids : 10g• Alimentation : 3,3 à 5 Vcc• Résolution : 240 x 240pixels• Interface SPI• Sur pastilles à souder• Sur connecteur EyeSPI• Dimensions : 43,7 x 41,8 x 5,5 mm
--	--

Tableau 4 : Caractéristiques de l'écran

D.6. Interface graphique

Pour offrir une expérience à l'utilisateur intuitive et facile à prendre en main, l'interface graphique sera un outil crucial et majeur de la conception du projet. Selon nous, sa conception doit être équilibrée autant esthétiquement que du point de vue fonctionnel. De plus, dans le cahier des charges, nous avons défini la performance suivante : "Interface utilisateur simple, intuitive, personnalisable et accessible".

Pour réaliser notre interface graphique nous avons imaginé et créé différentes maquettes et prototypes.

On s'est également posé la question de savoir à quoi l'interface graphique de la montre ressemblera. Après plusieurs prototypes, nous sommes arrivés à une interface 0.



Figure 5 : Interface Graphique de la montre

Pour arriver à cette interface, nous avons d'abord dû choisir un écran. Pour cela, nous en avons cherché un qui n'était ni trop grand ni trop petit et qui n'était pas tactile, car cela complexifie la création de l'interface. De plus, un écran tel que celui-ci serait trop petit pour un doigt humain (la précision serait moindre). Nous avons ainsi choisi un écran de 1,54 pouce et de 240 par 240 pixels.

Il fallait aussi penser à la manière dont l'utilisateur pourrait utiliser cette interface et cette montre, tel que le choix du nombre de boutons, leur position, et pendant combien de temps l'écran restera allumé. Nous avons donc statué sur le choix d'un seul bouton, qui est disponible dans le stock du bâtiment 14.

Nous avons également choisi de prendre 1 module RTC [18] qui permettra d'afficher l'heure sur l'écran car, avant tout, il s'agit principalement d'une montre.

D.7. Développement des schémas électriques pour le montage

Une fois la partie étude des différents modules réalisée, nous avons procédé à l'élaboration du câblage en vue de faciliter celui-ci lors de la conception.

Tout d'abord, nous avons représenté dans le tableau ci-dessous (Tab. 5), pour chaque composant, les différents pins auxquels ils doivent être branchés.

Pin	Vcc 5V	Gn d	2	3	5	6	8	9	10	1 1	12	1 3	SCL	SDA
Ecran Oled 1.54 pouces	x	x	x				x	x	x	x		x		
Module Bluetooth (NRF24L01)	x	x			x	x				x	x	x		
Module RTC	x	x											x	x
Bouton Poussoir	x	x		x										
Module Accéléromètre (MMA 8451)	x	x											x	x

Tableau 5 : Pins occupés sur l'Arduino

Puis nous avons dessiné un schéma du montage électrique final de la montre (Schéma 4) où les composants sont reliés en même temps à l'Arduino Nano Every.

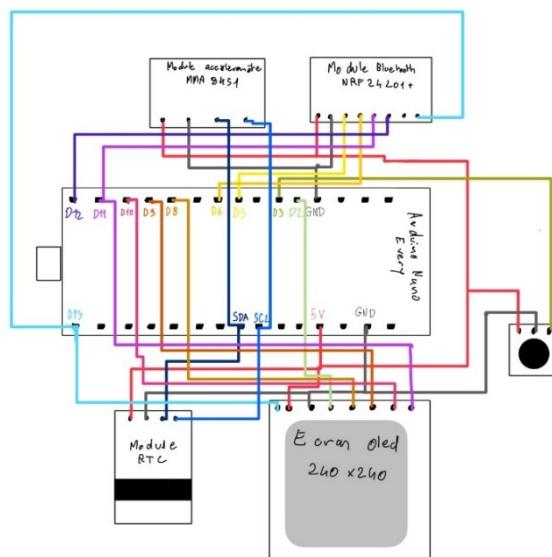


Schéma 4 : Schéma câblage montre

Puis nous avons fait de même avec le bracelet en représentant dans le tableau ci-dessous (Tab. 6), pour chaque composant, les différents pins auxquels ils doivent être branchés.

Pin	Vcc 5v	Gnd	MISO (D12)	MOSI (D11)	SCK (D13)	CSN (D6)	CE (D5)	SDA (A4)	SCL (A5)
Mma8451 (Accéléromètre)	X	X						X	X
NRF24L01+ (Module Bluetooth)	X	X	X	X	X	X	X		

Tableau 6 : Pins occupés sur l'Arduino

Enfin, nous avons dessiné un schéma du montage électronique final du bracelet (Schéma 5) où les composants sont reliés en même temps à l'Arduino Nano Every.

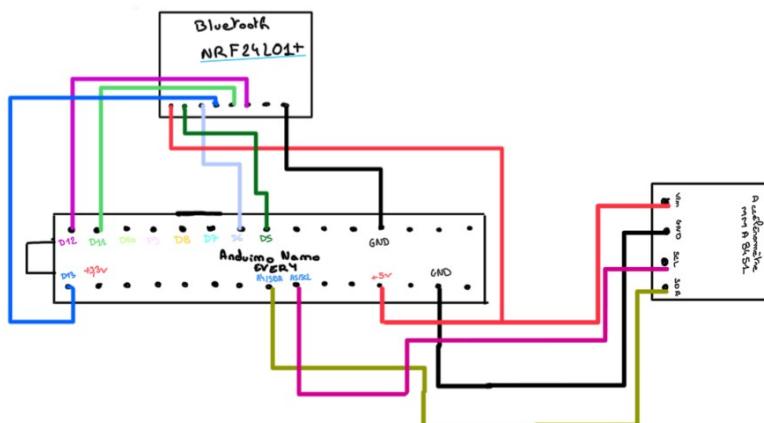


Schéma 5 : Schéma Câblage bracelet

D.8. Développement des codes Arduino

L'étape la plus cruciale du projet est le développement du code Arduino. Celui-ci doit être réalisé en plusieurs étapes :

- Analyse et structuration d'un code Arduino.
- Limites technologiques à prendre en compte.
- Test des composants séparément.
- Elaboration d'un schéma fonctionnel du programme.
- Programmation complète.
- Essais et amélioration.

D.8.1. Analyse et structuration d'un code Arduino

Un Arduino se programme en langage C++ grâce à l'IDE mis à la disposition des utilisateurs. Un certain nombre de librairies est également disponible au téléchargement, pour simplifier l'utilisation des différents modules.

Un programme Arduino se structure en deux grandes parties : le « void setup » et le « void loop ». Le « void setup » s'exécute une seule fois à chaque démarrage de l'Arduino. Tandis que le « void loop » s'exécute en boucle une fois le « void setup » fini.

D.8.2. Limite technologique à prendre en compte

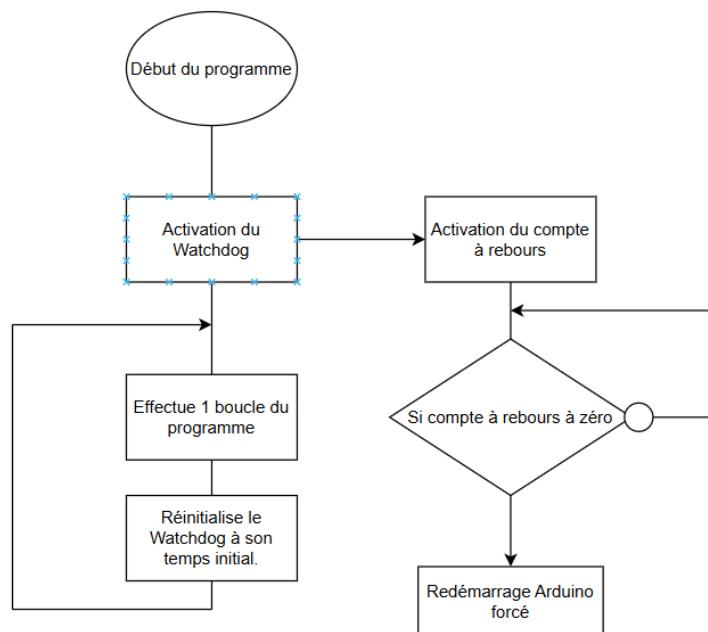
L'Arduino Nano Every possède une limite technologique à prendre en compte. Il est composé d'un seul cœur. Il n'effectue donc qu'une seule tâche à la fois. L'utilisation de « delay() » est à prohiber, en effet il impliquerait de nombreux problèmes :

- Ecran figé
- Perte de mesure
- Problème de réception des données

Au contraire, l'utilisation du Watchdog [19] est préférable. Un Watchdog est un dispositif intégré à la plupart des microcontrôleurs. Il permet d'effectuer des opérations particulières au bout d'un certain temps lorsqu'il est activé. On l'utilise en pratique pour :

- Réinitialiser le microcontrôleur en cas de plantage logiciel.
- Déclencher une interruption au bout d'un certain temps, pour effectuer des opérations particulières (mise en arrêt de moteur électrique en cas de problème, sauvegarde de données après un « bug » logiciel).

Dans notre cas, il permettra de réinitialiser l'Arduino en cas de plantage logiciel.



Voici un schéma (schéma 6) expliquant son implémentation :

Schéma 6 : Flowchart de l'utilisation du Watchdog

D.8.3. Test des composants séparément

Ensuite vient la phase de test des composants. Nous devons tester séparément le fonctionnement de chaque composant. On retrouve souvent sur le site de chaque module une librairie associée ou un exemple de code. On s'aidera de celui-ci pour tester nos composants et programmer l'ensemble du programme.

L'intérêt de cette manipulation est de constater :

- Les erreurs de conception, s'il y en a.
- Le bon fonctionnement seul des composants. En effet lorsque tous les composants seront branchés ensemble, ils pourraient rentrer en conflit et ne plus fonctionner.
- La validité du câblage.

D.8.4. Elaboration d'un schéma fonctionnel du programme

Le schéma suivant (schéma 7) résume ce que doit effectuer le programme de la montre.

On visualise les parties « void setup » et « void loop » présentées précédemment.

Il est similaire pour le bracelet sans la partie écran et module RTC.

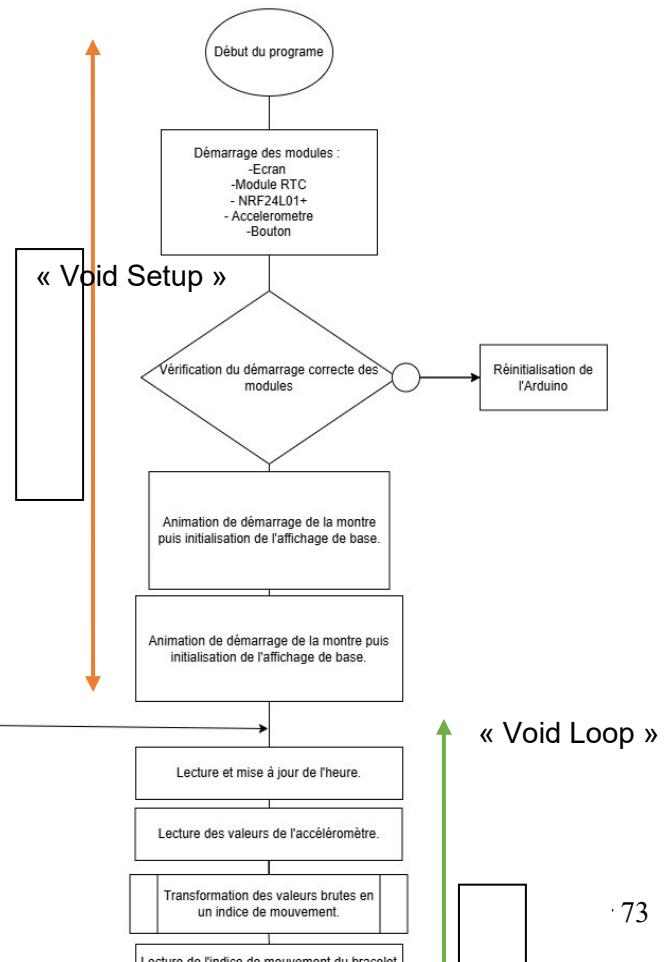


Schéma 7 : Schéma fonctionnel du programme Arduino

D.8.5. Essais et amélioration

Une fois l'ensemble du code développé, nous passerons à une série de tests que nous avons listée ci-dessous :

- Animation de l'écran
- Démarrage et initialisation de tous les composants.
- Affichage de l'interface graphique
- Envoi et réception de données sans obstacle intermédiaire.
- Envoi et réception de données avec un obstacle de type bras intermédiaire.
- Détection d'un mouvement nul.
- Détection des mouvements à basse vitesse.
- Détection des mouvements à grande vitesse.

Elle permettra d'évaluer la solidité de notre algorithme et de détecter les éventuels problèmes.

D.9. Gestion énergétique

Dimensionnement des batteries :

Pour notre projet, le choix des batteries est important. Dans le cahier des charges nous avons défini une performance liée à ce choix "La batterie doit avoir une durée de vie d'au moins deux ans, rechargeable tous les 3 jours".

Afin de dimensionner nos batteries, nous avons utilisé les fiches techniques pour estimer la consommation en ampères des différents éléments du système. Ensuite à l'aide

d'un ampèremètre, nous avons comparé les valeurs données par les documents et celles mesurées.

Nous avons supposé que l'utilisateur utiliserait la montre et le bracelet au maximum 12h par jour.

Dimensionnement de la batterie pour la montre (Tab. 7) :

Composants	Courant (issu des fiches techniques)	Courant mesuré	Temps utilisé par journée
Accéléromètre	156µA		12h
Arduino Nano	35mA		12h
Module Bluetooth récepteur	11,3mA		12h
Module RTC	1µA		12h
Module carte micro-SD	150mA		12h
Ecran oled	20mA		3h
Système complet		300mA	

Tableau 7 : caractéristiques des consommations pour la batterie de la montre

Pour une journée, la consommation maximale théorique de la montre en mA.h est donnée par :

$$\begin{aligned} Cons_{Max} &= \sum I \times t \\ Cons_{Max} &= (35 + 11,3 + 150 + 0,156 + 0,001) \times 12 + 20 \times 3 \\ Cons_{Max} &= 4398 \text{ mA.h} \end{aligned}$$

Dimensionnement de la batterie pour le bracelet (Tab. 8) :

Tableau 8 : caractéristiques des consommations pour la batterie de la montre

Composants	Courant (issu des fiches techniques)	Courant mesuré	Temps utilisé par journée
Accéléromètre	156µA		12h
Arduino Nano	35mA		12h
Module Bluetooth émetteur	12,3mA		12h
Système complet		45mA	

Pour une journée, la consommation maximale théorique du bracelet en mA.h est donnée par :

$$\begin{aligned}Cons_{Max} &= \sum I \times t \\Cons_{Max} &= (35 + 0,156 + 12,3) \times 12 \\Cons_{Max} &= 2549,5 \text{ mA.h}\end{aligned}$$

Il existe différentes technologies pour les batteries, chacune ayant ses propres caractéristiques. Les batteries les plus adaptées à notre projet sont les batteries lithium-ion (Li-ion), les batteries lithium-polymère (Li-Po) et les batteries nickel-hydrure (NiMH). Pour ce projet, nous avons besoin d'une batterie rechargeable, avec une bonne durée de vie et plutôt légère pour respecter notre cahier des charges. Nous garderons alors les batteries lithium-ion et les batteries lithium-polymères. De plus, nos cartes Arduino [20] demandent une plage de tension d'entrée de 7V à 21V.

Pour ce projet, nous n'avons pas de contrainte de budget (dans la limite du raisonnable), mais le coût de nos composants est un paramètre important. Nous avons donc choisi des batteries Li-Po qui sont rechargeables et nous utiliserons des régulateurs boosters qui permettront de délivrer une tension attendue par la carte Arduino.

Les valeurs théoriques et mesurées sont différentes car nous avons estimé la consommation maximale possible.

Nous avons alors choisi pour la montre la batterie « Accu Li-Po 3,7 Vcc 4000 mAh L805080 » qui nous paraît la plus adaptée pour notre système. Pour le bracelet, nous avons choisi la batterie « CELLEVIA BATTERIES LP674261 3,7V 2000mAh ». Ces batteries sont, selon nous, les plus propices pour notre projet.

D.10. Ergonomie et design

Pour la création de la montre et du bracelet, nous voulons mettre en place un circuit imprimé (PCB) [21] afin d'optimiser leur conception. Cette solution permet d'intégrer directement toutes les connexions sur une seule plateforme compacte. Cela réduira les risques de câbles détachés et les risques de mauvais contact. De plus, un PCB permettra une miniaturisation du dispositif et facilitera l'assemblage du système.

Concernant l'ergonomie et le design de la montre, un boîtier devra protéger les composants électroniques à l'intérieur de la montre et du bracelet. De plus, il doit garantir une résistance aux chocs.

Pour créer ces boîtiers, nous avons utilisé un logiciel de CAO (Conception Assistée par Ordinateur). Nous avons fait nos premiers prototypes sur l'application Onshape [22] avant de passer à Solidworks [23] bénéficier d'outils plus avancés et de fonctionnalités supplémentaires dans la création des modèles 3D.

Pour le bracelet :

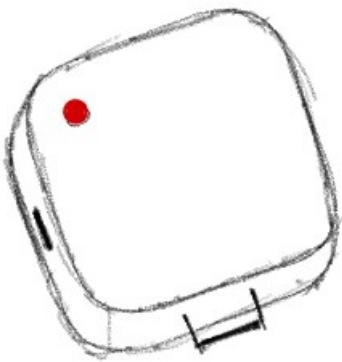


Figure 6 : Prototype du boîtier du bracelet

Pour la montre :

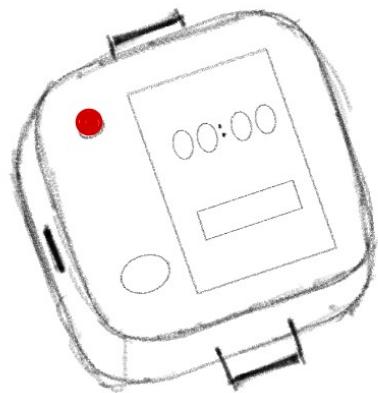


Figure 7 : Prototype du boîtier de la montre

Nous avons ajouté des attaches sur les côtés afin de pouvoir attacher le bracelet et la montre au poignet. Un emplacement pour la LED.

Ces deux figures ne sont que représentatives. C'est-à-dire que ces prototypes changeront, en fonction de la création des PCB [22], et de comment les composants se retrouveront les uns sur les autres. Il faut également penser aux boutons, ainsi créer des trous aux bons endroits, selon leur disposition. Les matériaux sélectionnés doivent être légers mais robustes.

On peut considérer que cette partie est la dernière dans la création de notre projet. Il faut en effet avoir finalisé plus ou moins toutes les autres parties avant d'imprimer ces boîtiers.

D.11. Application Windows

L'application a pour objectif principal de permettre la communication entre la montre et l'ordinateur. Il faut pouvoir récupérer les données de la montre à partir du port série et stocker celles-ci sur l'ordinateur.

Dans un second temps, nous souhaitons créer une interface graphique permettant d'afficher une courbe avec les données, afin de pouvoir avoir un retour sur l'amélioration sur le long terme de l'usage du membre atteint.

Pour cela, nous utilisons le langage python. Python [24] est un langage de programmation orienté objet. Il est simple mais puissant ce qui en fait un langage polyvalent. C'est un langage interprété, ce qui veut dire que le langage est directement exécuté par ce qu'on appelle un interpréteur sans passer par une étape de compilation. Ça en fait donc un langage idéal pour du prototypage ou de la preuve de concept comme notre montre. Il existe cependant des façons de créer un fichier exe auto-suffisant qui emporte avec lui l'interpréteur.

Python est un langage très utilisé dans le monde professionnel et a donc une communauté active. Il possède de nombreuses librairies standards et modules tiers qui apportent de nombreux outils pour résoudre des problèmes variés. Pour notre application, nous utilisions les librairies suivantes :

- PySerial [25] pour la communication série avec la montre
- Tkinter [26] pour créer l'interface graphique
- Matplotlib [27] pour créer les courbes de données
- Numpy [28] pour faciliter l'intégration de Matplotlib.

La librairie PySerial permet d'établir une connexion série et de recevoir ou d'envoyer des données via cette connexion. C'est donc la librairie qui permet d'assurer l'objectif principal de l'application qui est de récupérer les données de la montre.

La librairie tkinter se base sur l'utilisation de « widget ». Un widget est le nom général pour tout composant d'interface graphique que l'on va mettre dans notre application. Ce sont les briques avec lesquelles on construit l'application. Cela peut être un bouton, un texte, une image ou un cadre. Un cadre (ou Frame en anglais) est un élément un peu plus abstrait car il n'est pas toujours visible. Il s'agit pourtant d'un des widgets les plus importants et les plus utilisés car il permet de sectionner et structurer l'application en regroupant les autres widgets dans des groupes logiques.

La librairie Matplotlib permet de créer des graphiques de données. C'est la librairie que l'on utilise pour générer les courbes de données que l'on affiche sur l'application. On utilise également dans cette librairie la classe « FigureCanvasTkAgg » qui permet d'intégrer les courbes créées dans l'interface Tkinter. Pour notre application, la librairie Numpy est seulement utilisée pour transformer les données afin de faciliter la création des courbes avec Matplotlib.

E. Déroulement du projet

Ce projet a été développé en plusieurs étapes afin de répondre aux besoins. Nous avons dû nous organiser afin de permettre une bonne communication, productivité, mais surtout afin de garantir l'efficacité dans la réalisation du projet.

E.1. Planification du projet

Pour réaliser ce projet, plusieurs méthodes ont été mises en place pour garantir une progression efficace. L'outil de planification comme le diagramme de Gantt est essentiel. Ce dernier permet d'avoir une vue d'ensemble du projet, d'organiser les étapes clés et de définir des jalons importants. Pour créer ce diagramme, le logiciel utilisé est Beesbusy [29]. Nous avons pu créer de grands thèmes, qui présentent chacun des sous-tâches. Ainsi, nous y retrouvons sur la gauche les étapes clés de notre projet. Néanmoins, il est bien difficile de s'organiser dans un groupe de cette ampleur avec seulement quelques grandes tâches. C'est pourquoi nous avons tenté de déléguer au maximum et de séparer en plusieurs sous-tâches toutes les grandes parties. En voici une image (Fig. 8).

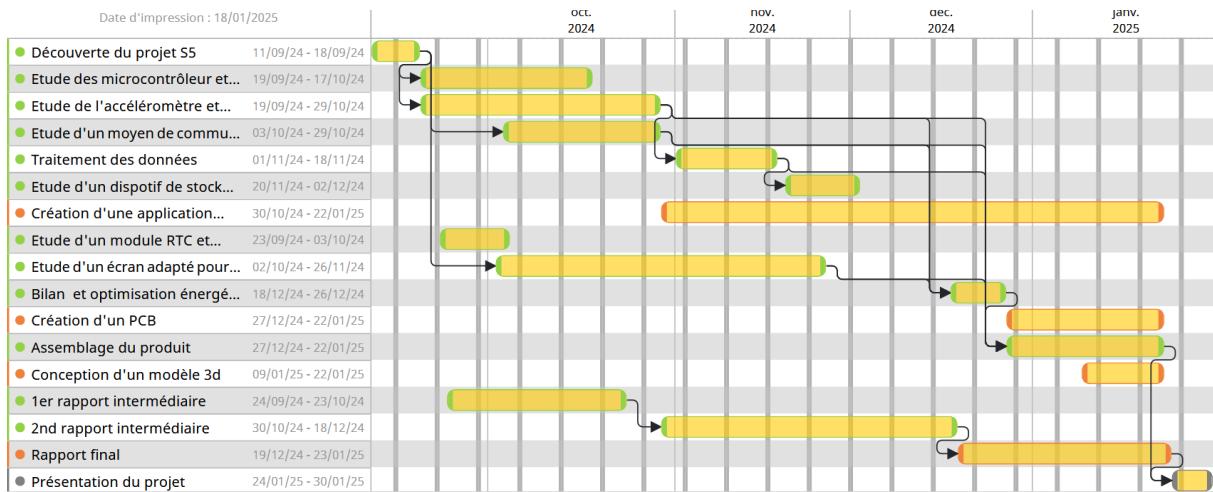


Figure 8 : Diagramme de Gantt

E.2. Répartition des tâches

Nous avons ainsi opté pour séparer notre projet en 14 grandes parties, comprenant entre 1 et 5 sous-tâches. Voici donc les grandes parties sur lesquelles nous nous sommes appuyés pour parvenir à créer notre projet :

- Découverte du projet
- Etude des microcontrôleurs et choix adapté.
- Etude d'un moyen de mesure des mouvements et prise en main de celui-ci.
- Etude d'un moyen de communication sans fil et mise en place de celui-ci.
- Traitement des données.
- Etude d'un moyen permettant de stocker les données.
- Création d'une application pour la transmission des données.
- Etude d'un écran adapté pour la montre et mise en place de celui-ci.
- Energie et optimisation énergétique.
- Création d'un PCB.
- Assemblage du projet.
- Conception d'un modèle 3D.
- Rédaction du rapport de projet
- Présentation du projet

Chaque sous-partie peut être définie durant une certaine période. Par conséquent, nous avons pu séparer les grands thèmes sur plusieurs périodes explicites. Celles-ci nous permettent d'avoir un ordre de grandeur de la durée d'une sous-tâche. Nous pouvons donc nous organiser, en fonction de l'avancement de chacun, et savoir s'il faut ou non accélérer grandement (dans l'idée de tous être coordonnés). En effet, il est très compliqué de

commencer certaines parties avant la fin de certaines autres. Pour donner un exemple assez trivial, on ne peut pas travailler complètement sur la l'assemblage du projet avant d'avoir fini toutes les parties précédentes qui permettent cet assemblage.

De plus, pour une meilleure organisation, il y a la possibilité d'associer une ou plusieurs personnes à chaque sous-tâche. Cette fonctionnalité n'est pas visible sur l'image ci-dessus, mais nous nous en sommes servis et cela nous a bien aidé afin que tout le monde ait un nombre et une quantité de tâches à peu près égaux. Cela améliore aussi la coordination entre certains petits groupes et entre les différentes grandes parties.

Le diagramme de Gantt détaille les tâches accomplies par chaque membre du groupe :

- Benjamin SAIGNE : Etude des accéléromètres et programmation de ces derniers. Choix d'un grand nombre de composants. Détermination d'une méthode pour analyser les données. Création des schémas électriques du projet. Assemblage des composants.
- Abel DIGUET : Etudes théoriques sur le stockage des données et l'identification des données à transmettre. Etude du mode stockage et programmation de celui-ci. Création de l'interface graphique de la montre.
- Zyed KAMIL : Etude des modules de communication sans fil et test de connexion entre eux. Création de l'interface graphique de la montre. Intégration du module RTC. Conception du modèle 3D du bracelet.
- Aurélien GOUMAIN : Etude des accéléromètres et de la détection des mouvements. Réalisation des soudures. Création de l'interface graphique de l'application. Conception d'un PCB.
- Pierre-Marius CAMASSE : Etude des différents composants pour l'affichage et la programmation de l'interface graphique. Réalisation du bilan énergétique et choix des batteries. Conception du modèle 3D de la montre et du bracelet.
- Adam MABROUQUE : Etude des modules de communication sans fil, programmation et configuration de la connexion sans fil entre les modules de connexion. Développement de l'application. Réalisation des soudures.

- Alexandre DE JESUS MARTINS : Etudes théoriques sur le stockage des données et l'identification des données à transmettre. Création de l'application qui permettra de transférer les données au médecin.

La répartition des tâches a été réalisée en tenant compte des avis et des préférences de chaque membre de l'équipe. Certaines personnes ayant plus d'expérience grâce à des projets réalisés en amont ont notamment aidé aux choix des différents composants.

E.3. Coordination des tâches

Afin d'assurer la coordination du projet, nous avons dû utiliser différents outils afin de stocker les travaux et plusieurs méthodes ont été mises en place pour faciliter l'organisation et la communication.

- **Réunion**

Tout d'abord, il est essentiel d'organiser des réunions au sein du groupe pour que chacun ait une vision claire de ses tâches à accomplir durant la séance. C'est pour cela qu'à chaque début et fin de séance, nous organisons des réunions. La réunion de début de séance permet à chaque membre du groupe d'avoir une idée précise de sa tâche à effectuer cette semaine-ci. De plus, afin d'optimiser les séances, chaque membre du groupe se verra attribuer le rôle d'animateur, celui du secrétaire ou encore celui du scribe. Enfin, la réunion de fin de séance permet de faire un récapitulatif de tout en rédigeant l'ordre du jour de la prochaine.

- **Communication**

Ensuite, afin de garantir une bonne communication au sein de l'équipe, dès la première séance, nous avons créé un groupe WhatsApp [30] nous permettant d'être connectés 24h/24 et 7j/7. Ce groupe nous permet de communiquer sur différents sujets afin que tout le monde soit au courant des nouvelles informations, telles que l'arrivée des composants après une commande par exemple.

- **Gestion des documents**

Pour gérer l'ensemble des documents de notre projet, nous avons choisi l'application Gitlab [31]. C'est un outil de gestion de projet utilisé surtout dans le développement d'applications. Nous l'utilisons dans notre cas pour bien organiser les tâches que nous devrons réaliser au cours de ce projet. Voici un tableau (Tab. 9) récapitulant les différentes branches du projet.

Branche	Utilité
Rapport Projet	L'ensemble des rapports du projet sont disponibles dans cette branche.
Arduino.ino	L'ensemble des codes Arduino utilisés dans le projet sont disponibles dans cette branche.
Flowchart	L'ensemble des Flowcharts utilisés pour concevoir les codes Arduino sont détaillés dans cette branche.
Schéma	L'ensemble des schémas électroniques réalisés sur Freeform sont disponibles dans cette branche.
Réunion	L'ensemble des comptes rendus de réunion réalisés à chaque séance sont disponibles dans cette branche.
Commande	L'ensemble des achats réalisés tout au long du projet sont disponibles dans cette branche.
Application	Le programme de l'application permettant l'envoi des données par ordinateur est disponible dans cette branche.
Lien	Les liens utilisés sont disponibles dans cette branche.
Organisation	Tous les documents utiles à l' organisation sont disponibles dans cette branche.

Tableau 9 : Les différentes branches du projet sous Gitlab

Gitlab s'organise en branches. La branche « main » est le répertoire principal et par défaut du projet, nous y stockons l'ensemble des consignes du projet et le mode d'emploi d'utilisation du Gitlab. Pour stocker et trier de façon efficace les documents du projet, nous avons créé une branche pour chaque catégorie de documents.

Gitlab permet d'ajouter facilement n'importe quel type de fichier à une branche, que ce soit un fichier Word, une image en png, etc. Pour un meilleur suivi des documents, nous avons choisi d'indiquer pour chaque fichier un nom qui le caractérise facilement et un numéro de version lors des mises à jour.

De plus, chaque branche contient un fichier « README.md » précisant les règles et informations complémentaires de la branche. Ce fichier permet à un membre du groupe qui n'aurait pas travaillé sur cette partie-là de comprendre et de modifier facilement un document.

Par exemple, on retrouve pour la branche « Flowchart » le type de formats autorisés et l'application sur laquelle les flowcharts ont été réalisés.

Gitlab est donc un outil de gestion puissant. Il permet à l'équipe de rester organisée et de se partager facilement l'ensemble des fichiers. Il est ainsi essentiel dans le développement du projet.

Voici donc la branche spécifique concernant les flowcharts, et expliquant à chaque personne du groupe comment fonctionne cette branche (Fig. 9).

README.md
Flowcharts
Les flowcharts sont la première trace de réflexion des différents programmes informatiques. Ils présentent une version simplifiée des codes Arduino.
Vous trouverez les flowcharts sous deux formats:
1. au format png

Figure 9 : Branche flowcharts explications

E.4. Composants

Gotronic, RS [32], Farnell [33], Conrad, Lextronic, Radiospares, etc., sont des fournisseurs conseillés par les tuteurs du projet. Ainsi nous nous sommes rendus sur leur site pour découvrir ce qui était proposé et choisir ce qui conviendrait le mieux pour notre projet. Nous avons comparé l'accessibilité des documentations en ligne, la consommation énergétique, la taille des composants, leurs délais de livraison et leur prix pour faire le meilleur choix pour notre projet.

Voici la fiche des différents composants, comportant leurs références, leurs prix et la quantité (Tab. 10) :

Référence	Fournisseur ou stock bât. 14	Désignation	Quantité	Prix unitaire TTC	Prix total TTC
9054655	RS	Accéléromètre Adafruit 3 axes i2c	2	6.53	13.06
35403	Gotronic	Module RTC DFROBOT	1	9.70	9.70
32123	Gotronic	Module transceiver nRF24L01+	2	Inconnu	Inconnu
38114	Gotronic	Ecran Oled 1.54 pouces IPS	1	21.75	21.75
192-7586	RS	Arduino Nano Every	2	14.10	28.2
09949	Gotronic	Accu LiPo 2000 mAh	1	18.90	18.90
09948	Gotronic	Accu LiPo 4000 mAh	1	23.50	23.50
38569	Gotronic	Chargeur LiPo USB-C	2	7.50	15.00
32800	Gotronic	Booster 4 à 25 V	2	17.90	35.80
	Bat 14	Bouton poussoir	1	0	0
	Bat 14	Résistance	1	0	0
				Total :	165.91

Tableau 10 : Tableau de commande des composants

E.5. Problèmes rencontrés

Au cours du projet, plusieurs problèmes ont été rencontrés, des problèmes techniques, de programmation, des oubli de commandes ou de transmissions de commandes, ce qui a parfois perturbé l'avancement du projet.

On peut ainsi lister les différents thèmes qui ont pu poser problème, ou même les contraintes et obligations qui nous ont plus ou moins ralenti.

Concernant :

- **Les batteries**

Il faut tout d'abord connaître le niveau de charge des batteries et transmettre cette information à l'utilisateur. Car en effet, sur l'interface de la montre, une icône permettant de savoir le niveau de charge est affiché ; et l'utilisateur pourra voir en temps réel durant une journée l'autonomie restante de sa montre. Le problème principal est donc de savoir comment mesurer cette charge, et programmer ceci avec l'affichage de l'écran.

- **Le stockage**

Etant donné qu'il y a eu des complications concernant les commandes, nous n'avons pas reçu les mémoires flashs qui permettent de stocker les données tout au long d'une journée. De plus, les cartes micro-SD misent à notre disposition consomment trop pour pouvoir être utilisées dans notre projet (cf. partie D.4.1).

- **Le développement de l'application**

Initialement, nous avions choisi d'utiliser Processing [34] pour créer l'application. Processing est un langage de programmation qui sert à créer des visuels ou interfaces graphiques. Il se base sur du C++ et ressemble beaucoup à Arduino dans sa structure : une fonction `setup()` appelée une fois à l'exécution du script et une fonction `draw()` appelée en boucle jusqu'à l'arrêt du script.

Nous avons passé plusieurs heures à apprendre comment utiliser Processing, à se familiariser avec et à commencer à implémenter certaines fonctionnalités de l'application avant de rencontrer un problème. Ce langage n'était pas adapté pour tracer des courbes de données, une des fonctionnalités de l'application. En effet, il n'y a aucune fonction déjà intégrée dans Processing pour tracer des données. Il est quand même possible de le faire, mais cela demande de tracer le moindre élément d'un graphique manuellement. Cela comprend chaque segment de courbes entre deux échantillons, les axes, chaque graduation. Cela aurait demandé une quantité de travail massive, que l'on a jugé trop importante pour une fonctionnalité secondaire. Cela aurait également ralenti notre application mais aussi aurait rendu les interactions avec la courbe (zoom, pan, etc.) quasiment impossible à implémenter.

Nous avons finalement opté pour le choix de changer de langage de programmation. Comme expliqué dans la partie D.11, avec les conseils de M. Facerias et M. Garcier, nous avons choisis d'utiliser Python avec la librairie tkinter pour l'interface graphique.

- **La communication au sein du groupe**

En effet, durant toute la création de notre projet, un grand problème a été la communication. Plus précisément, que ce soit au sein de notre groupe, ou même entre les autres groupes travaillant sur le même projet, nous avons eu des manques de communication. En effet, il aurait été sûrement plus pratique, et plus productif :

- De savoir ce qu'il reste à faire sur une mission d'un duo ou trio de personnes.
- Si ce binôme/trinôme rencontre des problèmes, et si ceux-ci sont solvables facilement ou non.
- De savoir que font les autres groupes de projet, comment ils fonctionnent, si tel ou tel moyen de procéder est mieux ou non...

Globalement, il aurait été mieux de plus parler aux autres. Cela nous aurait permis d'améliorer plusieurs points, de régler plusieurs problèmes, et même de rendre des comptes-rendus mieux rédiger et plus professionnels.

Finalement, nous avons eu différentes contraintes ou problèmes tout au long de la création de notre projet. Ces problèmes ont chacun été plus ou moins réglés. En effet, selon les difficultés matérielles ou autres, nous avons pu régler et/ou améliorer certains points comme, par exemple, le côté application. En effet, après s'être lancés dans cette ambition de créer une application pour permettre une communication entre le patient et le médecin, nous avons dû recommencer et changer plusieurs manières de procéder. Nous avons ainsi, améliorer ce point, sans pour autant terminer l'application et la rendre 100% fonctionnelle. Au contraire, si on prend l'exemple du stockage, nous n'avons pas pu résoudre ce manque de composant permettant de stocker toutes les données nécessaires avant un envoi sur ordinateur.

F. Résultats

F.1. Tests de performance

Les performances du dispositif ont été évaluées selon plusieurs critères, à savoir la précision des mesures, l'autonomie énergétique et la fiabilité des communications.

Précision des mesures actimétriques :

Les accéléromètres utilisés (MMA8451) ont permis de mesurer avec une précision de 14 bits les mouvements des deux bras sur les trois axes (x, y, z).

Nous avons effectué différents tests qui ont pour objectifs de mesurer la capacité des accéléromètres à différencier des mouvements d'intensités différentes :

- **Mouvements légers** : Balancement lent
- **Mouvements moyens** : Secousses modérées
- **Mouvements fort** : Secousses rapides

Les tests montrent que le dispositif détecte efficacement les différences de mouvement entre le bras sain et le bras atteint. Enfin, les résultats que l'on reçoit sont proportionnelles au mouvement que l'on applique.

Fiabilité de la communication sans fil :

Les performances des modules Bluetooth NRF24L01+ ont été testés pour garantir la transmission des données. En effet, elles ont été testées dans un rayon de 2.7m car la plus grande envergure jamais enregistrée entre 2 bras a été celle de Robert Wadlow [35] connu comme l'homme le plus grand de l'histoire avec un taille de 2,72m. Dans ce rayon-là, aucun dysfonctionnement n'a été à déplorer.

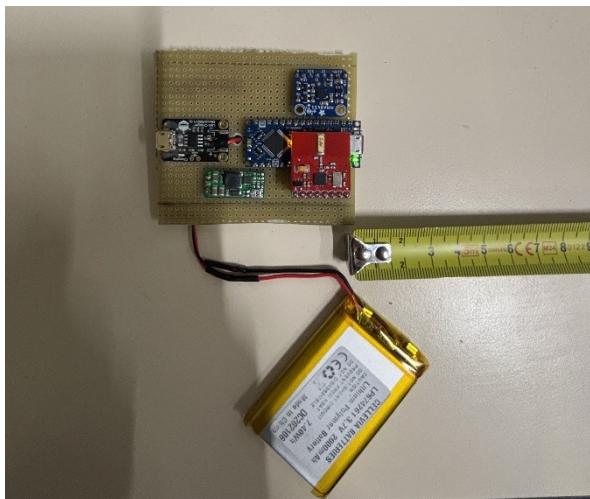


Figure 10 : Emplacement du bracelet

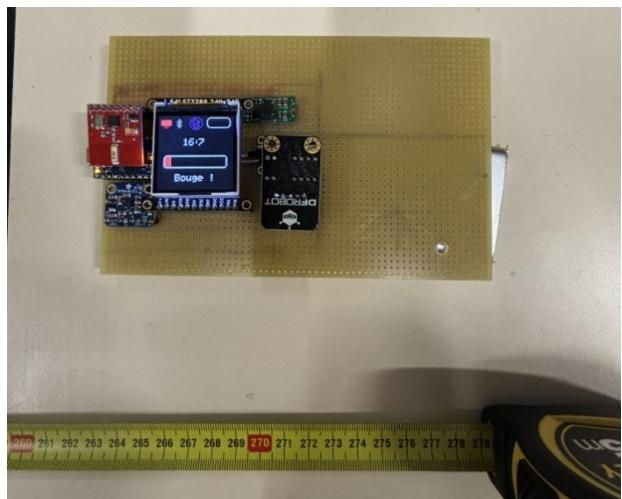


Figure 11 : Emplacement de la montre

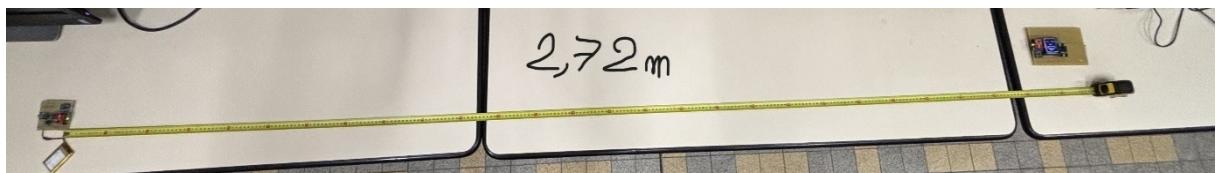


Figure 12 : Visualisation du test
Figure 12 : Visualisation de l'écart entre la montre et le bracelet

De plus, nous avons effectué d'autres séries de test :

Endroits	Distance Maximale de connexion
Dans le couloir du bâtiment 14	Portée de 17m
En extérieur	Portée de 22m

Tableau 11 : Comparatif de la distance de communication sans fil.

F.2. Résultats sur l'interface utilisateur

Une interface intuitive a été développée pour la montre, permettant au patient de vérifier rapidement si son bras atteint est suffisamment mobilisé. L'écran affiche :

- Le niveau d'activité du bras atteint, représenté par une jauge colorée.
- Le niveau de batterie, le statut de la connexion Bluetooth et de la connexion avec un ordinateur et le logo MEA [36].



Figure 16 à 19 : Images de l'affichage de l'écran en fonction des mouvements mesurés

Les tests de simulation sur les membres de l'équipe et d'autres équipes ont montré que tous les utilisateurs ont trouvé l'interface claire et facile à comprendre.

F.3. Application

L'application a été conçue pour permettre au patient de communiquer avec son médecin par e-mail. Elle offre la possibilité de transmettre les données stockées sur l'ordinateur de manière simple et efficace.

Concernant Processing :

Avant de passer sur Python, nous avions commencé à coder l'application sur Processing. Nous avons réussi à créer l'arrière-plan statique de l'application (Fig. 13) ainsi que de modifier le logo et le nom de l'application sur la barre des tâches et la barre de titre.

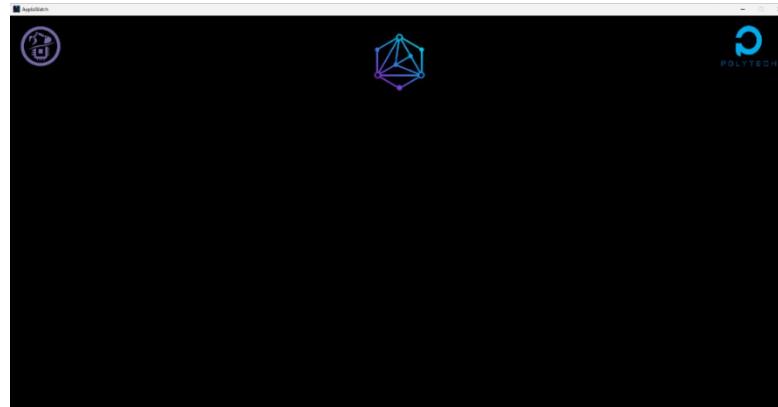


Figure 13 : Capture d'écran de l'application Processing

Nous avons également expérimenté un peu avec la librairie Serial de Processing pour la liaison avec la montre. Le premier prototype que nous avions fait permettait de changer la couleur d'un carré au centre de l'application à chaque fois que la carte Arduino envoyait le

caractère ‘A’. L’application renvoyait ensuite le caractère ‘B’ à la carte pour contrôler qu’on puisse bien communiquer dans les deux sens. Les programmes Processing sont en annexes.

Concernant Python :

L’application python peut être séparée en 3 grandes fonctionnalités : l’interface graphique, la liaison série avec la montre et le stockage des données et la représentation graphique des données.

Nous avons fait un concept de l’interface graphique sur tablette (Fig. 14) et celui-ci est segmentée en quatre quadrants (Fig. 15).

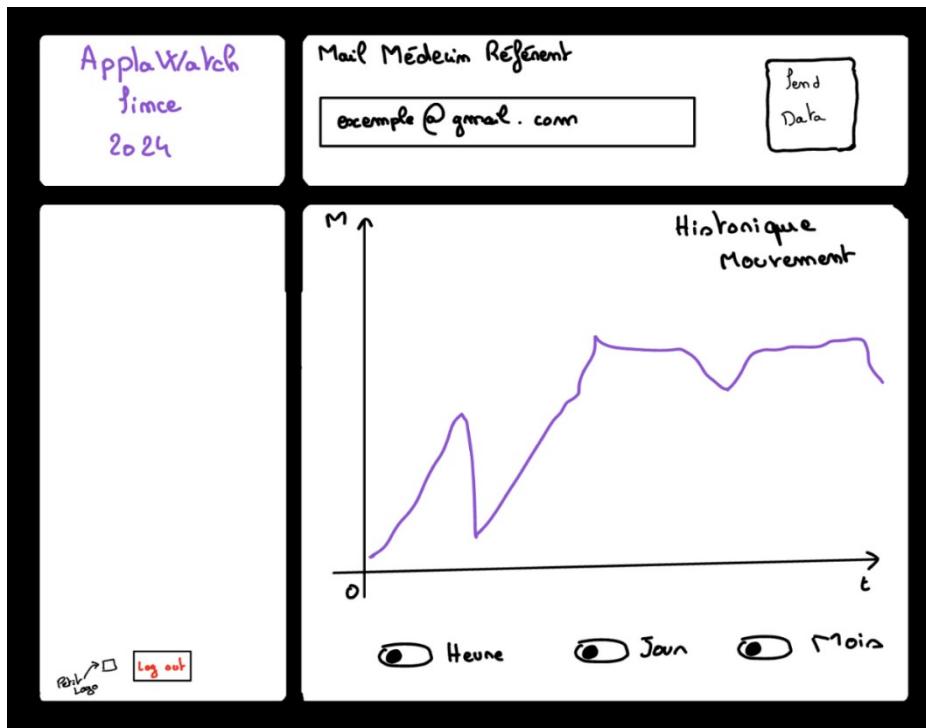


Figure 14 : Design de l’interface graphique

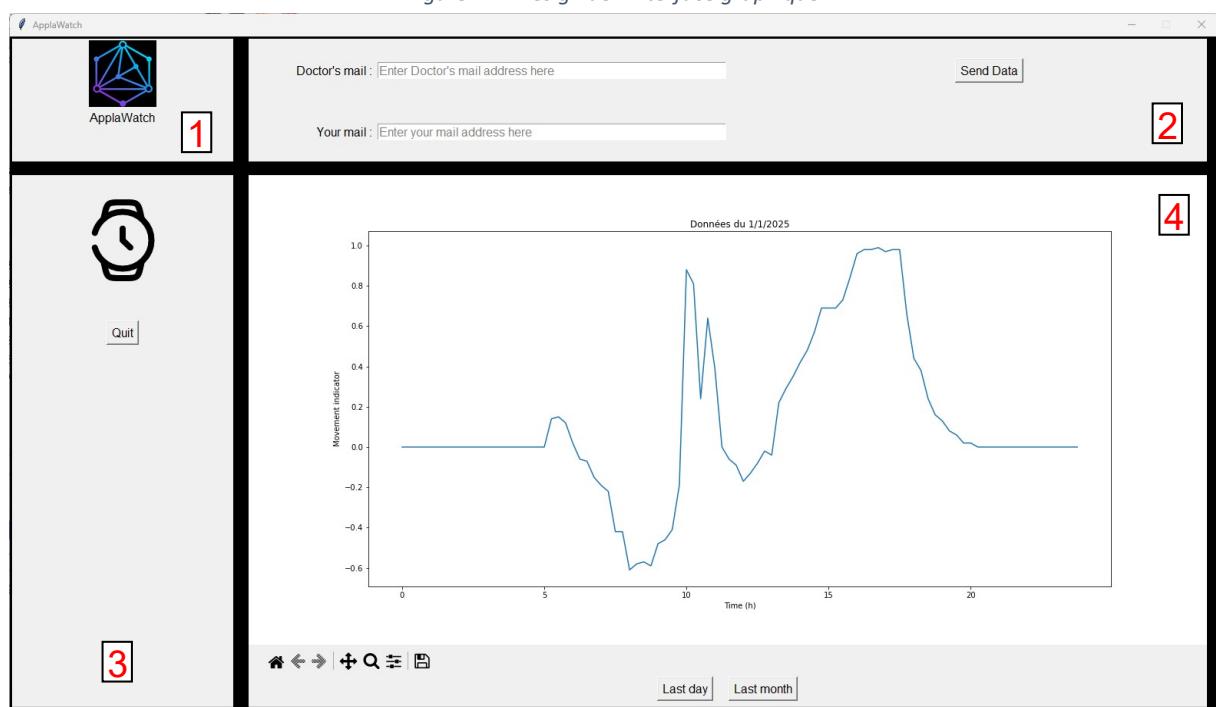


Figure 15 : Design de l'interface graphique

Le premier quadrant contient le logo et le nom de la marque. Le deuxième quadrant est le quadrant relatif à l'envoie des données. Il contient deux champs d'entrée de texte afin de pouvoir renseigner le mail du docteur et celui du patient, ainsi qu'un bouton pour envoyer les données et un champ de texte qui affiche d'éventuels messages d'erreur. Le troisième quadrant est celui dédié à la récupération des données et à la connexion avec la montre. Il contient un logo qui change de couleur pour indiquer si la montre est connectée ou non ainsi que le bouton pour fermer l'application. Enfin, le dernier quadrant est celui qui permet d'afficher les données du dernier jour ou du dernier mois stocké sur l'ordinateur.

F.4. Conclusions et perspectives

Objectifs atteints :

- Mesure fiable des mouvements du bras.
- Développement d'un prototype fonctionnel combinant une montre et un bracelet.
- Communication efficace des données via Bluetooth.
- Interface utilisateur opérationnelle.

Limites observées :

- Le design du boîtier doit encore être optimisé pour améliorer à la fois l'esthétique et l'ergonomie.
- La création de circuits imprimés (PCB) est encore en cours pour miniaturiser le dispositif.
- L'application pour ordinateur est partiellement fonctionnelle, elle nécessite des développements supplémentaires pour afficher des courbes évolutives des données et de permettre l'envoi de ces dernières.

Perspectives d'amélioration :

- Finalisation et intégration du design du boîtier.
- Rendre l'icône batterie fonctionnelle.
- Développement complet de l'application Windows pour le transfert et l'analyse des données par les professionnels de santé.
- Test à plus grande échelle avec des utilisateurs post-AVC pour valider le dispositif.

F.5. Présentation du prototype fonctionnel

Nous avons conçu et assemblé un prototype combinant une montre et un bracelet dédiés au suivi des mouvements post-AVC. Le dispositif est équipé des éléments suivants :

Montre :

- Équipée d'un écran OLED de 1,54 pouces affichant les indicateurs clés (mouvements détectés, statut Bluetooth, niveau de batterie, logo MEA, connexion avec un ordinateur).
- Elle intègre un accéléromètre pour mesurer les mouvements du bras atteint, ainsi qu'un module Bluetooth pour recevoir les données du bracelet.

Bracelet :

- Conçu pour être porté sur le bras sain, il est équipé d'un accéléromètre qui mesure les mouvements du bras dans les 3 axes : x, y, z.
- Il transmet les données mesurées à la montre via son module Bluetooth.

Matériaux et conception :

- Les composants électroniques ont été intégrés et soudés sur des platines de prototypage.
- Des boîtiers 3D ont été modélisés pour protéger les composants et assurer une utilisation ergonomique.

La montre :

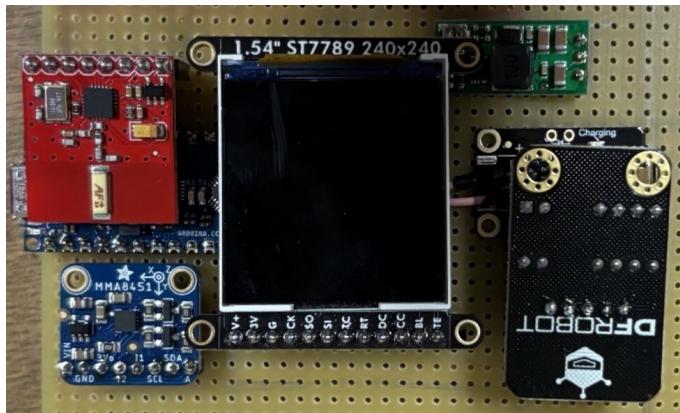


Figure 20 : Montage de la montre

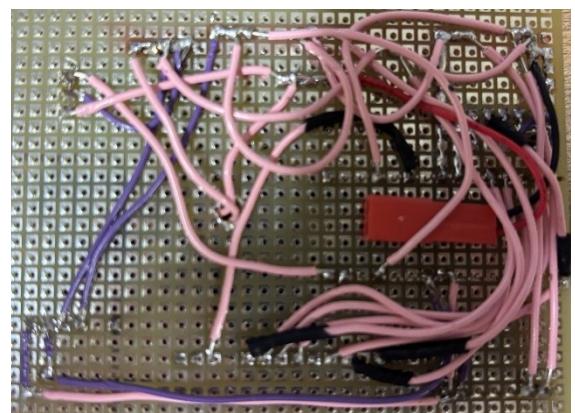


Figure 21 : Câblage de la montre

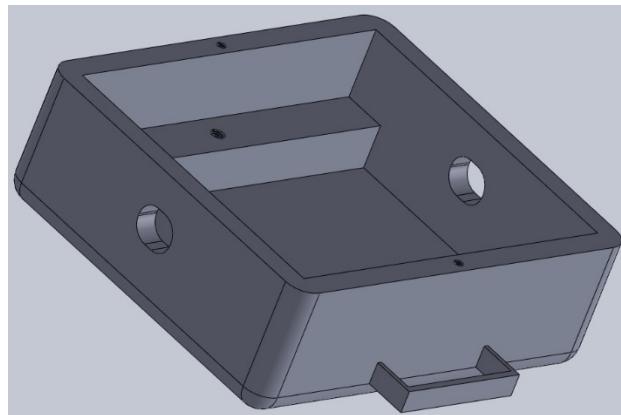


Figure 22 : Modèle 3D du boîtier de la montre

Le bracelet :

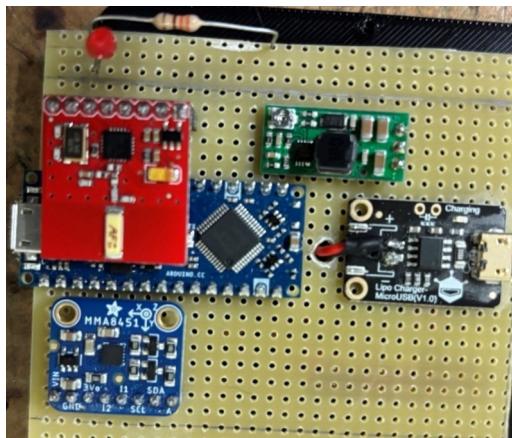


Figure 23 : Montage du bracelet

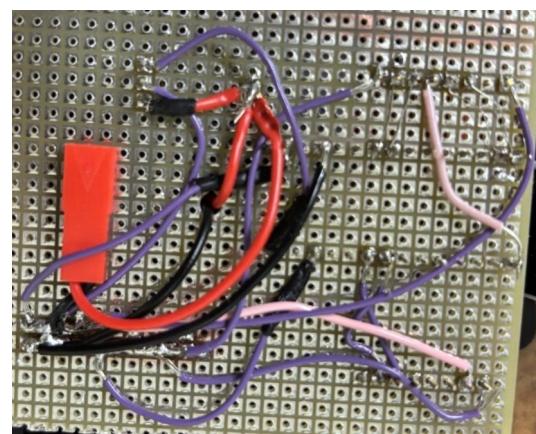


Figure 24 : Câblage du bracelet

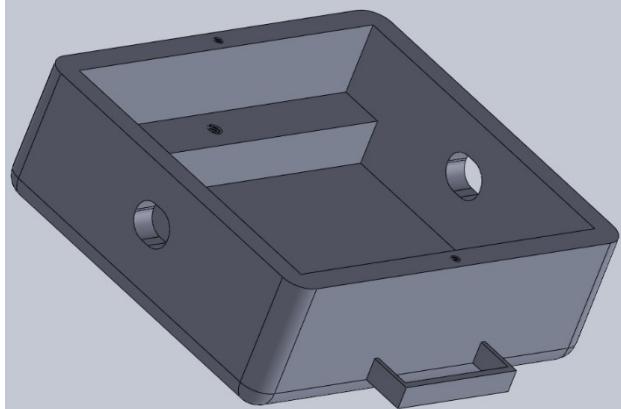


Figure 25 : Modèle 3D du boîtier du bracelet.



Figure 26 : Modèle 3D de la surface supérieure du boîtier du bracelet.





Figure 27 : Rendu final du bracelet dans son boitier.

G. Mode d'emploi

ApplaWatch

Mode d'emploi

Version 2.0

Table des matières

1 Introduction

- 1 A propos de l'ApplaWatch
- 1 A lire avant utilisation
- 1 Nettoyage et entretien de l'ApplaWatch

2 Chapitre 1 : Premiers Pas

- 2 Contenu de la boîte
- 2 Vue en détail
- 2 Touches
- 2 Batterie
- 2 Vérifier le statut de chargement de la batterie

- 2 Installer l'ApplaWatch
 - 2 Affichage
 - 2 Icône
-
- 3 [Chapitre 2 : L'application](#)
 - 3 Téléchargement
 - 3 Installation
 - 3 Appairage du dispositif
 - 3 Visualisation des données
 - 3 Transfert des données au médecin
-
- 4 [Chapitre 3 : Sécurité et assistance](#)
 - 4 Assistance
 - 4 Remplacement de la batterie
 - 4 Sécurité

Introduction

A propos de l'ApplaWatch

1

L'ApplaWatch est un système de mesure de mouvement à but médical. Ce dispositif permet d'assurer un suivi quotidien de l'utilisation de vos membres supérieurs.

Ce dispositif est doté d'un écran OLED, d'un module RTC pour une gestion de l'heure, d'un accéléromètre pour la détection des mouvements et d'une connectivité Bluetooth. Il peut être utilisé pour suivre vos données en temps réel et vous offrir un retour utilisateur immédiat.

A lire avant utilisation

Veuillez lire attentivement ce mode d'emploi, afin de pouvoir utiliser l'appareil correctement et en toute sécurité.

Précautions concernant la résistance de l'appareil à l'eau et à la poussière

- Ne pas exposer le dispositif à l'eau
- N'utilisez pas l'appareil lorsque vous plongez dans l'eau, lorsque vous faites de la plongée avec un tuba ou pratiquez tout autre sport aquatique en eaux vives.
- Si vos mains sont humides, séchez-les soigneusement avant de manipuler l'appareil.
- N'exposer pas votre appareil à une quantité non raisonnable de poussière.

Précautions concernant la surchauffe

Si vous éprouvez un inconfort dû à une surchauffe lors de l'utilisation du dispositif, cessez de l'utiliser immédiatement et retirez-le de votre poignet.

Nettoyer et entretenir l'ApplaWatch

- Nettoyer régulièrement le dispositif avec un chiffon sec et doux. Eviter tout contact direct avec des produits chimiques qui puissent détériorer les parties en plastiques.

Premiers Pas

2

Contenu de la boîte

- 1 Mode d'emploi en Français
- 2 Câbles USB/Micro USB d'une longueur de 50 cm
- 2 Chargeurs USB
- 1 Montre
- 1 Bracelet
- 1 QR-Code menant à la page de téléchargement de l'application
- 1 chiffon doux et sec

Touches

Touche	Fonction
 Marche /Arrêt	<ul style="list-style-type: none">• Appuyer sur cette touche pour allumer l'écran.• Si l'écran est allumé, appuyer une seconde fois pour le mettre en veille.



Le bracelet ne contient aucun bouton de mise en veille, il est toujours actif.



Lors de la première mise en route et à chaque premier démarrage de la journée, une petite animation s'affichera à l'écran de la montre.

Batterie



Chargez la batterie pendant au moins deux heures avant d'utiliser l'appareil pour la première fois.

- 1** Branchez un câble micro-USB sur le port de chargement de la montre et l'autre extrémité sur le chargeur USB. Puis branchez-le sur une prise de courant. Faire de même avec le bracelet.
- 2** Branchez un câble micro-USB sur le port de chargement de la montre et l'autre extrémité sur le port USB d'un ordinateur. Faire de même avec le bracelet.

Vérifier le statut de chargement de la batterie



Le statut de chargement de la batterie sur le bracelet et sur la batterie se lit de différentes façons.

- 1** Statut de chargement de la montre

Témoin	Statut du chargement
Barre vide	<ul style="list-style-type: none">Batterie vide. Brancher rapidement le dispositif.
Barre à moitié pleine	<ul style="list-style-type: none">Batterie à moitié pleine.
Barre pleine	<ul style="list-style-type: none">Batterie pleine.

- 2** Statut de chargement du bracelet

Témoin (Led rouge)	Statut du chargement
Clignote	<ul style="list-style-type: none">Batterie vide. Brancher rapidement le dispositif.
Allumer	<ul style="list-style-type: none">Batterie en cours de chargement
Eteinte	<ul style="list-style-type: none">Batterie pleine.



Si vous utilisez plusieurs fois l'écran, la batterie se déchargera plus rapidement.

Installer l'ApplaWatch

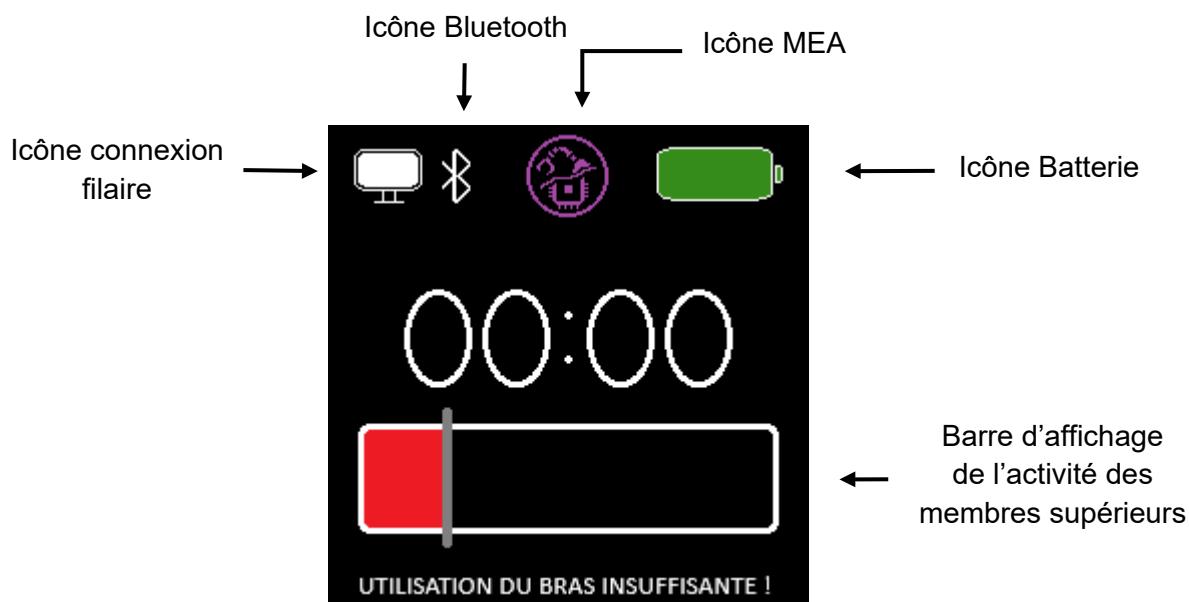
1^{ère} étape : Ouvrez la boucle du bracelet et mettez-la autour du poignet du membre sein. Ajustez le bracelet à votre poignet, et refermez la boucle.

2^{ème} étape : Ouvrez la boucle de la montre et mettez-la autour du poignet du membre atteint. Ajustez la montre à votre poignet, et refermez la boucle.



Ne pliez pas les attaches du bracelet et de la montre de manière excessive.
Ceci pourrait endommager l'ApplaWatch.

Affichage



L'affichage peut varier légèrement selon le modèle et les mises à jour.
Veuillez contacter l'assistance en cas de problème de compréhension.

Icône

Icône	Signification
	Statut de la connexion Bluetooth entre le bracelet et la montre. <ul style="list-style-type: none">Une icône blanche indique une connexion établie.Une icône rouge indique une connexion non établie.

3

	Statut de la connexion filaire entre la montre et l'ordinateur. <ul style="list-style-type: none">• Une icône blanche indique une connexion établie.• Une icône rouge indique une connexion non établie.
	Logo de la filière Microélectronique et Automatique où a été développé le dispositif. Cette icône n'est pas une icône d'information.
	Statut du mouvement de bras. <ul style="list-style-type: none">• Une icône pleine indique que vous bougez assez votre bras atteint.• Une icône vide indique que vous ne bougez pas assez votre bras atteint.• Un code couleur (rouge vers vert) permet également de repérer en un coup d'œil si vous bougez assez votre bras.

L'application

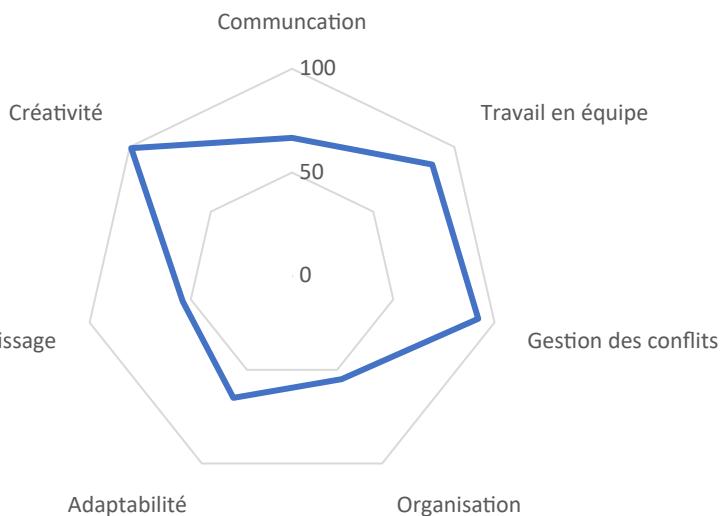
©2024 ApplaWatch Inc. Tous droits réservés. ApplaWatch, le logo ApplaWatch sont des marques de ApplaWatch Inc, déposées en France.

H. Retour d'expérience

Le retour d'expérience sur le projet met en avant le point de vue de chaque membre du groupe, illustré à travers un graphique des différentes compétences. Celui-ci a été élaboré à partir d'un document fourni par l'université Paul Sabatier [31] et complété par un retour personnel.

AURELIEN GOUMAIN

Conclusion - Mes compétences

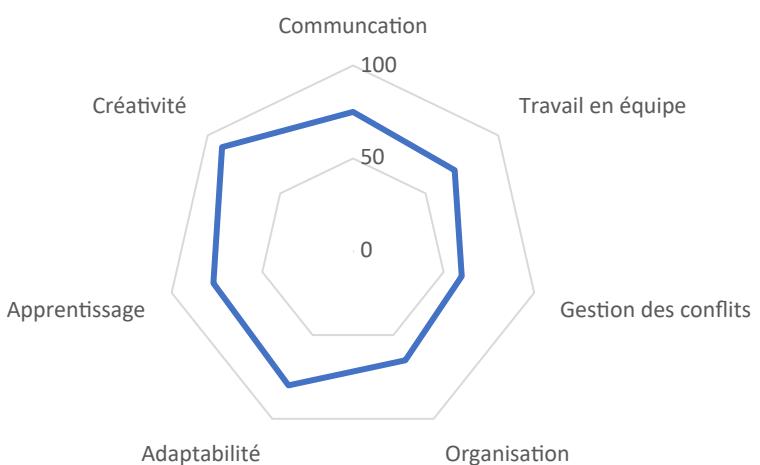


« J'ai énormément appris lors de ce projet de groupe, aussi bien sur le plan technique (comment choisir la batterie, les composants, le code pour la Bluetooth, etc.) que sur le plan de l'organisation de projet, comme organiser des réunions en début de séance pour savoir qui fait quoi, etc. J'ai aussi amélioré mes compétences relationnelles avec mes camarades, comme faire des compromis, exprimer mes idées correctement et écouter les autres avec attention. J'ai pu aussi mettre à profit ma créativité et mes innovations pour faire de ce projet quelque chose de vraiment unique ! »

ZYED KAMIL

Conclusion - Mes compétences

« Ce projet a été pour moi une découverte en électronique. Cela fut la première fois que j'achetais des composants et les codais afin de sortir quelque chose de fonctionnel. J'ai appris énormément de connaissances en codage. J'ai su donner mon avis quand il le fallait et j'ai apporté de la créativité et des connaissances en rédaction de rapport. »



BENJAMIN SAIGNE

Conclusion - Mes compétences

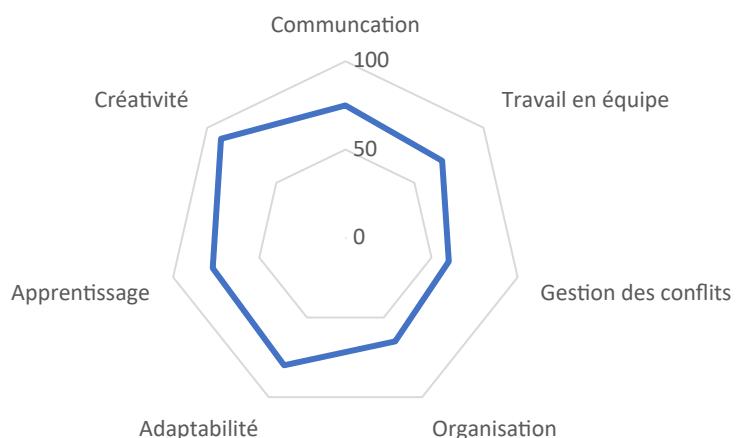


« Ce projet a été une excellente occasion pour moi de découvrir les projets en équipe. En effet, j'ai appris à gérer des situations plus ou moins complexes. De plus cette expérience m'a permis de surmonter des défis techniques, comme la gestion de l'alimentation et la connexion Bluetooth. Ce que j'en retiens, c'est qu'il est important pour réussir un projet de bien le planifier et bien répartir les tâches. »

PIERRE-MARIUS CAMASSES

« Ce projet a été une expérience enrichissante pour moi. Mon engagement m'a permis de développer des compétences techniques solides et d'élargir mes connaissances théoriques. Chaque étape de ce projet m'a permis de consolider mon socle de compétences et d'affiner mon esprit critique. Sur le plan professionnel et personnel, ce projet s'est révélé formateur dans la gestion du développement d'un produit. »

Conclusion - Mes compétences



Abel Diguet

Conclusion - Mes compétences



« En débutant ce projet je n'avais que très peu de connaissance dans le domaine de la micro électrique. Mais grâce à ce dernier je me suis entraîné et j'ai commencé à comprendre les bases de ce domaine. De plus j'ai pris conscience des difficultés qui pouvaient être rencontrées lors de travaux en groupe, comme les désaccords ou parfois les petits problèmes d'organisation, mais j'ai surtout appris à régler ces derniers au sein d'une équipe. »

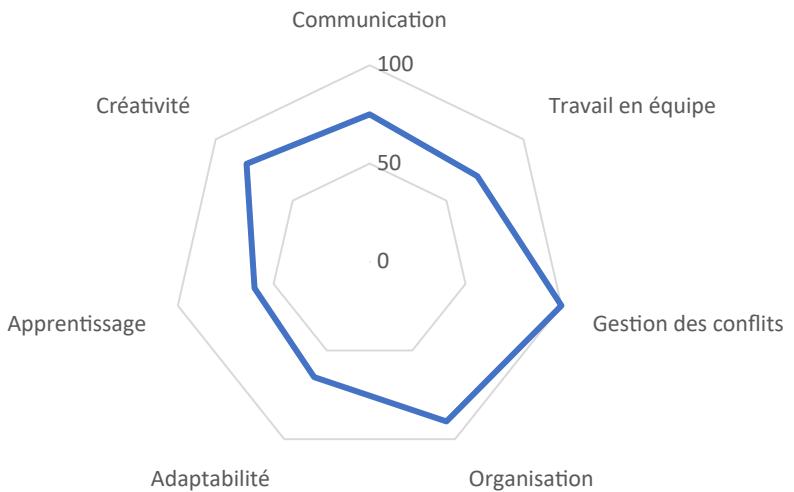
Adam MABROUQUE

« Durant ce projet, j'ai pu découvrir un peu plus le travail en grand groupe. Le fait qu'on soit 7 nous permet de travailler sur plus de tâches en même temps, mais cela crée aussi plus de questions et d'opinions différentes, et ainsi des conflits. Ce qui fut agréable, c'est que tous les conflits ont été réglés rapidement et sans problèmes. Travailler en grand groupe permet aussi d'avoir plus de réponses possibles à ces questions, et donc plus de solutions. Néanmoins, dû au grand nombre de tâches, je n'ai pas pu apprendre autant que je le souhaitais. Je suis tout de même satisfait car j'ai pu en apprendre

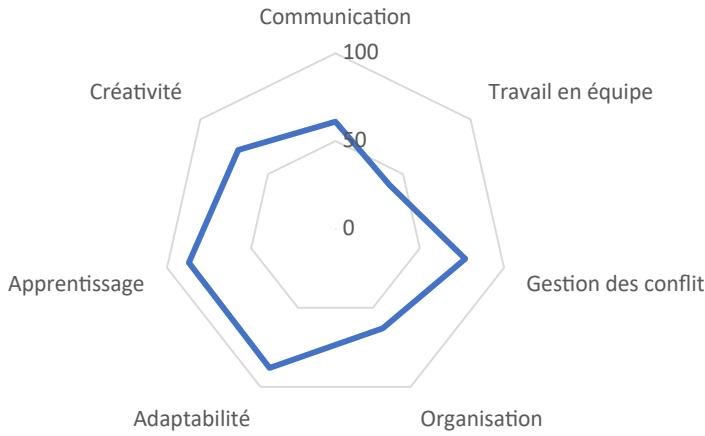
davantage sur plusieurs thèmes, comme la soudure ou la programmation, qui sont des thèmes qui m'intéressaient grandement. L'aide de mes camarades dans ces différents thèmes m'ont donc beaucoup apporté. »

Alexandre DE JESUS MARTINS

Conclusion - Mes compétences



Conclusion - Mes compétences



« Ce projet a été très enrichissant pour moi. Si j'avais déjà une expérience de travail en équipe, c'était principalement en équipe restreinte, allant du binôme au groupe de 4-5 étudiants au plus. Ce projet a été le premier en équipe aussi grande, à savoir 7 personnes. Cette équipe étendue a clairement montré l'importance de la communication et de l'organisation pour bien travailler. Ces compétences ont longtemps été des bêtes noires pour moi, mais ce projet m'a permis de travailler dessus et de les développer.

Au-delà de l'expérience de travail en équipe, j'avais déjà des compétences techniques en électronique et en informatique, acquises sur des projets impliquant de l'Arduino et une application python. Cependant j'ai pu étoffer mes compétences en informatique en travaillant sur l'application et son interface graphique, aspect de la programmation que je n'avais jamais touché auparavant. »

Retour d'expérience du groupe de projet

Lors de la réalisation de ce projet, notre groupe a vécu une expérience extrêmement enrichissante que ce soit en termes d'apprentissage ou sur le relationnel. Dès les premiers instants, l'entente et la collaboration se sont établies au sein du groupe, chaque membre de ce dernier a réussi à trouver sa place.

Par la suite, cette expérience nous a permis de tirer de nombreuses leçons. Nous avons réalisé l'importance d'anticiper les contraintes matérielles dès les premières phases et de renforcer les tests unitaires d'intégration pour limiter les retards.

Par ailleurs, améliorer les communications avec les partenaires et savoir trouver une solution à chaque problème rencontré se sont révélés essentiels pour éviter toute forme de blocage.

Enfin, ce projet a été une réussite globale malgré les défis rencontrés. Nous avons atteint la majorité de nos objectifs initiaux.

I. Webographie

Datasheets :

- [7] Arduino, « Arduino Nano Every », ABX00028 datasheet, November. 2024.
- [13] Nordic Semiconductor, « Single Chip 2.4GHz Transceiver », nRF24L01+ datasheet, September. 2008.
- [9] Freescale Semiconductor, “Xtrinsic MMA8451Q 3-axis, 14-bit/8bit Digital Accelerometer,” MMA8451Q datasheet, Feb. 2013.
- [18] Non renseigné, “High precision RTC Module-SD2405AL”, RTC datasheet, Non renseigné.

Sites internet:

- [14] TONI_K. (Non renseigné). NRF24L01+ Transceiver Hookup Guide [Online]. Available: <https://learn.sparkfun.com/tutorials/nrf24l01-transceiver-hookup-guide>
- [7] Kattni Rembor, M. LeBlanc-Williams, Eva Herrada. (2023). Adafruit 1.3 and 1.54, 240*240 Wide Angle TFT LCD Displays [Online]. Available: <https://learn.adafruit.com/adafruit-1-3-and-1-54-240-x-240-wide-angle-tft-lcd-displays?view=all>
- [20] Arduino. (2024). Arduino IDE 2.3.3 [Online]. Available : <https://www.arduino.cc/en/software>
- [9] Arobases. (2012-2024). Gotronic Robotique et composants électroniques [Online]. Available : <https://www.gotronic.fr>
- [32] Rs Components. (Non renseigné). RS [Online]. Available : <https://fr.rs-online.com/web/>
- [33] Farnell. (Non renseigné). Farnel an avnet company [Online]. Available :

<https://fr.farnell.com>

- [12] Jerome (Updated 2024). Module NRF24L01 Arduino : caractéristiques, librairies, et explications (tutoriel avec exemples de code arduino) [Online]. Available :
<https://passionelectronique.fr/tutorial-nrf24l01/>
- [30] WhatsApp. (Non renseigné). WhatsApp [Online]. Available :
https://www.whatsapp.com/?lang=fr_FR
- [31] GROUPE2. (2024). Actimetrie [Online]. Available :
<https://gitlab.polytech.umontpellier.fr/ProjetTeamAppleWatch/actimetrie>
- [15] GROUPE2. (2024). Beesbusy [Online]. Available :
<https://learn.sparkfun.com/tutorials/nrf24l01-transceiver-hookup-guide>
- [1] AssuranceMaladie.(2025).Comprendre l'accident vasculaire cérébral et l'accident ischémique transitoire [Online].Available:
<https://www.ameli.fr/assure/sante/themes/accident-vasculaire-cerebral-avc/avc-comprendre>
- [2] Dr Bertrand Lapergue.(Non renseigné).Tout savoir sur l'accident vasculaire cérébral [Online]. Available :
<https://www.frm.org/fr/maladies/recherches-maladies-cardiovasculaires/accident-vasculaire-cerebral/focus-sur-l-accident-vasculaire-cerebral>
- [3] Gotronic.(2025). Carte ESP32-WROOM-32 GT0162 [Online].Available :
[Carte ESP32-WROOM-32 GT0162 - Gotronic](#)
- [4] RaberryPi.(2025). Raspberry Pi [Online]. Available :
[Buy a Raspberry Pi – Raspberry Pi](#)
- [5] STMicroelectronics.(2025).STM32 Nucleo boards [Online].Available :
[STM32 Nucleo boards - STMicroelectronics](#)
- [6] Arduino.(2025). Nano Every [Online]. Available :
[Nano Every | Arduino Documentation](#)
- [8] Gotronic.(2025). Accéléromètre 3 axes MMA8451 [Online]. Available :
[Accéléromètre 3 axes MMA8451 - Gotronic](#)
- [10] LoRa Alliance.(2025). What is LoRaWAN ? [Online]. Available :
[About LoRaWAN® - LoRa Alliance®](#)
- [11] Gotronic.(2025). Module Bluetooth HC05 [Online]. Available :
[Module Bluetooth HC05 - Gotronic](#)

- [12] Jérôme.(2024). Module NRF24L01 Arduino : caractéristiques, librairies, et explications (tutorial avec exemples de code arduino) [Online]. Available :
[Tuto NRF24L01 : code arduino, librairie, fonctionnement, ...](#)
- [15] QuaiLab.(2025). Arduino et ses mémoires [Online]. Available :
[Arduino et ses mémoires | Quai Lab](#)
- [16] Kaouthar Draif.(2023). Afficheur Arduino : Comprendre les différents afficheurs pour votre projet Arduino [Online]. Available :
[Afficheur Arduino : Comprendre les différents afficheurs pour votre projet Arduino - Moussasoft](#)
- [17] Dalian Good Display.(Non renseigné). E-Paper display FAQ tools [Online]. Available :
[Affichage e-paper -- FAQ_Good Display](#)
- [19] Jerome.(2023). Watchdog Arduino : explication de fonctionnement, et exemples de code (mode reset, timer interruption, et réveil après sleep) [Online]. Available :
[Watchdog Arduino : Tutorial avec Exemples de Code !](#)
- [23] Solidworks.(Non renseigné). Solidworks est la solution éprouvée pour la conception 3D et le développement de produits [Online]. Available :
[La solution éprouvée pour la conception 3D et le développement de produits | SOLIDWORKS](#)
- [22] Onshape.(Non renseigné). Le leader de la CAO et du PDM cloud native [Online]. Available:
[Onshape | Plateforme de développement de produits](#)
- [21] Wikipedia.(2025). Circuit imprimé [Online]. Available:
[Circuit imprimé — Wikipédia](#)
- [24] Python.(Non renseigné). Python [Online]. Available :
[Welcome to Python.org](#)
- [25] PyPi.(Non renseigné). Pyserial 3.5 [Online]. Available :
[pyserial · PyPI](#)
- [26] Tkinter.(Non renseigné). Tkinter --- Python interface to Tcl/Tk [Online]. Available :
[tkinter --- Python interface to Tcl/Tk — Documentation Python 3.13.1](#)
- [27] Matplotlib.(Non renseigné) Matplotlib : Visualization with Python [Online]. Available :
[Matplotlib — Visualization with Python](#)
- [28] Numpy.(2024). Numpy [Online]. Available :
[NumPy](#)

- [29] beesbusy.(2025). Planification et pilotage des travaux [Online]. Available: [Outil de travail collaboratif de gestion de projets -Beesbusy](#)
- [34] Processing.(2025). Welcome to Processing ! [Online]. Available : [Welcome to Processing! / Processing.org](#)
- [35] Wikipedia.(2024). Robert Wadlow [Online]. Available : [Robert Wadlow — Wikipédia](#)
- [36] Polytech Montpellier.(2022). Microélectronique et automatique [Online]. Available : <https://www.polytech.umontpellier.fr/formation/cycle-ingenieur/microelectronique-et-automatique/enquelques-mots-me>

J. Annexes

Code Arduino du bracelet :

```
1. //-----NRF24L01-----//
2. #include <SPI.h>
3. #include "printf.h"
4. #include "RF24.h"
5.
6. #define CE_PIN 5
7. #define CSN_PIN 6
8.
9. RF24 radio(CE_PIN, CSN_PIN);
10.
11. uint8_t address[][6] = { "1Node", "2Node" };
12. bool radioNumber = 1; // 0 uses address[0] to transmit, 1 uses address[1] to transmit
13. bool role = false; // true = TX role, false = RX role
14.
15. //-----Accelerometre-----//
16. #include <Wire.h>
17. #include <Adafruit_MMA8451.h>
18. #include <Adafruit_Sensor.h>
19. Adafruit_MMA8451 mma = Adafruit_MMA8451();
20.
21. //-----Variables-----//
22.
23. float x=0; //Valeur d'accélération selon x
24. float x_final=0;
25. float y=0; //Valeur d'accélération selon y
26. float y_final=0;
27. float z=0; //Valeur d'accélération selon z
28. float z_final=0;
29. float IndiceMVBracelet=0; //Indice de mouvement du bracelet
30.
31. void setup() {
32.
33. //Initialisation PortSerie à une vitesse de 9600 Bauds
34. Serial.begin(9600);
35.
36. //Initialisation NRF24L01
37. if (!radio.begin()) {
38. Serial.println(F("radio hardware is not responding!!"));
39. while (1) {} // hold in infinite loop
40. }
41. radio.setPALevel(RF24_PA_LOW);
42. radio.setPayloadSize(sizeof(IndiceMVBracelet)); // float datatype occupies 4 bytes
43. radio.openWritingPipe(address[0]);
44. radio.stopListening(); // put radio in RX mode
45. //Initialisation Accelerometre
46. if (! mma.begin()) {
47. Serial.println("Couldnt start");
48. while (1);
49. }
```

```

50. Serial.println("MMA8451 found!");
51.
52. mma.setRange(MMA8451_RANGE_4_G);
53.
54. delay(250);
55.
56. }
57.
58. void loop() {
59.
60. //-----Lecture des valeurs de l'accéléromètre-----//
61. mma.read();
62. if(abs(mma.x)>(x+200) || abs(mma.x)<(x-200)){
63.   x=abs(mma.x);
64.   x_final=x_final+(x/831200);
65.   IndiceMVBracelet=IndiceMVBracelet+ x_final;
66. }
67. if(abs(mma.y)>(y+200) || abs(mma.y)<(y-200)){
68.   y=abs(mma.y);
69.   y_final=y_final+(y/831200);
70.   IndiceMVBracelet=IndiceMVBracelet+ y_final;
71. }
72. if(abs(mma.z)>(z+200) || abs(mma.z)<(z-200)){
73.   z=abs(mma.z);
74.   z_final=z_final+(z/831200);
75.   IndiceMVBracelet=IndiceMVBracelet+ z_final;
76. }
77.
78. //-----Envoi des valeurs à la montre-----//
79.
80. bool report = radio.write(&IndiceMVBracelet, sizeof(float)); // transmit & save the
report
81.
82.
83. /*if (report) {
84.   Serial.print(F("Envoie des données réussis !")); // payload was delivered
85.   Serial.print(F("Valeur Envoyée: "));
86.   Serial.println(IndiceMVBracelet); // print payload sent
87. } else {
88.   //Serial.println(F("Problème de communication")); // payload was not delivered
89. }*/
90.
91. */
92.
93. }

```

Code Arduino de la montre :

```

1. //-----Init Couleur-----//
2.
3. #define BLACK      0x0000
4. #define VIOLET     0x925A
5. #define CYAN       0x67FC
6. #define WHITE      0xFFFF
7. #define RED        0xF800
8. #define BLUE       0x011F
9. #define ORANGE     0xFAC0
10. #define YELLOW     0xFF20
11. #define GREEN      0x0F00
12. #define DARKGREEN  0x2AC8
13. //-----Init Ecran OLED-----//

```

```

14.
15. #include <Adafruit_GFX.h>      // Core graphics library
16. #include <Adafruit_ST7789.h> // Hardware-specific library for ST7789
17. #include <SPI.h>
18.
19. #define TFT_CS          10
20. #define TFT_RST         9
21. #define TFT_DC          8
22.
23. Adafruit_ST7789 display = Adafruit_ST7789(TFT_CS, TFT_DC, TFT_RST);
24.
25. // 'Icone ordi allumé', 50x50px
26. const unsigned char epd_bitmap_Icone_ordi_allum_ [] PROGMEM = {
27.   0x00, 0x00,
0x00, 0x00,
28.   0x00, 0x00,
0x00, 0x00,
29.   0x00, 0x00,
0x00, 0x00,
30.   0x00, 0x00,
0x00, 0x00,
31.   0x00, 0x00,
0x00, 0x00,
32.   0x00, 0x00,
0xff, 0xfe,
33.   0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x4f, 0xff, 0xfc,
0x80, 0x00,
34.   0x00, 0x9f, 0xff, 0xff, 0xfe, 0x40, 0x00, 0x00, 0xbff, 0xff, 0xff, 0x40, 0x00,
0x00, 0xbff,
35.   0xff, 0xff, 0xff, 0x40, 0x00, 0xbff, 0xff, 0xff, 0x40, 0x00, 0x00, 0xbff, 0x0ff,
0x0ff,
36.   0xff, 0x40, 0x00, 0x00, 0xbff, 0xff, 0xff, 0x40, 0x00, 0x00, 0xbff, 0xff, 0x0ff,
0x0ff, 0x40,
37.   0x00, 0x00, 0xbff, 0xff, 0x40, 0x00, 0x00, 0xbff, 0xff, 0xff, 0x40, 0x00, 0x00,
0x00, 0xbff,
38.   0xbff, 0xff, 0xff, 0x40, 0x00, 0x00, 0xbff, 0xff, 0xff, 0x40, 0x00, 0x00, 0x00,
0xbff, 0x0ff,
39.   0xff, 0xff, 0x40, 0x00, 0xbff, 0xff, 0xff, 0x40, 0x00, 0x00, 0xbff, 0x0ff, 0x0ff,
0x0ff,
40.   0x40, 0x00, 0x00, 0x9f, 0xff, 0xfe, 0x40, 0x00, 0x00, 0x47, 0xff, 0xff, 0xf8,
0x80, 0x00,
41.   0x00, 0x30, 0x00, 0x00, 0x03, 0x00, 0x00, 0x0f, 0xff, 0xfc, 0x00, 0x00,
0x00, 0x00,
42.   0x02, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x08, 0x00, 0x00, 0x00, 0x00,
0x00, 0x08,
43.   0x00, 0x00, 0x00, 0x00, 0x02, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x08,
0x00,
44.   0x00, 0x00, 0x00, 0x3f, 0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,
45.   0x00, 0x00,
0x00, 0x00,
46.   0x00, 0x00,
0x00, 0x00,
47.   0x00, 0x00,
0x00, 0x00,
48.   0x00, 0x00,
0x00, 0x00,
49. };
50.
51. const int epd_bitmap_allArray_LEN1= 1;
52. const unsigned char* epd_bitmap_allArray1[1] = {
53.   epd_bitmap_Icone_ordi_allum_
54. };
55.
56. // 'Icone ordi éteint', 50x50px
57. const unsigned char epd_bitmap_Icone_ordi_teint [] PROGMEM = {

```

```

58.    0x00, 0x00,
0x00, 0x00,
59.    0x00, 0x00,
0x00, 0x00,
60.    0x00, 0x00,
0x00, 0x00,
61.    0x00, 0x00,
0x00, 0x00,
62.    0x00, 0x00,
0x00, 0x00,
63.    0x00, 0x00,
0xff, 0xfe,
64.    0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x40, 0x00, 0x00, 0x00,
0x80, 0x00,
65.    0x00, 0x80, 0x00, 0x00, 0x40, 0x00, 0x00, 0x80, 0x00, 0x00, 0x40, 0x00,
66.    0x00, 0x00, 0x40, 0x00, 0x80, 0x00, 0x00, 0x00, 0x40, 0x00, 0x80, 0x00, 0x00,
67.    0x00, 0x40, 0x00, 0x80, 0x00, 0x00, 0x40, 0x00, 0x80, 0x00, 0x00, 0x40, 0x00,
68.    0x00, 0x00, 0x80, 0x00, 0x00, 0x40, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x40,
69.    0x80, 0x00, 0x00, 0x40, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x40, 0x00, 0x00,
70.    0x00, 0x00, 0x40, 0x00, 0x80, 0x00, 0x00, 0x00, 0x40, 0x00, 0x80, 0x00, 0x00,
71.    0x40, 0x00, 0x00, 0x80, 0x00, 0x00, 0x40, 0x00, 0x00, 0x40, 0x00, 0x00, 0x00,
72.    0x00, 0x30, 0x00, 0x00, 0x03, 0x00, 0x00, 0x0f, 0xff, 0xfc, 0x00, 0x00,
73.    0x02, 0x08, 0x00, 0x00, 0x00, 0x00, 0x02, 0x08, 0x00, 0x00, 0x00, 0x00,
74.    0x00, 0x00, 0x00, 0x00, 0x02, 0x08, 0x00, 0x00, 0x00, 0x00, 0x02, 0x08,
75.    0x00, 0x00, 0x00, 0x3f, 0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
76.    0x00, 0x00,
77.    0x00, 0x00,
78.    0x00, 0x00,
79.    0x00, 0x00,
80.    };
81.
82. const int epd_bitmap_allArray_LEN2= 1;
83. const unsigned char* epd_bitmap_allArray2[1] = {
84.     epd_bitmap_Icone_ordi_teint
85. };
86.
87. // 'Icone bluetooth allumé', 50x50px
88. const unsigned char epd_bitmap_Icone_bluetooth_allum_ [] PROGMEM = {
89.     0x00, 0x00,
0x00, 0x00,
90.     0x00, 0x00,
0x00, 0x00,
91.     0x00, 0x00,
0x00, 0x00,
92.     0x00, 0x00,
0x00, 0x00,
93.     0x00, 0x00,
0x00, 0x00,
94.     0x00, 0x00, 0x00, 0x00, 0x01, 0x80, 0x00, 0x00, 0x00, 0x00, 0x03, 0xc0,
0x00, 0x00,

```

```

95.    0x00, 0x00, 0x00, 0x03, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,
96.    0x00, 0x03, 0x18, 0x00, 0x00, 0x00, 0x00, 0x03, 0x0c, 0x00, 0x00, 0x00, 0x00,
0x00, 0x43,
97.    0x06, 0x00, 0x00, 0x00, 0x00, 0x63, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x33,
0x06, 0x00,
98.    0x00, 0x00, 0x00, 0x00, 0x1b, 0x0c, 0x00, 0x00, 0x00, 0x00, 0x0f, 0x18, 0x00,
0x00, 0x00,
99.    0x00, 0x00, 0x07, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x60, 0x00, 0x00, 0x00,
0x00, 0x00,
100.   0x03, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x07, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00,
0x0f, 0x30,
101.   0x00, 0x00, 0x00, 0x00, 0x1b, 0x18, 0x00, 0x00, 0x00, 0x00, 0x33, 0x0c,
0x00, 0x00,
102.   0x00, 0x00, 0x63, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x43, 0x02, 0x00, 0x00,
0x00, 0x00,
103.   0x00, 0x03, 0x06, 0x00, 0x00, 0x00, 0x00, 0x03, 0x0c, 0x00, 0x00, 0x00, 0x00,
0x00, 0x03,
104.   0x18, 0x00, 0x00, 0x00, 0x00, 0x03, 0x30, 0x00, 0x00, 0x00, 0x00, 0x03,
0x60, 0x00,
105.   0x00, 0x00, 0x00, 0x00, 0x03, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x01, 0x80, 0x00,
0x00, 0x00,
106.   0x00, 0x00,
0x00, 0x00,
107.   0x00, 0x00,
0x00, 0x00,
108.   0x00, 0x00,
0x00, 0x00,
109.   0x00, 0x00,
0x00, 0x00,
110.   0x00, 0x00,
0x00, 0x00
111. };
112.
113. const int epd_bitmap_allArray_LEN3 = 1;
114. const unsigned char* epd_bitmap_allArray3[1] = {
115.     epd_bitmap_Icone_bluetooth_allum_
116. };
117.
118. // 'Icone MEAB', 50x50px
119. const unsigned char epd_bitmap_Icone_MEAB [] PROGMEM = {
120.     0x00, 0x00,
0x00, 0x00,
121.     0x00, 0x00,
0x00, 0x00,
122.     0x00, 0x00, 0x00, 0x00, 0x07, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x3f, 0xff,
0x80, 0x00,
123.     0x00, 0x00, 0xf8, 0x03, 0xe0, 0x00, 0x00, 0x00, 0x03, 0xc0, 0x00, 0x78, 0x00,
0x00, 0x00,
124.     0x07, 0x00, 0x00, 0x1c, 0x00, 0x00, 0xe0, 0x00, 0x00, 0xe0, 0x00, 0x00, 0x00,
0x00, 0x00,
0x18, 0x17,
125.     0x80, 0x03, 0x00, 0x00, 0x30, 0x3b, 0xc0, 0x01, 0x80, 0x00, 0x00, 0x60, 0x90,
0xf0, 0x00,
126.     0xc0, 0x00, 0x00, 0xe0, 0xc0, 0x60, 0x00, 0xe0, 0x00, 0x00, 0xc0, 0x60, 0x48, 0x00,
0x60, 0x00,
127.     0x01, 0x81, 0x00, 0x1c, 0x00, 0x30, 0x00, 0x01, 0x87, 0x80, 0x09, 0xc0, 0x30, 0x00,
0x03, 0xd0,
128.     0x80, 0x03, 0xe0, 0x18, 0x00, 0x03, 0x18, 0xc0, 0x07, 0xf8, 0x18, 0x00, 0x03, 0x30,
0x60, 0x07,
129.     0xf0, 0x18, 0x00, 0x06, 0x30, 0x60, 0x07, 0x0f, 0x0c, 0x00, 0x06, 0x10, 0x60, 0x0e,
0x78, 0xc0,
130.     0x00, 0x06, 0x00, 0xc0, 0x0c, 0xc0, 0x00, 0x06, 0x01, 0x00, 0x03, 0x80, 0x0c,
0x00, 0x06,
131.     0x00, 0x00, 0x1e, 0x00, 0x0c, 0x00, 0x06, 0x00, 0x3f, 0xf0, 0x40, 0x0c, 0x00, 0x06,
0x00, 0x60,

```

```

132.     0x00, 0xc0, 0x0c, 0x00, 0x06, 0x01, 0xc0, 0x00, 0xc0, 0x0c, 0x00, 0x06, 0x07, 0x10,
0x00, 0xf0,
133.     0x0c, 0x00, 0x03, 0x0c, 0x30, 0xf0, 0xc0, 0x18, 0x00, 0x03, 0x08, 0xf0, 0xf0, 0xf0,
0x18, 0x00,
134.     0x03, 0x00, 0x30, 0xf0, 0xc0, 0x18, 0x00, 0x01, 0x80, 0xf0, 0xf0, 0xf0, 0x30, 0x00,
0x01, 0x80,
135.     0x30, 0x00, 0xc0, 0x30, 0x00, 0xc0, 0xf0, 0x00, 0xf0, 0x60, 0x00, 0x00, 0xe0,
0x30, 0x00,
136.     0xc0, 0xe0, 0x00, 0x00, 0x60, 0x30, 0x00, 0xc0, 0xc0, 0x00, 0x00, 0x30, 0x1f, 0xff,
0x81, 0x80,
137.     0x00, 0x00, 0x18, 0x0f, 0xff, 0x03, 0x00, 0x00, 0x0e, 0x04, 0x92, 0x0e, 0x00,
0x00, 0x00,
138.     0x07, 0x04, 0x92, 0x1c, 0x00, 0x00, 0x03, 0xc0, 0x00, 0x78, 0x00, 0x00, 0x00,
0x00, 0xf8,
139.     0x03, 0xe0, 0x00, 0x00, 0x00, 0x3f, 0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0x07,
0xfc, 0x00,
140.     0x00, 0x00,
0x00, 0x00,
141.     0x00, 0x00,
0x00, 0x00
142. };
143.
144. const unsigned char* epd_bitmap_allArray4[1] = {
145.     epd_bitmap_Icone_MEAB
146. };
147.
148. //-----Init Module RTC-----//
149.
150. #include "GravityRtc.h"
151. #include "Wire.h"
152.
153. GravityRtc rtc;      //RTC Initialization
154.
155. //-----Init NRF24L01-----//
156.
157. #include <SPI.h>
158. #include "printf.h"
159. #include "RF24.h"
160.
161. #define CE_PIN 5
162. #define CSN_PIN 6
163.
164. RF24 radio(CE_PIN, CSN_PIN);
165.
166. uint8_t address[][6] = { "1Node", "2Node" };
167. bool radioNumber = 1; // 0 uses address[0] to transmit, 1 uses address[1] to transmit
168. bool role = true; // true = TX role, false = RX role
169.
170. //-----Init Accelerometre-----//
171.
172. #include <Wire.h>
173. #include <Adafruit_MMA8451.h>
174. #include <Adafruit_Sensor.h>
175. Adafruit_MMA8451 mma = Adafruit_MMA8451();
176.
177. //-----Init Variables-----//
178.
179. float x=0; //Valeur d'accélération selon x
180. float x_final=0;
181. float y=0; //Valeur d'accélération selon y
182. float y_final=0;
183. float z=0; //Valeur d'accélération selon z
184. float z_final=0;
185. float IndiceMVBracelet=0; //Indice de mouvement du bracelet
186. float IndiceMVMontre=0; //Indice de mouvement de la montre
187. float ValeurP=0;

```

```

188. float ValeurAntP=0;
189. int rtcAntMin=0;
190. int rtcAntHour=0;
191. int rtcAntSecond =0;
192. uint16_t color;
193.
194. const int buttonPin = 2;           // Pin du bouton
195. unsigned long debounceTime = 15; // Temps de détection de rebond pour éviter les faux
contacts
196. unsigned long ledOnTime = 15000; // Temps (en millisecondes) avant d'éteindre la LED
si non réappuyé
197. unsigned long buttonPressTime = 0;
198. unsigned long ledTimer = 0;       // Timer pour éteindre la LED après un certain temps
199. bool ledState = false;          // État de la LED (allumée ou éteinte)
200. bool buttonState = false;        // État actuel du bouton
201. bool lastButtonState = false;    // Dernier état du bouton (pour détecter les
changements)
202.
203. // Position et dimensions du rectangle extérieur (blanc)
204. int PositionX = 20; // Position X
205. int PositionY = 130; // Position Y
206. int Largeur = 200; // Largeur
207. int Hauteur = 40; // Hauteur Hau
208. int arrondi = 10; // Rayon des coins arrondis
209. int Ep = 3; // Épaisseur
210.
211. // --- Affichage de la barre de batterie ---
212. int x_bat = 155; // Position X en haut à droite
213. int y_bat = 6; // Position Y en haut
214. int Lar_bat = 75; // Largeur
215. int Hau_bat = 34; // Hauteur
216. int arrondi_bat = 12; // Rayon des coins arrondis
217. int Ep_bat = 2; // Épaisseur
218.
219. float Lar_int = 0;
220. int x_int = 0;
221. int y_int = 0;
222. int Hau_int = 0;
223. int arrondi_inter = 0;
224.
225. void setup(void) {
226.
227. //-----Setup Serial-----//
228.
229. Serial.begin(9600);
230.
231. //-----Setup Oled-----//
232. pinMode(3, OUTPUT);
233. display.init(240, 240); // Initialisation pour un écran 240x240
234. display.setRotation(0); // Orientation de l'écran
235. display.fillScreen(ST77XX_BLACK); // Efface l'écran en noir
236. analogWrite(3, 0);
237. delay(100);
238. display.setCursor(10, 70);
239. display.setTextColor(CYAN);
240. display.setTextSize(3);
241. display.print("Stroke Watch");
242. display.setCursor(60, 105);
243. display.setTextSize(2);
244. display.setTextColor(VIOLET);
245. display.print("Since 2024");
246. for (int i=0; i<=255;i++){
247.     analogWrite(3, i);
248.     delay(14);
249. }

```

```

250. display.fillRect(0, 165, 240, 75, BLACK);
251. display.setCursor(46, 165);
252. display.setTextSize(2);
253. display.setTextColor(WHITE);
254. display.print("Chargement");
255. delay(350);
256.
257. //-----Setup RTC-----//
258. rtc.setup();
259. rtc.adjustRtc(F(__DATE__), F(__TIME__));
260.
261. //-----Setup NRF24L01-----//
262. if (!radio.begin()) {
263.     display.setCursor(50, 165);
264.     display.setTextSize(2);
265.     display.setTextColor(RED);
266.     display.print("Bluetooth Error");
267.     while (1) {} // hold in infinite loop
268.
269. }
270.
271. radio.setPALevel(RF24_PA_LOW);
272. radio.setPayloadSize(sizeof(IndiceMVBracelet)); // float datatype occupies 4 bytes
273. radio.openReadingPipe(1, address[0]);
274. radio.startListening(); // put radio in RX mode
275.
276. display.fillRect(0, 165, 240, 75, BLACK);
277. display.setCursor(50, 165);
278. display.setTextSize(2);
279. display.setTextColor(WHITE);
280. display.print("Bluetooth Ok");
281. delay(350);
282.
283. //-----Setup Accelerometre-----//
284. if (! mma.begin()) {
285.     display.fillRect(0, 165, 240, 75, BLACK);
286.     display.setCursor(30, 165);
287.     display.setTextSize(2);
288.     display.setTextColor(RED);
289.     display.print("Accelerometer Error");
290.     delay(350);
291.     while (1);
292. }
293. mma.setRange(MMA8451_RANGE_4_G);
294.
295. display.fillRect(0, 165, 240, 75, BLACK);
296. display.setCursor(30, 165);
297. display.setTextSize(2);
298. display.setTextColor(WHITE);
299. display.print("Accelerometer Ok");
300. delay(350);
301. display.fillRect(0, 165, 240, 75, BLACK);
302. display.setCursor(30, 165);
303. display.setTextSize(2);
304. display.setTextColor(WHITE);
305. display.print("Configuration Ok");
306. delay(500);
307.
308. display.fillScreen(ST77XX_BLACK);
309.
310. //-----Setup Bouton-----//
311. pinMode(2, INPUT);
312.
313. //-----Setup Affichage Ecran-----//

```

```

315.
316. // Dessiner le cadre extérieur blanc
317. for (int i = 0; i < Ep; i++) {
318.     display.drawRoundRect(PositionX + i, PositionY + i, Largeur - 2 * i, Hauteur - 2
* i, arrondi, ST77XX_WHITE);
319. }
320.
321. // Dessiner le cadre extérieur blanc pour la batterie
322. for (int i = 0; i < Ep_bat; i++) {
323.     display.drawRoundRect(x_bat + i, y_bat + i, Lar_bat - 2 * i, Hau_bat - 2 * i,
arrondi_bat, ST77XX_WHITE);
324. }
325.
326. display.drawBitmap(0,0,epd_bitmap_Icone_ordi_allum_ ,50,50,ST77XX_WHITE);
327. display.drawBitmap(50,0,epd_bitmap_Icone_bluetooth_allum_,50,50,ST77XX_WHITE);
328. display.drawBitmap(95,0,epd_bitmap_Icone_MEAB,50,50,0x780F);
329.
330. rtc.read();
331. rtcAntHour=rtc.hour;
332. rtcAntMin=rtc.minute;
333. display.setTextSize(3);
334. display.setCursor(80, 75);
335. display.setTextColor(WHITE);
336. display.print(rtc.hour);
337. display.setTextSize(3);
338. display.setCursor(130, 75);
339. display.setTextColor(WHITE);
340. display.print(rtc.minute);
341. display.setCursor(115, 75);
342. display.print(":");
343.
344. ValeurP = 0.5;
345. }
346.
347. void loop() {
348.
349. //-----Lecture de l'heure-----//
350.
351. rtc.read();
352.
353. if((rtc.hour)!=rtcAntHour){
354.     display.fillRect(79, 74, 40, 40, BLACK);
355.     display.setTextSize(3);
356.     display.setCursor(80, 75);
357.     display.setTextColor(WHITE);
358.     display.print(rtc.hour);
359.     rtcAntHour=rtc.hour;
360. }
361. if((rtc.minute)!=rtcAntMin){
362.     display.fillRect(129, 74, 40, 40, BLACK);
363.     display.setTextSize(3);
364.     display.setCursor(130, 75);
365.     display.setTextColor(WHITE);
366.     display.print(rtc.minute);
367.     rtcAntMin=rtc.minute;
368. }
369.
370. //-----Lecture des valeurs de l'accéléromètre-----//
371. mma.read();
372. if(abs(mma.x)>(x+200) || abs(mma.x)<(x-200)){
373.     x=abs(mma.x);
374.     x_final=x_final+(x/8312);
375.     IndiceMVMontre=IndiceMVMontre+ x_final;
376. }
377. if(abs(mma.y)>(y+200) || abs(mma.y)<(y-200)){

```

```

378.     y=abs(mma.y);
379.     y_final=y_final+(y/8312);
380.     IndiceMVMontre=IndiceMVMontre+ y_final;
381. }
382. if(abs(mma.z)>(z+200) || abs(mma.z)<(z-200)){
383.     z=abs(mma.z);
384.     z_final=z_final+(z/8312);
385.     IndiceMVMontre=IndiceMVMontre+ z_final;
386. }
387.
388. //-----Réception données bluetooth-----//
389. uint8_t pipe;
390. if (radio.available(&pipe)) {           // is there a payload? get the pipe
number that received it
391.     uint8_t bytes = radio.getPayloadSize(); // get the size of the payload
392.     radio.read(&IndiceMVBracelet, bytes);
393.     //Serial.println("Well Received");           // fetch payload from FIFO
394.     display.drawBitmap(50,0,epd_bitmap_Icone_bluetooth_allum_,50,50, WHITE);
395. }else{
396.     display.drawBitmap(50,0,epd_bitmap_Icone_bluetooth_allum_,50,50, RED);
397. }
398.
399. if((IndiceMVBracelet < IndiceMVMontre) && (IndiceMVBracelet < IndiceMVMontre - 600))
{
400.     ValeurP = 0.9;
401. }
402. if((IndiceMVBracelet < IndiceMVMontre) && (IndiceMVBracelet < IndiceMVMontre - 300))
{
403.     ValeurP = 0.7;
404. }
405. if((IndiceMVBracelet > IndiceMVMontre - 150) && (IndiceMVBracelet < IndiceMVMontre + 150)){
406.     ValeurP = 0.5;
407. }
408. if((IndiceMVBracelet > IndiceMVMontre) && (IndiceMVBracelet > IndiceMVMontre + 300))
{
409.     ValeurP = 0.3;
410. }
411. if((IndiceMVBracelet > IndiceMVMontre) && (IndiceMVBracelet > IndiceMVMontre + 600))
{
412.     ValeurP = 0.1;
413. }
414.
415. if(IndiceMVMontre > 4000000000 || IndiceMVBracelet > 4000000000){
416.     z_final =0;
417.     y_final =0;
418.     x_final =0;
419.     if(IndiceMVBracelet >= IndiceMVMontre){
420.         IndiceMVBracelet= abs(IndiceMVBracelet-IndiceMVMontre);
421.         IndiceMVMontre = 0;
422.     }else{
423.         IndiceMVMontre= abs(IndiceMVBracelet-IndiceMVMontre);
424.         IndiceMVBracelet =0;
425.     }
426. }
427.
428. /*
429. if(IndiceMVBracelet == 0){
430.     ValeurP=0;
431. }
432. if((IndiceMVMontre/IndiceMVBracelet)>1 && IndiceMVBracelet !=0){
433.     ValeurP=1;
434. }else{
435.     ValeurP=(IndiceMVMontre/IndiceMVBracelet);
436. }

```

```

437. /*
438. //-----Ecran-----
439. const char* message;
440.
441. if(ValeurP>(ValeurAntP+0.1) || ValeurP<(ValeurAntP-0.1)){
442.     display.fillRect(0, 180, 240, 75, BLACK);
443.     if(ValeurP<=0.2){
444.         message = "Bouge !";
445.         color = RED;
446.
447.     }
448.     if(ValeurP<=0.4 && ValeurP>=0.2){
449.         message = "Insuffisant";
450.         color = ST77XX_ORANGE;
451.
452.     }
453.     if(ValeurP<=0.6 && ValeurP>=0.4){
454.         message = "Encore";
455.         color = ST77XX_YELLOW;
456.
457.     }
458.     if(ValeurP<=0.8 && ValeurP>=0.6){
459.         message = "C'est bien";
460.         color = ST77XX_GREEN;
461.
462.     }
463.     if(ValeurP<=1 && ValeurP>=0.8){
464.         message = "Excellent";
465.         color = 0x03E0;
466.
467.     }
468.     display.fillRoundRect(x_int, y_int, Largeur-6, Hau_int, arrondi_inter, BLACK);
469.     ValeurAntP=ValeurP;
470.     int16_t text_x, text_y;
471.     uint16_t text_Lar, text_Hau;
472.     display.setTextSize(3);
473.     display.getTextBounds(message, 0, 0, &text_x, &text_y, &text_Lar, &text_Hau);
474.     int text_x_centre = (240 - text_Lar) / 2;
475.     int text_y_centre = 190;
476.     display.setCursor(text_x_centre, text_y_centre);
477.     display.setTextColor(ST77XX_WHITE);
478.     display.print(message);
479.
480.     Lar_int = Largeur * ValeurP;
481.     x_int = PositionX + Ep;
482.     y_int = PositionY + Ep;
483.     Hau_int = Hauteur - 2 * Ep;
484.     arrondi_inter = 5;
485.
486.     // Dessiner la barre d'activité
487.     display.fillRoundRect(x_int, y_int, Lar_int, Hau_int, arrondi_inter, color);
488.
489. }
490. //-----Bouton-----
491.
492. if (reading != lastButtonState) {
493.     lastButtonState = reading;
494.     if (reading == LOW) { // Si le bouton est appuyé (état LOW avec pull-up)
495.         buttonPressTime = millis(); // Sauvegarder le temps de l'appui

```

```

502.     if (!ledState) {
503.         // Si la LED est éteinte, allumer la LED
504.         analogWrite(3, 255);
505.         ledState = true;
506.         ledTimer = millis() + ledOnTime; // Démarrer le timer pour éteindre la LED après
un certain temps
507.     } else {
508.         // Si la LED est allumée et que le bouton est réappuyé, éteindre la LED
509.         analogWrite(3, 0);
510.         ledState = false;
511.     }
512. }
513. }
514. // Si la LED est allumée et que le temps imparti est écoulé, éteindre la LED
515. if (ledState && millis() > ledTimer) {
516.     analogWrite(3, 0);
517.     ledState = false;
518. }
519.
520. }

```

Code Processing de l'application :

Application.pde

```

1. import processing.serial.*;
2.
3. void setup() {
4.   size(1920,1080);
5.   setSurface();
6.   setBackground();
7.   setSerial();
8.   fill(0);
9.
10. }
11.
12.
13. void draw() {
14.   //drawSendButton();
15.   changeCursor();
16.
17.   myPort.bufferUntil(65);
18. }

```

Background.pde

```

1. void setBackground()
2. {
3.   final int SIZE_LOGOS = 100;
4.   final int MARGIN = 25;
5.   final int WIDTHPOLY = SIZE_LOGOS*228/160;
6.   PImage mea;
7.   PImage stroke;
8.   PImage poly;
9.   mea = loadImage("logoMEA.png");
10.  stroke = loadImage("SW_Logo_OG.png");
11.  poly = loadImage("logoPolytech.png");
12.
13.  background(0);
14.  image(mea, MARGIN, MARGIN, SIZE_LOGOS, SIZE_LOGOS);
15.  image(stroke, (width-3*SIZE_LOGOS)/2, MARGIN, 3*SIZE_LOGOS, 3*SIZE_LOGOS);
16.  image(poly, width-WIDTHPOLY-MARGIN, MARGIN, WIDTHPOLY, SIZE_LOGOS);
17. }
18.
19. void setSurface() {
20.   final String ICON = "SW_Logo_Nom.png";
21.   PImage icon;
22.   icon = loadImage("SW_Logo.png");

```

```

23.
24.   surface.setTitle("ApplaWatch");
25.   surface.setIcon(icon);
26. }
27.

```

GUI.pde

```

1. final float SendButtonX = 1920*0.75;
2. final float SendButtonY = 1080*0.25;
3. final int SendButtonWidth = 400;
4. final int SendButtonHeight = 50;
5.
6. void drawSendButton()
7. {
8.   fill(200, 100, 100);
9.   rect(SendButtonX, SendButtonY, SendButtonWidth, SendButtonHeight, 20);
10.  fill(20);
11.  textSize(40);
12.
13. //text("Send data", SendButtonX + 110, SendButtonY + 40);
14.  text("Button", SendButtonX + 110, SendButtonY + 40);
15. }
16.
17. void mousePressed()
18. {
19.   if (mouseX > SendButtonX && mouseX < SendButtonX + SendButtonWidth && mouseY > SendButtonY
&& mouseY < SendButtonY + SendButtonHeight)
20.   {
21.
22.   }
23. }
24.
25. void changeCursor()
26. {
27.   if (mouseX > SendButtonX && mouseX < SendButtonX + SendButtonWidth && mouseY > SendButtonY
&& mouseY < SendButtonY + SendButtonHeight)
28.   {
29.     cursor(HAND);
30.   }
31.   else cursor(ARROW);
32. }
33.
34. void debug_XY()
35. {
36.   if(mousePressed) {
37.     int x = mouseX;
38.     int y = mouseY;
39.     String str = "(" + str(x) + "," + str(y) + ")";
40.     fill(255);
41.     rect(x, y, 50, 15);
42.     fill(0);
43.     textSize(12);
44.     text(str, x, y + 12);
45.   }
46. }
47.

```

Serial.pde

```

1. import processing.serial.*;
2.
3. Serial myPort; // Create object from Serial class
4. String val; // Data received from the serial port
5. boolean color_switch = true;
6.
7. void setSerial()
8. {
9.   String portName = Serial.list()[1]; //change the 0 to a 1 or 2 etc. to match your port

```

```
10.  myPort = new Serial(this, portName, 9600);
11. }
12.
13. void handshake()
14. {
15.
16. }
17.
18. void serialEvent(Serial myPort)
19. {
20.   if (color_switch)
21.   {
22.     fill(255, 50, 50);
23.   }
24.   else
25.   {
26.     fill(0);
27.   }
28.   color_switch = !(color_switch);
29.   myPort.write(66);
30. }
31.
```

Code Python de l'application :

Application.py

```
1. import tkinter
2. # numpy as np
3. import matplotlib.pyplot as plt
4. from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
5.
6. import graphFunctions as gf
7. import guiFunctions as GUI
8. import dataFunctions as df
9.
10. class LabeledEntry(tkinter.Entry):
11.     def __init__(self, master=None, label="Search", **kwargs):
12.         self.string_var = tkinter.StringVar()
13.         tkinter.Entry.__init__(self, master, textvariable=self.string_var, **kwargs)
14.         self.label = label
15.         self.on_exit()
16.         self.bind('<FocusIn>', self.on_entry)
17.         self.bind('<FocusOut>', self.on_exit)
18.
19.     def on_entry(self, event=None):
20.         if self.get() == self.label:
21.             self.delete(0, tkinter.END)
22.             self.configure(fg="black")
23.
24.     def on_exit(self, event=None):
25.         if not self.get():
26.             self.configure(fg="grey")
27.             self.insert(0, self.label)
28.
29. root = tkinter.Tk()
30. GUI.frame_configuration(root)
31.
32. frm_logo = tkinter.Frame(root, height=162, width=288)
33. frm_logo.grid(column=0, row=0)
34. frm_logo.pack_propagate(0)
35.
36. frm_hpanel = tkinter.Frame(root, height=162, width=1248)
37. frm_hpanel.grid(column=1, row=0)
38. frm_hpanel.grid_propagate(0)
39.
40. frm_vpanel = tkinter.Frame(root, height=702, width=288)
41. frm_vpanel.grid(column=0, row=1)
42. frm_vpanel.pack_propagate(0)
43.
44. frm_curve = tkinter.Frame(root, height=702, width=1248)
45. frm_curve.grid(column=1, row=1)
46. frm_curve.pack_propagate(0)
47.
48. #####
49. # Logo Panel
50. #####
51. img_logo = tkinter.PhotoImage(file="AW_Logo.png")
52. logo = tkinter.Label(frm_logo, image=img_logo).pack(padx=30)
53. name = tkinter.Label(frm_logo, text="ApplaWatch", font="50").pack(padx=30)
54.
55.
56. #####
57. # Top Panel
58. #####
59. GUI.frame_configuration(frm_hpanel, frame="hpanel")
60.
61. tkinter.Label(frm_hpanel, text="Doctor's mail : ", font="30").grid(row=0, column=0,
sticky="e")
62. tkinter.Label(frm_hpanel, text="Your mail : ", font="30").grid(row=1, column=0, sticky="e")
63.
64. entry_doctor = LabeledEntry(master=frm_hpanel, label="Enter Doctor's mail address here",
font="30", width=50)
65. entry_doctor.grid(row=0, column=1, sticky="w")
```

```

66.
67. entry_sender = LabeledEntry(master=frm_hpanel, label="Enter your mail address here",
font="30", width=50)
68. entry_sender.grid(row=1, column=1, sticky="w")
69.
70. data_console_var = tkinter.StringVar()
71. data_console_var.set("")
72.
73. send_button = tkinter.Button(frm_hpanel, text="Send Data", command= lambda:
GUI.send_data(entry_doctor, entry_sender, data_console_var))
74. send_button.configure(font="50", cursor='hand2')
75. send_button.grid(row=0, column=2, sticky="w")
76.
77. data_console = tkinter.Label(master=frm_hpanel, textvariable=data_console_var)
78. data_console.grid(row=1, column=2, sticky="w")
79.
80. #####
81. # Vertical Panel          #
82. #####
83. img_watch = tkinter.PhotoImage(file="Watch.png")
84. tkinter.Label(frm_vpanel, image=img_watch).pack(padx=30, pady=20)
85.
86. tkinter.Button(frm_vpanel, text="Quit", font="50", command=root.destroy,
cursor='hand2').pack(padx=30, pady=20)
87.
88.
89. #####
90. # Curve Panel           #
91. #####
92. fig, ax = plt.subplots()
93.
94. canvas = FigureCanvasTkAgg(fig, master=frm_curve)
95. canvas.get_tk_widget().pack(fill=tkinter.BOTH)
96. canvas.get_tk_widget().configure(width=1248, height=620)
97.
98. toolbar = NavigationToolbar2Tk(canvas, frm_curve, pack_toolbar = False)
99. toolbar.update()
100. toolbar.pack(anchor="w", padx=20, fill = tkinter.X)
101.
102. frm_curve_panel = tkinter.Frame(frm_curve)
103. frm_curve_panel.pack()
104.
105. button1 = tkinter.Button(frm_curve_panel, text="Last day", font="50", command= lambda:
gf.plot_day(canvas, fig, ax), cursor='hand2')
106. button1.pack(side=tkinter.LEFT, padx=10)
107. button2 = tkinter.Button(frm_curve_panel, text="Last month", font="50", command= lambda:
gf.plot_month(canvas, fig, ax), cursor='hand2')
108. button2.pack(side=tkinter.LEFT, padx=10)
109.
110. gf.plot_day(canvas, fig, ax)
111.
112. root.mainloop()
113.

```

dataFunctions.py

```

1. import numpy as np
2.
3.
4. def search_filename(directory, mode):
5.     with open(directory + "data_config.txt") as file:
6.         for line in file:
7.             config = line.strip().split('=')
8.             if(mode == "month" and config[0] == "last_month"):
9.                 month = config[1]
10.                filename = month + "\\" + month + "_mean.txt"
11.            elif(mode == "day" and config[0] == "last_day"):
12.                day = config[1]
13.                filename = day[:4] + "\\" + day + ".txt"

```

```

14.     return filename
15.
16. def read_from_file(mode="day", path=".\\Data\\"):
17.     filename = search_filename(path, mode)
18.     print(filename)
19.     x = []
20.     y = []
21.     with open(path + filename, 'r') as file:
22.         for line in file:
23.             values = line.strip().split(',')
24.             x.append(float(values[1]))
25.             y.append(float(values[0]))
26.
27.     a = np.array(x)
28.     b = np.array(y)
29.     if (mode == "day"):
30.         text = filename[5:-4]
31.     else:
32.         text = filename[:4]
33.     print(text)
34.     return a, b, text
35.

```

graphFunctions.py

```

1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4. import dataFunctions as df
5.
6. def plot_graph(fig, ax, x, y, text, x_unit):
7.     ax.clear()
8.     ax.plot(x, y)
9.
10.    plt.title("Données du " + text)
11.    plt.xlabel("Time " + x_unit)
12.    plt.ylabel("Movement indicator")
13.
14.
15. def plot_day(canvas, fig, ax):
16.     x, y, day = df.read_from_file("day")
17.     text = str(day[5:6]) + "/" + str(day[3:4]) + "/20" + str(day[:2])
18.     plot_graph(fig, ax, x, y, text, "(h)")
19.     canvas.draw()
20.
21. def plot_month(canvas, fig, ax):
22.     x, y, month = df.read_from_file("month")
23.     text = str(month[3:4]) + "/20" + str(month[:2])
24.     plot_graph(fig, ax, x, y, text, "(day)")
25.     canvas.draw()
26.

```

guiFunctions.py

```

1. def normalScreen(event, root):
2.     root.attributes("-fullscreen", False)
3.
4. def frame_configuration(parent, frame="root"):
5.     if(frame == "root"):
6.         parent.title("ApplaWatch")
7.         parent.geometry('1575x900')
8.         parent.resizable(False, False)
9.         parent.configure(bg="Black")
10.        # root.attributes("-fullscreen", True)
11.        parent.bind("<Escape>", lambda event: normalScreen(event, parent))
12.
13.        parent.grid_rowconfigure(0, weight=2)
14.        parent.grid_rowconfigure(1, weight=8)
15.        parent.grid_columnconfigure(0, weight=2)

```

```

16.         parent.grid_columnconfigure(1, weight=8)
17.
18.     elif(frame == "hpanel"):
19.         parent.grid_rowconfigure(0, weight=1)
20.         parent.grid_rowconfigure(1, weight=1)
21.         parent.grid_columnconfigure(0, weight=1)
22.         parent.grid_columnconfigure(1, weight=5)
23.         parent.grid_columnconfigure(2, weight=4)
24.
25. def send_data(entry_doctor, entry_sender, console):
26.     print("Send button pressed")
27.     if (entry_doctor.get() == entry_doctor.label):
28.         console.set("You must enter your Doctor's mail")
29.     elif (entry_sender.get() == entry_sender.label):
30.         console.set("You must enter your mail")
31.

```

SerialFunctions.py

```

1. import serial
2.
3. def get_port(ser):
4.     for i in range(0, 15):
5.         ser.port = "COM" + str(i)
6.         ser.write(b"SYN\n")
7.         ack = ser.readline().decode('utf-8')
8.         if (ack == "ACK\n"):
9.             port = "COM" + str(i)
10.            return port
11.
12. ser = serial.Serial()
13. ser.baudrate = 9600 #Bd
14. ser.timeout = 1 #s
15. port = get_port(ser)
16.

```

Diagramme de Gantt détaillé du projet :

