Institut national de recherche en informatique et en automatique

Grenoble INP - Phelma Ensimag

Rapport de stage

Contrôle d'environnement de fauteuil roulant électrique.

Brice ARNAUD

Encadrants : Christophe BRAILLON
Roger PISSARD

9 Mai 2022 - 29 Juillet 2022

Table des matières

Table des figures

Glossaire

CAN Controller Area Network, bus de communication série utilisé en électronique et automobile. 2, 6, 7, 11, 14, 16–18

JSM Joystick Module, joystick d'assistance propriétaire communiquant avec le moteur à l'aide du protocole R-net. 2, 7, 11, 14–17, 20, 21

R-net Protocole de communication propriétaire utilisée par le JSM et le moteur du fauteuil. 6, 7, 11, 14, 15, 18, 19, 21

Raspberry Pi Nano-ordinateur monocarte à processeur ARM de la taille d'une carte de crédit. 2, 6, 7, 9, 10, 13, 14, 16–18, 22

Remerciements

Je souhaite tout d'abord remercier Christophe Braillon et Roger Pissard pour m'avoir choisi pour ce stage. Merci à eux pour l'encadrement qu'ils nous ont fourni. Je me suis senti à la fois libre et épaulé. Ils sont deux inspirations pour moi.

Je souhaite aussi remercier les membres de l'équipe SED. Merci à Sarah de m'avoir accueilli au forum STEEL de l'Ensimag. Merci à Frédéric et Roxanne pour leur bonne humeur. Merci à Samuel pour les activités extra-professionnelles.

Enfin, merci aux services administratifs de Phelma pour nous avoir simplifié les démarches nécessaires.

Chapitre 1

Contexte

1.1 Structure d'accueil : L'Inria

Lors de mon stage, j'ai été accueilli à l'Inria, l'institut national de recherche en informatique et en automatique. Il s'agit d'un établissement public à caractère scientifique et technologique, présidé par M. Bruno Sportisse. Il existe une dizaine de centres répartis sur toute la France, mais j'ai travaillé dans le centre de recherche Inria de l'Université Grenoble Alpes situé à Inovallée/Montbonnot.

Depuis sa création il y a 50 ans, l'Inria a pour mission le développement de la recherche et de la valorisation en sciences et techniques de l'information et de la communication, au niveau national comme au plan international. Aujourd'hui, l'institut continue d'accompagner la transformation numérique de la science, de l'économie et de la société. L'Inria et ses 3 900 chercheurs et ingénieurs produisent ainsi des connaissances dans des domaines tels que l'intelligence artificielle, la science des données ou même la robotique.
L'Inria se caractérise aussi par son engagement dans les programmes européens tels que *Horizon Europe*. On dénombre 180 projets dans le cadre de ce programme, ce qui fait de l'Inria un acteur français majeur dans la recherche et de l'innovation à l'échelle européenne.

L'Inria soutient aussi d'autres voies d'innovation tel que la création de start-ups technologiques. Ces 20 dernières années, ce sont 170 start-ups qui ont été soutenues par *Inria Start-up Studio*.

Le fonctionnement de l'Inria s'articule autour d'équipes-projets souvent en collaboration avec d'autres organismes d'enseignement et de recherche. Les chercheurs et ingénieurs coopèrent pour faire avancer leurs projets. Certains d'entre eux sont propices à la création de stage, comme celui dans lequel ma collègue étudiante Roxanne Xu et moi sommes impliqués.

1.2 Équipe intégrée

Durant ces trois mois, c'est l'équipe SED de l'Inria de Grenoble qui m'a accueilli. Il s'agit du Service Expérimentation et Développement qui a pour but de soutenir les équipes-projets pour le développement logiciel et la mise en place de plateformes. Les ingénieurs de recherche du service ont accès à des ressources informatiques et mécaniques telles que des imprimantes 3D et des découpeuses laser.

Durant ce stage, j'ai été accueilli et encadré par deux de ces ingénieurs de recherche Inria, Christophe Braillon et Roger Pissard, que je vais maintenant présenter.

1.2.1 Christophe Braillon

Christophe Braillon est un ancien diplômé de l'Ensimag. Il y donne des cours en parallèle de sa thèse pendant quelques années avant d'intégrer le service SED en tant qu'ingénieur de recherche. Entre 2011 et 2017, il lance une start-up soutenue par l'Inria : initialement nommée *HikoB*. Après cela, il retourne à plein temps au SED et en devient le chef en avril 2021.

1.2.2 Roger Pissard

Roger Pissard rejoint l'Inria après sa thèse en 1993. En tant qu'ingénieur de recherche, il se spécialise en robotique, systèmes embarqués, réseaux de capteurs. Durant ces années, il co-fonde deux start-ups incubées à l'Inria. La première est *Athys* entre 1999 et 2003, qui a été rachetée par *Dassaut Systèmes*. La seconde est *mars :hello !* entre 2017 et 2019.

Ces centres d'intérêts professionnels, aujourd'hui, sont la conception et la création de logiciels et dispositifs d'aide au handicap.

1.3 Le projet

Avec ma collègue Roxanne Xu, nous avons travaillé pendant trois mois sur le projet nommé **Magic Joystick**. Pour détailler le projet, il me faut d'abord introduire *My Human Kit*.

1.3.1 My Human Kit

My Human Kit (ou MHK) est une association qui invite les personnes en situation de handicap à devenir porteur de projet. Ces personnes viennent avec un besoin et des idées, des propositions qui répondent à ce besoin. *My Human Kit* a alors pour objectif de fabriquer des aides techniques pour et avec le porteur de projet. Ces aides peuvent ne pas exister sur le marché, mais elles peuvent aussi être bien trop coûteuses ou même inesthétiques. *My Human Kit* propose alors des alternatives.

My Human Kit est à l'initiative de la création d'un réseau d'associations, nommé Humanlab, qui regroupe les associations qui proposent en France ce type de services. On peut citer le Humanlab Saint-Pierre à Palavas ou Autonabee à Lyon.
L'Inria apporte un soutien technique à ce réseau à travers son action exploratoire Humanlab-Inria.

L'association dispose de plusieurs Fablabs. Ce sont des ateliers de fabrication numérique, des lieux de créativité et d'entraide où sont réalisés des prototypes. De plus, les porteurs de projet peuvent profiter de l'expertise des membres de l'association pour en apprendre sur l'informatique, la robotique ou l'électronique. *My Human Kit* et *Humanlab* sont aussi une aide privilégiée de sociabilité pour ces personnes en situation de handicap.

Il est important de préciser que l'intégralité des projets sont documentés et en libre accès sur leur Wikilab. Le projet sur lequel j'ai travaillé ne fait pas exception.

1.3.2 Magic Joystic

Parmi les nombreux projets que porte l'association, il y en a un en collaboration avec l'Inria qui a commencé en 2019 : **Magic Joystick**. Le porteur du projet s'appelle Jonathan Menir. Il est le coprésident de *My Human Kit* et a été l'un des premiers à fréquenter le Humanlab. Jonathan souffre du handicap de tétraplégie

qui l'immobilise quasi entièrement. Il se déplace à l'aide d'un fauteuil roulant électrique, piloté par un joystick.

Le handicap de Jonathan est sévère, ce qui fais que les joysticks classique que l'on trouve sur la plupart des fauteuils ne conviennent pas à sa mobilité. Ils sont trop durs à bouger. De plus, les boutons qui contrôlent son environnement (phares, assise, vitesse...) sont trop éloignés de son doigt.

Le but du projet **Magic Joystick** est donc de produire un joystick faible force qui puisse remplacer un joystick classique afin que Jonathan retrouve un peu de son autonomie.

Il existe déjà des joysticks miniatures qui pourraient convenir à Jonathan. Cependant, ils sont extrêmement coûteux et ne permettent pas une interaction complète avec l'environnement de vie. Par exemple, il serait impossible de contrôler un téléphone portable. Il serait aussi nécessaire d'investir beaucoup pour pouvoir contrôler des volets ou une télévision en infrarouge directement avec le joystick.

1.3.3 Historique et enjeux

Ce projet a déjà plusieurs années d'existence. Un premier prototype a été réalisé en octobre 2019 lors d'un Fabrikarium chez *ArianeGroup*. Les participants ont notamment réussi à bouger le fauteuil à l'aide d'une manette de jeu vidéo.

La principale difficulté qui s'est présentée est que le protocole de communication entre le joystick initial et le moteur du fauteuil est propriétaire. Il se nomme **R-net** et les trames d'échange qui circulent sur un bus CAN ne sont pas documentées au grand public. Il a donc fallu faire de la rétro-ingénierie sur le protocole **R-net** afin de comprendre et de reproduire certaines trames de contrôle.

À la fin de ces quelques jours de Fabrikarium, les participants ont pu contrôler le fauteuil avec une manette de jeu vidéo Xbox.

L'année suivante lors d'un Hackathon, le prototype de 2019 a été repris, amélioré et interfacé avec le fauteuil de Jonathan. Pour ce faire, un micro-ordinateur Raspberry Pi a été configuré et utilisé. Il a donc été possible de bouger le fauteuil électrique avec le joystick faible force imprimé en 3D.

En 2021, le Hackathon organisé avait pour but d'élargir l'utilisation du joystick et de la Raspberry Pi pour offrir un contrôle d'environnement modulable (volets ouvrants, télévision, smart phone...). C'est à partir de cette année que le projet sera aussi connu sous le nom de *Magic Control*, afin de prendre en compte ces avancées liées à la domotique.

Mon stage reprend les éléments de ces dernières années. Et il nous a été demandé d'imaginer et de développer des améliorations pour ce projet, que je vais maintenant détailler.

Dans ce rapport, je vais d'abord présenter l'architecture matérielle et logicielle du système. Ensuite je détaillerai les développements logiciels que j'ai réalisés autour de l'accès Wi-Fi, le protocole R-net et la calibration du joystick. Je terminerai par un bilan de mon travail et de mes impressions sur ce stage.

Chapitre 2

Architecture du projet

2.1 Architecture physique

Afin de comprendre les travaux que j'ai réalisés pendant ces trois mois de stages, il me faut expliquer comment fonctionnent entre eux les différents équipements. La figure 2.1 montre l'architecture matérielle du projet **MagicJoystick**.

Sur cette figure, on identifie notamment l'ancien joystick d'assistance nommé JSM (JoyStick Module) qui communiquait initialement directement avec le moteur. Le bus utilisé est le CAN, ou *Controller Area Network*. Il s'agit d'un bus de communication série utilisé en électronique, et particulièrement dans l'industrie automobile. Il fonctionne sur le principe du multiplexage, où chaque équipement connecté communique avec tous les autres.

Sur ce bus circulent des trames qui suivent un protocole propriétaire du fabricant du fauteuil, le protocole **R-net**. Ces trames peuvent par exemple représenter la position actuelle du joystick, le niveau de batterie restant, ou même l'instruction d'extinction du moteur.

On observe que le micro-ordinateur Raspberry Pi (RPi) est en "coupure" sur le bus CAN, entre le moteur et le JSM. Le câble a été coupé, dénudé et connecté à une carte *PiCAN2*, elle-même branchée aux pins d'entrée-sortie de la Raspberry Pi. Le Rpi est donc en position de relayer les messages R-net en les modifiant ou pas. Cette position stratégique permet aussi de capturer les échanges entre JSM et moteur afin de les comprendre par rétro-ingénierie. Une fois certaines trames identifiées, notamment celles qui donnent la position du joystick, il sera possible de modifier les données de ces trames R-net pour envoyer non pas les positions du JSM, mais de notre joystick faible force.

Notre joystick faible force est branché sur les pins d'entrée-sortie de la Raspberry Pi. La figure 2.2 montre une représentation 3D des pièces imprimées qui le composent. On peut y voir notamment le bras principal en gris, ainsi que les vis en noir permettant de régler la force requise pour bouger le joystick. Le joystick revient en position neutre grâce à de petits aimants.

Pour récupérer la position du joystick faible force, un aimant est placé au bout du bras. Il profite ainsi d'un effet de levier. Dans la cavité striée vient se glisser un socle avec un capteur de positionnement *MLX90333*. Celui-ci est relié à un convertisseur analogique-numérique 12 bits *ADS1015*, puis à la Raspberry Pi grâce à un bus I2C comme décrit sur la figure 2.3. Ainsi, il est possible de lire en temps réel la position de l'aimant.

Un clic faible force est positionné sous le pouce de Jonathan. Il est relié à l'un des pins d'entrée-sortie de la Raspberry Pi. Son utilité sera développée plus tard.

Enfin, il existe sur la Raspberry Pi deux interfaces qui vont nous être utiles pour contrôler l'environnement du fauteuil. Il s'agit des interfaces Wi-Fi et Bluetooth.

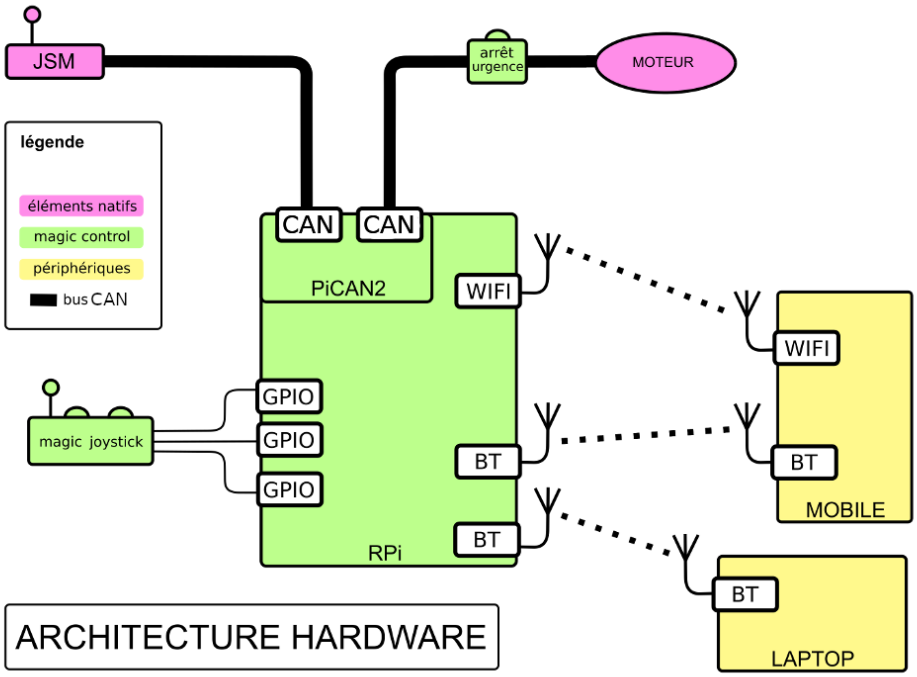


FIGURE 2.1 – Architecture matérielle du projet MagicJoystick

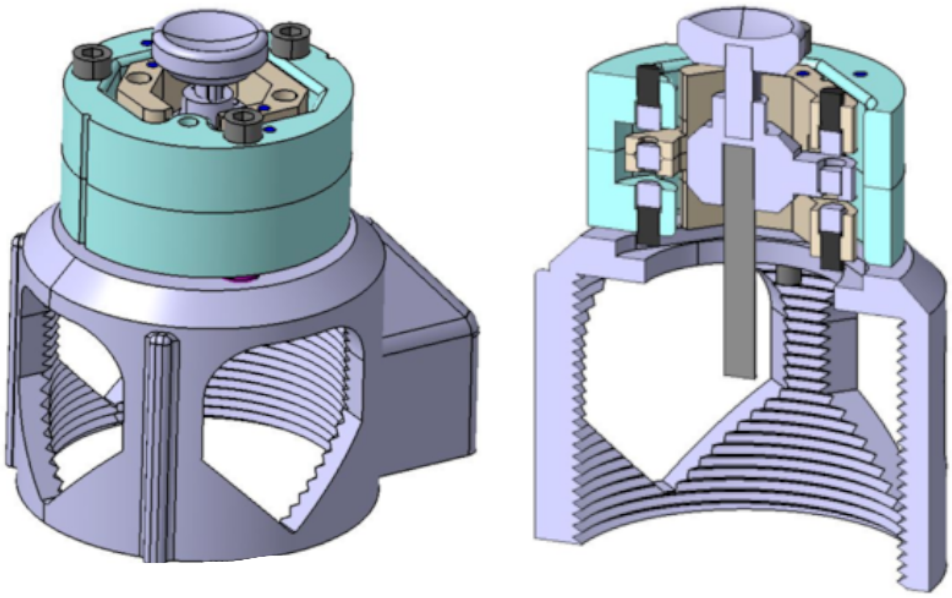


FIGURE 2.2 – Représentation 3D du joystick faible force sans capteur de position

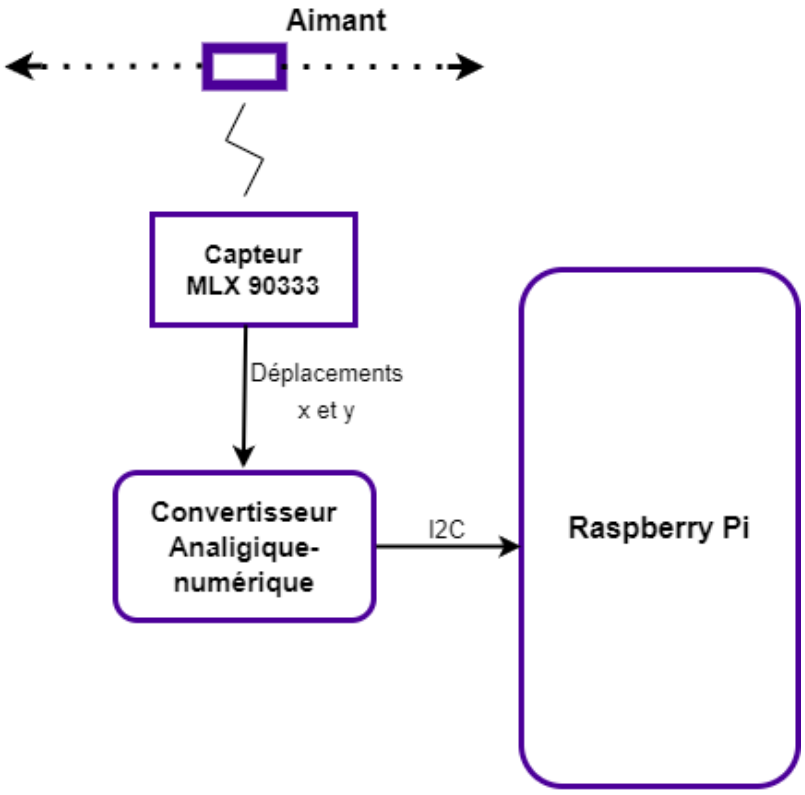


FIGURE 2.3 – Schéma bloc du circuit de récupération des positions du joystick faible force

L'interface Bluetooth a pour but principal l'interaction avec le téléphone portable de Jonathan. Il doit être en mesure d'utiliser son portable via le joystick sous son doigt. L'un des buts du stage est de faire émuler à la Raspberry Pi une souris Bluetooth, de la connecter à un smartphone afin de transmettre les mouvements faits au joystick sur le téléphone. Il faut aussi développer une façon ergonomique de passer du contrôle du fauteuil au contrôle de la souris sur le téléphone.

L'interface Wi-Fi a pour but d'être configuré de façon à ce que la Raspberry Pi agisse comme un point d'accès Wi-Fi. Le but est de faire tourner un serveur HTTP, et que les périphériques (essentiellement le téléphone) puisse s'y connecter et interagir avec le fauteuil. Je vais détailler le fonctionnement de ce serveur HTTP dans la section suivante : l'architecture logicielle.

2.2 Architecture logicielle

La figure 2.4 résume l'architecture logicielle du projet. Détaillons le rôle de tous les services.

Le service central est un agent de communication inter-service basé sur le protocole MQTT. Ce dernier est basé sur le protocole TCP/IP. Les clients, ici nos services, peuvent s'abonner à une liste de sujets prédéfinis. Lorsque l'un d'entre eux publie un message sur un sujet, l'information est relayée par l'agent à tous les clients qui s'y sont abonnés. Par exemple, quand le service *Joystick* publie un message concernant un clic, le service *Bluetooth* qui y est abonné reçoit les données et peut agir en conséquence de son côté en émulant un clic de souris bluetooth. L'agent utilisé est *Mosquitto*. Il est configuré pour être lancé au démarrage de la Raspberry.

Le service nommé *Joystick* a pour but de lire les données produites par le convertisseur analogique-

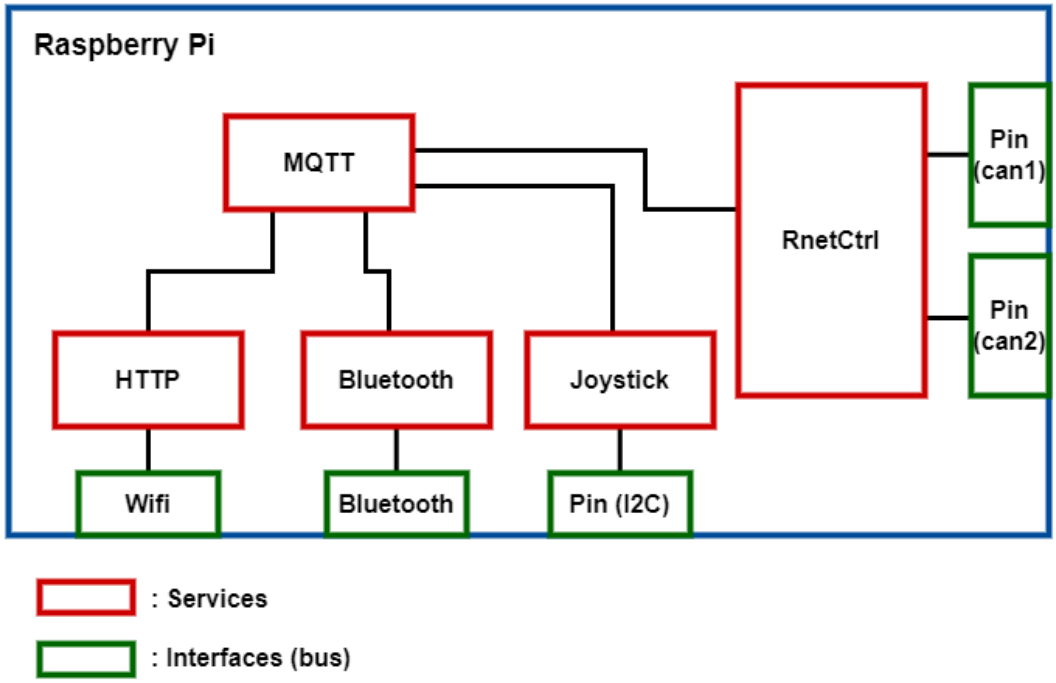


FIGURE 2.4 – Schéma bloc de l'organisation et de la communication des services sur la Raspberry Pi

numérique, ainsi que les clics du joystick faible force. Ces données sont traitées puis publiées pour les autres services. Il est écrit en Python et se sert de bibliothèques pour interagir avec les pins d'entrée-sortie de la Raspberry Pi.

Le service Bluetooth aura pour rôle d'émuler une souris Bluetooth afin de permettre le contrôle complet d'un téléphone portable avec le joystick faible force. Il devra être abonné aux informations venant du service *Joystick* afin de bouger la souris en conséquence. Il sera écrit lors de ce stage en Python.

Le service nommé *HTTP* est écrit en Python à l'aide de *Flask*. Il s'agit d'un utilitaire Python permettant de construire rapidement des applications web. Voici un exemple d'application minimale qui ouvre une page avec le texte "Hello, World!" lorsque on se connecte sur l'IP de la machine, sur le port de l'application.

```
1 #Application minimale avec Flask
2 from flask import Flask
3
4 app = Flask(__name__)
5
6 @app.route("/")
7 def hello_world():
8     return "<p>Hello, World!</p>"
```

L'application a évolué au fil du stage afin d'offrir une interface pour les fonctionnalités du projet. La figure 2.5 montre à gauche la page d'accueil de l'application. On y trouve un bouton pour rejoindre la page de contrôle du fauteuil, à droite sur la figure. Sur celle-ci on observe par exemple que les phares et les feux de détresse du fauteuil sont actuellement allumés, mais que les clignotants sont éteints. On voit aussi que l'indicateur de conduite est vert, ce qui veut dire que le joystick faible force contrôle le fauteuil, et non pas

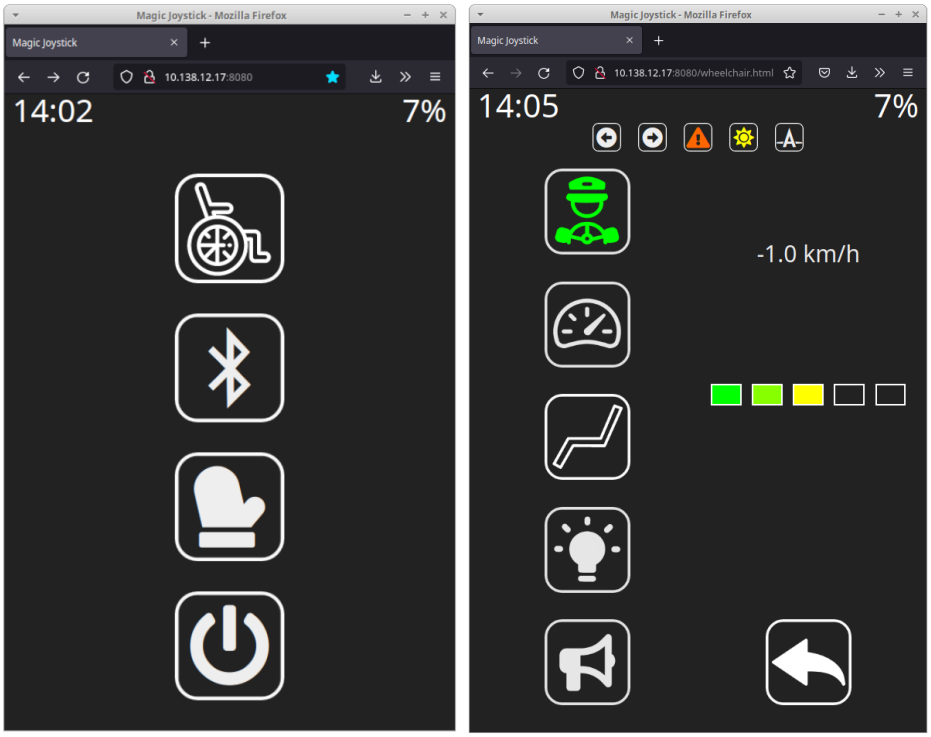


FIGURE 2.5 – Captures d'écran de la page d'accueil (gauche) et de la page de contrôle (droite) de l'application web.

la souris sur le téléphone.

Enfin, on voit sur la figure 2.4 le service nommé *RnetCtrl*. Il s'agit d'un code Python qui s'occupe des interfaces CAN et du trafic de trames R-net. Lorsque le fauteuil est en état de marche, ce service relaie les trames dans un sens comme dans l'autre à l'exception des trames de position de joystick venant du JSM. Ces dernières sont remplacées par nos propres trames, avec la position du joystick faible force.

C'est aussi le service *RnetCtrl* qui va communiquer aux autres les informations venant du moteur. On peut citer le niveau de batterie du fauteuil, la vitesse actuelle, ou même le kilométrage.

Prenons maintenant un cas de figure. L'utilisateur clique avec la souris positionnée sur le bouton des phares dans l'interface web de son téléphone. Le clic est capturé par le service *Joystick* et communiqué au service *Bluetooth* qui y est abonné. Celui-ci effectue un clic sur la souris qu'il émule. Le téléphone (connecté en bluetooth) reçoit l'information et le clic est effectué sur le navigateur. L'application web (service *HTTP*) reçoit l'information au travers de la connexion Wi-Fi et publie un message de changement d'état des phares. Ce message n'est relayé qu'au service *RnetCtrl*, car il est le seul à y être abonné. Il crée ensuite la trame R-net qui allume les phares (ou celle qui les éteindrait s'ils étaient déjà allumés).

Enfin, le contrôle et la surveillance de tous ces services sont mutualisés dans un seul d'entre eux. Nous utilisons *Supervisor*, un système de contrôle de processus. Il fonctionne avec un fichier de configuration dans lequel on décrit tous les services, leurs noms, leurs fichiers source, la commande à lancer pour les exécuter, et plus encore. De son côté, il offre une interface web dont voici une capture d'écran, figure 2.6. On peut y voir l'état de tous les processus suivi. Par exemple, le processus *ble_hid_mouse* a subit une erreur critique. On peut consulter directement sur cette interface les sorties standard des processus en temps réel.

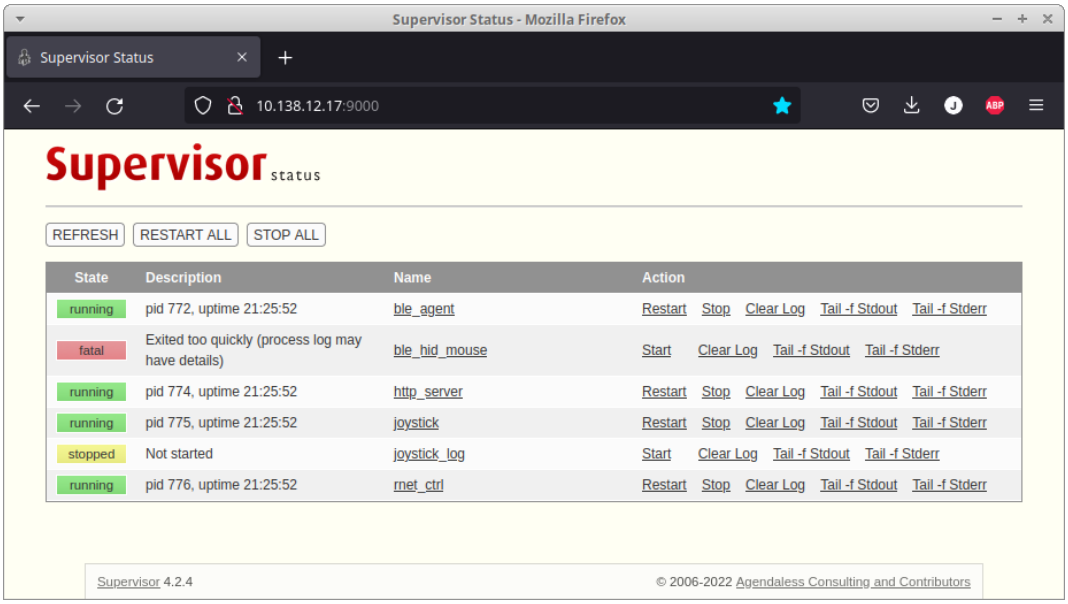


FIGURE 2.6 – Capture d'écran de l'interface web qu'offre *Supervisor*

Chapitre 3

Réalisations

Je vais maintenant décrire en détail mes travaux sur ce projet durant ce stage.

Pendant les premiers jours, j'ai exploré les différentes parties du projet. Une fois à l'aise avec la théorie, il a fallu passer à la pratique en installant sur une Raspberry Pi l'intégralité du projet afin de l'observer en marche. La Raspberry Pi fonctionne avec le système d'exploitation *Raspbian*, un dérivé de *Debian* conçu pour cette carte. Il a ensuite fallu configurer l'interface Wi-Fi afin que la carte agisse comme un point d'accès Wi-Fi.

3.1 Point d'accès Wi-Fi

3.1.1 Pourquoi un point d'accès Wi-Fi ?

Tout d'abord, ce point d'accès permet à un appareil de s'y connecter. Ainsi, un utilisateur à portée du fauteuil pourra ouvrir une connections SSH passant par Wi-Fi depuis n'importe où. Ceci permet par exemple de relancer un processus, de relancer la Raspberry, ou même d'avoir accès à des fichiers de log pour tracer d'éventuels bugs. Comme le fauteuil est un élément mobile, ne pas avoir à le brancher sur un réseau pour le reconfigurer ou le réparer est un point primordial.

Le nouveau réseau local formé par le point d'accès permet l'utilisation d'applications qui s'exécutent sur la Raspberry Pi. C'est le cas de notre serveur web. Une fois connecté au réseau, un appareil a seulement besoin d'un navigateur pour utiliser la page de contrôle du fauteuil. La sécurité est assurée par le mot de passe à la connexion sur le réseau.

3.1.2 Création du point d'accès

Pour faire de la Raspberry Pi un point d'accès Wi-Fi, nous avons suivi sa documentation et ses spécifications en utilisant des utilitaires préexistants. **Hostapd** est un daemon qui remplie cette fonction. Il est configurable à l'aide d'un fichier de configuration. On y trouve par exemple le nom qu'aura le futur point d'accès, le nom de l'interface que l'on souhaite utiliser (la Raspberry n'en a qu'une), ou le nombre maximum d'appareils qui peuvent s'y connecter. Un élément de configuration important est le mot de passe du point d'accès. Étant donné que la Raspberry a le contrôle du fauteuil, seul l'utilisateur final doit pouvoir s'y connecter.

Hostapd seul ne suffit pas à faire fonctionner le réseau. En effet, il n'implémente pas le protocole DHCP. Le protocole DHCP, ou *Dynamic Host Configuration Protocol* est un protocole réseau dont le rôle est d'assurer la configuration automatique des paramètres IP d'une machine, notamment en lui attribuant

automatiquement une adresse IP et un masque de sous-réseau. DHCP peut aussi configurer l'adresse de la passerelle par défaut (ici notre Raspberry Pi).

C'est le daemon **Dnsmasq** qui a été choisis pour réaliser cette tache. Il s'agit d'un serveur léger conçu pour des petits réseaux. Tout comme **Hostapd**, il propose un fichier de configuration dans lequel on peut spécifier les caractéristiques de son réseau. Nous y avons indiqué entre autre la même interface utilisée par **Hostapd**, ainsi que la plage d'adressage IP et le masque du réseau.

3.1.3 Ce qu'il reste à faire

Habituellement, faire un tel point d'accès permet à des appareils sans fil de se connecter à Internet via la plate-forme, qui elle est branchée physiquement à Internet. Cependant, notre Raspberry Pi est vouée à n'avoir aucun accès Internet. Les appareils qui y sont connectés via le Wi-Fi ont certes accès au contrôle du fauteuil, mais plus à Internet. C'est d'ailleurs pourquoi aucun relai de paquets IP n'est pas configuré pour notre point d'accès. Nous n'avons pas eu le temps de chercher une solution à ce problème.

On doit pouvoir trouver des pistes de solution dans la configuration du point d'accès, en indiquant clairement que ce dernier n'offre aucun accès Internet aux appareils qui s'y connectent. Une autre piste serait de configurer directement ces appareils en choisissant une autre source Internet, comme la 4G sur un téléphone.

3.2 Protocole R-net

Cette partie est dédiée aux détails protocole R-net et aux avancées faites concernant le contrôle du fauteuil. Je présenterai à la fin les améliorations envisagées avec l'équipe sur ces aspects.

3.2.1 Communication R-net

Notre but est de prendre le contrôle du fauteuil. Pour cela, il faut être capable de comprendre et répliquer certaines trames R-net clé. Regardons comment est constituée une trame.

Une trame R-net est formée de 16 octets. La figure 3.1 montre une trame de position de joystick envoyé par le JSM. Les huit premiers octets représentent le type de trame, le reste représente les données.

L'interprétation des huit premiers bit est complexe et incertaine. On interprète les huit premiers bits comme une source. Dans la figure 3.1, l'identifiant de la source (le JSM) est 0x0013. Cependant, cette valeur n'est valable que pour ce type de trames. Une trame envoyée par le JSM qui ne représente pas une position de joystick pourra voir ses huit premiers bits différés. Cependant, on retrouve à coup sûr un code représentant l'action décrite par la trame. Pour la position de joystick, cette valeur est 0x0200.

On peut voir sur la figure les valeurs de X et Y sur les premiers bits de la partie data. L'interprétation de ces valeurs sera développée dans la partie 3.3.

Avant de pouvoir communiquer entre eux à l'aide du protocole R-net, les appareils branchés sur le réseau CAN du fauteuil ont une phase d'appareillage. Cette dernière permet à chaque appareil de cartographier le réseau et de s'attribuer un identifiant fixe. Cette phase ne se répétera que si un élément nouveau vient s'ajouter au réseau, ou si un élément y est retiré. Elle dure un certain temps et les échanges qui s'y font ne sont compris que par le fabricant.

Une fois cette phase terminée, un redémarrage est demandé à l'utilisateur. À partir de ce moment-là, ce sera la phase dite d'initialisation qui se lancera à chaque démarrage. Le but de celle-ci est d'allumer tous les appareils sur le réseau CAN et de les initialiser. Cette phase nécessaire est aussi inconnue du grand public. C'est pourquoi seule sa fin est identifiée et attendue pour prendre le contrôle du fauteuil.

Voici un extrait d'un fichier de log : 3.2 sur lequel on peut voir la fin de la séquence d'initialisation, suivie d'une séquence classique d'échange de trame sur le réseau CAN du fauteuil. Comme dit précédemment la

FIGURE 3.1 – Découpage d'une trame R-net de position de joystick, reçue d'un JSM.

séquence d'initialisation est complexe et inconnue. C'est pour cela que l'on va attendre sa fin pour commencer à modifier les trames qui contrôlent réellement le fauteuil.

On est maintenant en droit de se poser la question suivante : comment comprendre le rôle de trames dont on ne sait rien ? La prochaine partie va décrire comment les premières trames ont été décodées, et comment lors de ce stage, nous avons trouvé le rôle d'autres trames.

3.2.2 Rétro-ingénierie du protocole

Comment découvrir le rôle de nouvelles trames ?

Reprenons la figure 3.2. La première trame à avoir été découverte et interprétée est celle de position de joystick. Le chemin de pensée pour en déduire sa nature est le suivant :

- Il s'agit de la trame la plus présente dans la phase de fonctionnement. Un simple histogramme l'a montré.
- La régularité de cette trame est forte. On voit qu'elle est envoyée par le JSM toutes les 0.015s (cf figure 3.2), et on sait qu'un joystick doit être réactif autant pour avancer que pour s'arrêter.
- Lorsque l'on bouge le joystick pendant un enregistrement, seul la seconde moitié de cette trame change. Les autres trames sont inchangées.

Tous ces indices laissent à penser que cette trame porte l'information de position du joystick.

Prenons un exemple de trame qui a été découverte pendant mon stage : les trames de placement de l'assise. Sur le fauteuil, il y a une assise complètement mobile. Le dossier peut être baissé, ou le cale-pied monté pour le confort de l'usager. Or, les vérins qui contrôlent l'assise sont pilotables via le JSM. Et lors du Hackathon de Juin, nous avons identifié les trames responsables afin de les répéter.

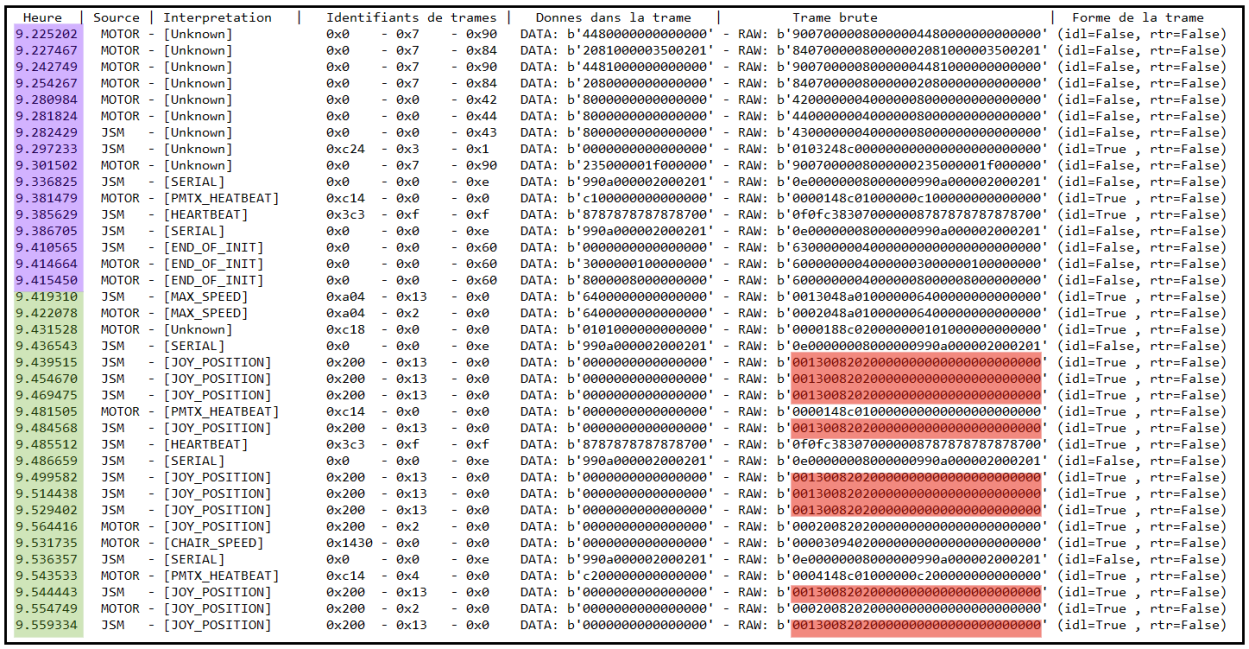
Pour ce faire, nous avons enregistré un échange pendant qu'un vérin était en action. Nous avons remarqué une trame supplémentaire inconnue répétée à une fréquence de 100 Hz, dont les données changeaient lorsque un autre vérin était actionné. Nous en avons déduit la trame responsable du mouvement des vérins, ainsi que l'indice de chaque vérin dans la partie donnée. En recréant cette trame, nous avons pu depuis le serveur web, contrôler l'assise du siège.

Nous avons aussi trouvé les trames responsables de la gestion des phares, et du retour de vitesse (que nous renvoie le moteur).

3.2.3 Sucre pratique

Une fois la compréhension de ces trames acquise, j'ai ajouté certaines fonctionnalités pratiques au fauteuil. Voici la liste.

J'ai commencé par ajouter le redémarrage automatique du programme lorsque le bouton OFF du JSM est



Heure	Source	Interpretation	Identifiants de trames	Donnes dans la trame	Trame brute	Forme de la trame
9.225202	MOTOR	[Unknown]	0x0 - 0x7 - 0x90	DATA: b'4480000000000000'	RAW: b'900700000800000044800000000000'	(idl=False, rtr=False)
9.227467	MOTOR	[Unknown]	0x0 - 0x7 - 0x84	DATA: b'2081000003500201'	RAW: b'840700000800000020810000003500201'	(idl=False, rtr=False)
9.242749	MOTOR	[Unknown]	0x0 - 0x7 - 0x90	DATA: b'4481000000000000'	RAW: b'900700000800000044810000000000'	(idl=False, rtr=False)
9.254267	MOTOR	[Unknown]	0x0 - 0x7 - 0x84	DATA: b'2080000000000000'	RAW: b'840700000800000020800000000000'	(idl=False, rtr=False)
9.280984	MOTOR	[Unknown]	0x0 - 0x0 - 0x42	DATA: b'8000000000000000'	RAW: b'420000004000000800000000000000'	(idl=False, rtr=False)
9.281824	MOTOR	[Unknown]	0x0 - 0x0 - 0x44	DATA: b'8000000000000000'	RAW: b'440000004000000800000000000000'	(idl=False, rtr=False)
9.282429	JSM	[Unknown]	0x0 - 0x0 - 0x43	DATA: b'8000000000000000'	RAW: b'430000004000000800000000000000'	(idl=False, rtr=False)
9.297233	JSM	[Unknown]	0xc24 - 0x3 - 0x1	DATA: b'0000000000000000'	RAW: b'0103248c0000000000000000000000'	(idl=True , rtr=False)
9.301502	MOTOR	[Unknown]	0x0 - 0x7 - 0x90	DATA: b'235000001f000000'	RAW: b'900700000800000023500001f000000'	(idl=False, rtr=False)
9.336825	JSM	[SERIAL]	0x0 - 0x0 - 0xe	DATA: b'990a000002000201'	RAW: b'0e0000000800000990a000002000201'	(idl=False, rtr=False)
9.381479	MOTOR	[PMTX_HEATBEAT]	0xc14 - 0x0 - 0x0	DATA: b'c100000000000000'	RAW: b'0000148c01000000c100000000000000'	(idl=True , rtr=False)
9.385629	JSM	[HEARTBEAT]	0x3c3 - 0xf - 0xf	DATA: b'8787878787878700'	RAW: b'0f0fc383070000008787878787878700'	(idl=True , rtr=False)
9.386705	JSM	[SERIAL]	0x0 - 0x0 - 0xe	DATA: b'990a000002000201'	RAW: b'0e0000000800000990a000002000201'	(idl=False, rtr=False)
9.410565	JSM	[END_OF_INIT]	0x0 - 0x0 - 0x60	DATA: b'0000000000000000'	RAW: b'6300000004000000000000000000000'	(idl=False, rtr=False)
9.414664	MOTOR	[END_OF_INIT]	0x0 - 0x0 - 0x60	DATA: b'3000000100000000'	RAW: b'6000000004000000300000010000000'	(idl=False, rtr=False)
9.415450	MOTOR	[END_OF_INIT]	0x0 - 0x0 - 0x60	DATA: b'8000008000000000'	RAW: b'6000000004000000800000800000000'	(idl=False, rtr=False)
9.419310	JSM	[MAX_SPEED]	0xa04 - 0x13 - 0x0	DATA: b'6400000000000000'	RAW: b'0013048a0100000064000000000000'	(idl=True , rtr=False)
9.422078	MOTOR	[MAX_SPEED]	0xa04 - 0x2 - 0x0	DATA: b'6400000000000000'	RAW: b'0002048a0100000064000000000000'	(idl=True , rtr=False)
9.431528	MOTOR	[Unknown]	0xc18 - 0x0 - 0x0	DATA: b'0101000000000000'	RAW: b'0000188c0200000001010000000000'	(idl=True , rtr=False)
9.436543	JSM	[SERIAL]	0x0 - 0x0 - 0xe	DATA: b'990a000002000201'	RAW: b'0e0000000800000990a000002000201'	(idl=False, rtr=False)
9.439515	JSM	[JOY_POSITION]	0x200 - 0x13 - 0x0	DATA: b'0000000000000000'	RAW: b'0013008202000000000000000000000'	(idl=True , rtr=False)
9.454670	JSM	[JOY_POSITION]	0x200 - 0x13 - 0x0	DATA: b'0000000000000000'	RAW: b'0013008202000000000000000000000'	(idl=True , rtr=False)
9.469475	JSM	[JOY_POSITION]	0x200 - 0x13 - 0x0	DATA: b'0000000000000000'	RAW: b'0013008202000000000000000000000'	(idl=True , rtr=False)
9.481505	MOTOR	[PMTX_HEATBEAT]	0xc14 - 0x0 - 0x0	DATA: b'0000000000000000'	RAW: b'0000148c01000000000000000000000'	(idl=True , rtr=False)
9.484568	JSM	[JOY_POSITION]	0x200 - 0x13 - 0x0	DATA: b'0000000000000000'	RAW: b'0013008202000000000000000000000'	(idl=True , rtr=False)
9.485512	JSM	[HEARTBEAT]	0x3c3 - 0xf - 0xf	DATA: b'8787878787878700'	RAW: b'0f0fc383070000008787878787878700'	(idl=True , rtr=False)
9.486659	JSM	[SERIAL]	0x0 - 0x0 - 0xe	DATA: b'990a000002000201'	RAW: b'0e0000000800000990a000002000201'	(idl=False, rtr=False)
9.499582	JSM	[JOY_POSITION]	0x200 - 0x13 - 0x0	DATA: b'0000000000000000'	RAW: b'0013008202000000000000000000000'	(idl=True , rtr=False)
9.514438	JSM	[JOY_POSITION]	0x200 - 0x13 - 0x0	DATA: b'0000000000000000'	RAW: b'0013008202000000000000000000000'	(idl=True , rtr=False)
9.529402	JSM	[JOY_POSITION]	0x200 - 0x13 - 0x0	DATA: b'0000000000000000'	RAW: b'0013008202000000000000000000000'	(idl=True , rtr=False)
9.564416	MOTOR	[JOY_POSITION]	0x200 - 0x2 - 0x0	DATA: b'0000000000000000'	RAW: b'0002008202000000000000000000000'	(idl=True , rtr=False)
9.531735	MOTOR	[CHAIR_SPEED]	0x1430 - 0x0 - 0x0	DATA: b'0000000000000000'	RAW: b'0000309402000000000000000000000'	(idl=True , rtr=False)
9.536357	JSM	[SERIAL]	0x0 - 0x0 - 0xe	DATA: b'990a000002000201'	RAW: b'0e0000000800000990a000002000201'	(idl=False, rtr=False)
9.543533	MOTOR	[PMTX_HEATBEAT]	0xc14 - 0x4 - 0x0	DATA: b'c200000000000000'	RAW: b'0004148c01000000c200000000000000'	(idl=True , rtr=False)
9.544443	JSM	[JOY_POSITION]	0x200 - 0x13 - 0x0	DATA: b'0000000000000000'	RAW: b'0013008202000000000000000000000'	(idl=True , rtr=False)
9.554749	MOTOR	[JOY_POSITION]	0x200 - 0x2 - 0x0	DATA: b'0000000000000000'	RAW: b'0002008202000000000000000000000'	(idl=True , rtr=False)
9.559334	JSM	[JOY_POSITION]	0x200 - 0x13 - 0x0	DATA: b'0000000000000000'	RAW: b'0013008202000000000000000000000'	(idl=True , rtr=False)

FIGURE 3.2 – Extrait d'un fichier de log enregistré lors de l'allumage du fauteuil. En violet la fin de phase d'initialisation. En vert la phase de fonctionnement classique et en rouge des trames de position de joystick

pressé. Une trame spécifique est envoyée à ce moment-là, et il suffisait de remettre notre service *RnetCtrl* dans son état initial. Avant cela il, était nécessaire de redémarrer à la main ce programme.

Ensuite, j'ai ajouté un contrôle automatique des clignotants. Lorsque l'on se déplace en extérieur avec le fauteuil, il est très inconfortable de devoir quitter sa conduite pour allumer un clignotant à chaque virage en sachant qu'il faut l'éteindre immédiatement après. C'est pourquoi j'ai mis en place une option capable d'allumer et d'éteindre automatiquement les clignotant en fonction de l'inclinaison gauche-droite du joystick faible force. Ceci ajoute un confort de conduite non-négligeable pour l'utilisateur. Dans ce mode, les feux de détresse s'allument aussi si le fauteuil recule.

Enfin, j'ai ajouté le fait que le joystick faible force contrôle, soit le fauteuil, soit la souris sur le portable. Il n'y a pas d'interférence possible. C'est-à-dire que le joystick ne peut théoriquement pas bouger dans le vide, ou contrôler les deux en même temps. De plus, l'option clignotant automatique décrite précédemment n'est pas active quand le joystick contrôle la souris sur le portable.

3.2.4 Tentatives de substitution du JSM

Au cours du Hackathon de Juin auquel nous avons participé, une idée nous est venue. Un des participants a proposé d'émuler un JSM à l'aide de la Raspberry Pi. La figure 3.3 décrit la nouvelle disposition imaginée.

Quels sont les avantages de cette disposition ? Tout d'abord, la Raspberry en coupure n'était pas capable d'allumer le fauteuil. En effet, il fallait appuyer sur le bouton du JSM pour cela. Dans cette situation, cela devient possible. Ensuite, une seule interface CAN est désormais nécessaire, contre deux en coupure. De plus, nous ne sommes plus dépendants de l'état de fonctionnement du JSM. Le fauteuil reste utilisable s'il tombe en panne. Enfin, si c'est la Raspberry Pi qui tombe en panne, l'utilisateur n'est pas immobilisé. Il y a encore le JSM initial branché sur le réseau qui peut déplacer le fauteuil avec l'aide d'une tierce personne.

Maintenant, il reste à savoir comment émuler un JSM dont le protocole de communication reste encore

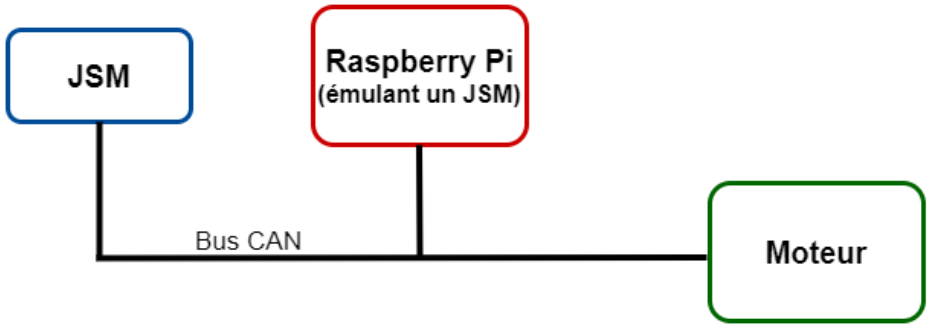


FIGURE 3.3 – Schéma bloc du réseau CAN du fauteuil avec la Raspberry Pi émulant un JSM

majoritairement inconnu. Durant les phases de fonctionnement classique (hors séquence d'initialisation et d'appareillage), la communication est en fait assez simple. Comme on peut le voir sur la figure 3.2, toutes les trames envoyées par le JSM dans la période verte sont connues. Les trames de position de joystick sont envoyées à fréquence régulière, ainsi que d'autres trames périodiques comme celle de *HEARTBEAT*, qui assurent que chaque appareil est toujours branché.

Ainsi, seul les séquences d'initialisation et d'appareillage posent problème. Cependant, il n'est pas nécessaire de les comprendre pour les rejouer. En utilisant les deux interfaces CAN de la Raspberry Pi, ainsi qu'un second JSM, il est possible d'apprendre ces séquences trame par trame afin de les répéter.

J'ai repris des bouts de code d'anciens Hackathon qui permettaient d'apprendre et de rejouer des séquences de trames sur un bus CAN. J'ai reconstruit un script Python qui, avec les branchements de la figure 3.4, permet très facilement de sauvegarder les deux séquences voulues.

En résumé, quand une trame arrive sur l'une des interfaces, son origine, son heure d'arrivée et la trame en elle-même sont sauvegardées dans un fichier. Pour rejouer ces séquences, le fichier est lu chronologiquement. Les trames qui ont été sauvegardées avec le JSM temporaire comme origine sont alors envoyées par la Raspberry Pi au bon moment. Les trames sauvegardées avec le reste du réseau comme origine sont attendues avant de passer à la suite.

Lors de nos expérimentations, les résultats étaient instables. Le moteur pouvait réagir correctement en s'initialisant sans complication, mais il se coupait aussi aussitôt la séquence d'initialisation finie.

Malgré le fait d'être capable de reproduire à l'identique ces séquences, cela n'a pas suffi à contrôler totalement le fauteuil dans la position d'émulation figure 3.3. Il faudrait comprendre plus en détail ces séquences pour arriver à les utiliser. Christophe a émis l'idée de faire des études statistiques sur les trames présentes, et m'a demandé de lui faire des enregistrements de ces séquences à la fin de mon stage.

3.3 Joystick faible force

Lors de mon stage, j'ai aussi dû reprendre le traitement des données que nous envoie le joystick faible force (cf figure 2.3). C'est le service *Joystick* (cf figure 2.4) qui s'occupe de cela. Je vais décrire dans l'ordre les modifications que j'ai implémentées et leurs impacts sur le projet.

3.3.1 Intégration logicielle d'un nouveau capteur

Tout d'abord, l'équipe a choisi un nouveau capteur de position qui a été testé pendant le Hackathon par l'un de nos collègues du SED. J'ai repris son travail qui offrait un exemple d'utilisation de ce nouveau capteur afin de l'introduire dans le code pré-existant.

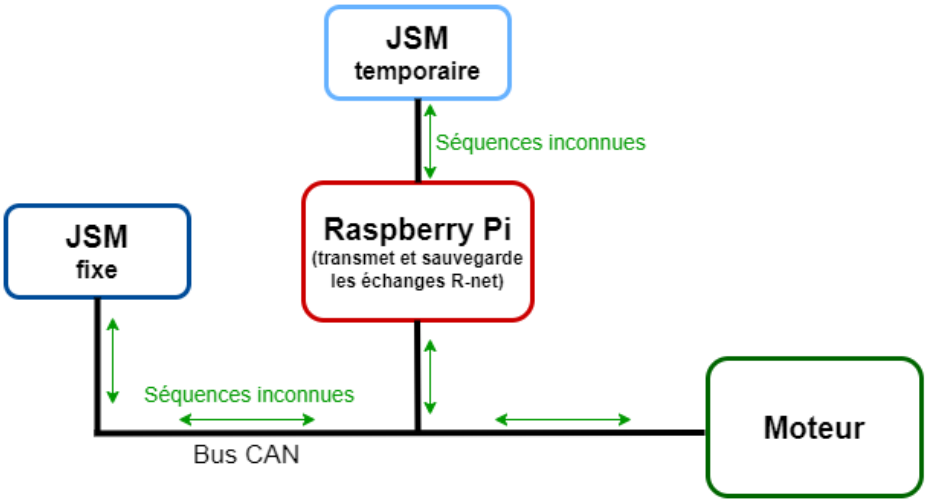


FIGURE 3.4 – Schéma bloc du réseau CAN du fauteuil avec la Raspberry Pi en apprentissage des séquences d'initialisation et d'appareillage. Les flèches vertes représentent les échanges de trames lors de ces séquences.

Il s'agit du capteur *MLX90393*. Ce dernier a la particularité d'avoir un convertisseur analogique-numérique intégré. Il communique directement via l'I2C. J'ai donc ajouté un paramètre au script Python du service *Joystick* afin d'indiquer quel capteur on souhaite utiliser.

3.3.2 Normalisation des valeurs brutes

Les valeurs transmises dans les trames R-net de position de joystick sont des valeurs entre -100 et 100 pour X et pour Y. Elles doivent être codées sur 8 bits en complément à deux. Il faut donc traduire les valeurs brutes lues sur les pins d'entrée en des valeurs comprise entre -100 et 100.

Pour cela, j'ai simplement utilisé la fonction affine par partie présentée sur la figure 3.5. Chaque coordonnée a sa propre fonction, celle-ci est celle de X, mais celle de Y est quasi identique. Informatiquement, je n'ai eu qu'à calculer le coefficient directeur de la bonne droite pour trouver la valeur R-net correspondante. Avec un X brut plus grand que la valeur de repos, on trouve la formule suivante :

$$X_{Rnet} = \frac{100}{X_{brut\_max} - X_{repos}} \times X_{brut}$$

Les valeurs minimales et maximales de X sont dynamiques. En effet, on ne connaît pas à l'avance l'amplitude du joystick faible force et les valeurs brutes associées aux buttées. Il faut donc que cet algorithme de traitement prenne cela en compte. La courbe est affine par morceaux, car l'amplitude positive n'est pas forcément égale à l'amplitude négative. Le joystick peut être asymétrique. De plus, le capteur dépend d'effets non-linéaires.

3.3.3 Zone morte

L'étape suivante fut d'ajouter une zone morte au calcul précédent. Le capteur de position est précis, mais pas forcément très stable. Pour ajouter un léger confort, nous avons décidé qu'un petit pourcentage de l'amplitude des valeurs brutes allait être associé à la valeur R-net nulle. La figure 3.6 décrit la nouvelle transformation. La formule se voit légèrement modifiée.

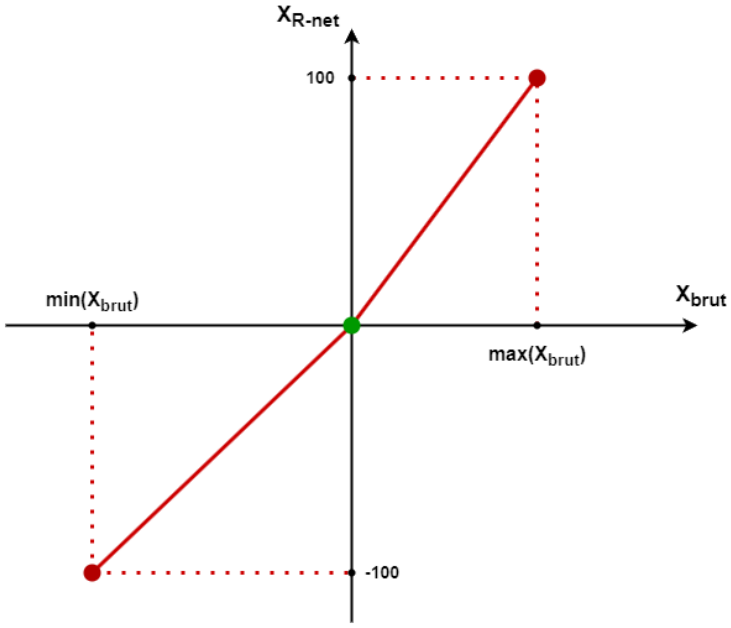


FIGURE 3.5 – Fonction permettant de traduire une valeur brute de X en valeur compatible avec le protocole R-net. En abscisse les valeurs de X brutes, en ordonnée les valeurs R-net valides. Le point vert représente la valeur de X brute avec le joystick au repos.

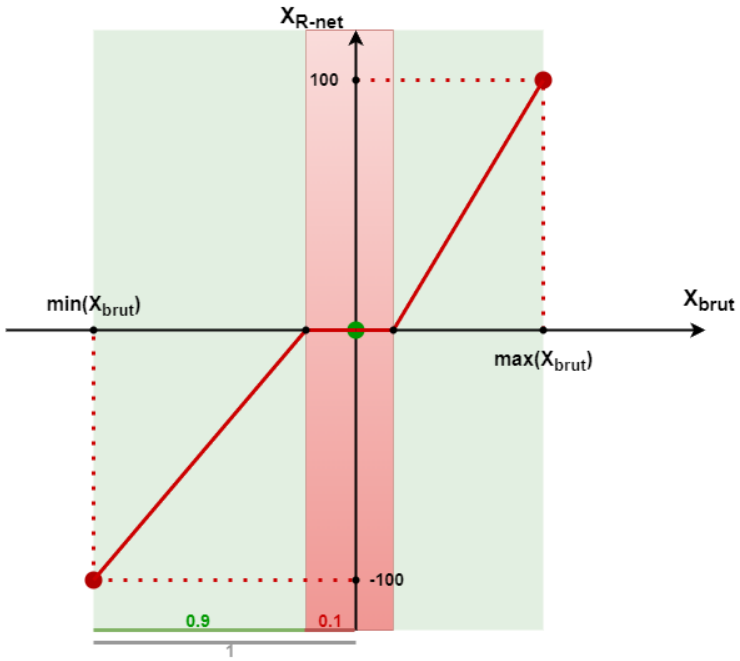


FIGURE 3.6 – Fonction similaire à la figure 3.5, mais faisant apparaître la zone morte du joystick en rouge. La zone mort ici représentée fais un dixième de l'amplitude totale du joystick faible force. Les axes sont identiques à la figure 3.5

3.3.4 Auto-calibration du joystick

Après cela, une nouvelle amélioration a été proposée et implémentée. Il s'agit d'une mise à jour automatique de la valeur de repos. Cette dernière était initialement déterminée au démarrage du service, en assumant que le joystick faible force est à sa position de repos. Cependant, le risque que le point de repos se déplace n'est pas nul car la mécanique du joystick faible force n'est pas d'une précision industrielle. Le mettre à jour ajoute un confort et une sécurité.

Le principe est le suivant. Soit F_X la fenêtre temporelle des N derniers échantillons de positions brutes pour la coordonnée X. $F_X[i]$ est la i-ème valeur la plus récente. On calcule l'espérance $\mathscr{E}$ et la variance $\mathscr{V}$ de F_X avec les formules suivantes :

$$\mathscr{E} = \frac{1}{N}\sum_{i=1}^{N} F_X[i]$$

$$\mathscr{V} = \frac{1}{N}\sum_{i=1}^{N} (F_X[i] - \mathscr{E})^2$$

On souhaite mettre à jour la valeur de la position de repos uniquement quand le joystick est réellement au repos. Ces deux valeurs vont nous aider à choisir quand la mettre à jour. La valeur de $\mathscr{V}$ représente la quantité de mouvement sur les derniers N échantillons, car il s'agit des écarts à la moyenne $\mathscr{E}$. Ainsi, quand $\mathscr{V}$ est proche de zéro, c'est que le joystick faible force ne bouge pas. Cette situation peut décrire deux comportements. Soit le joystick est effectivement au repos, soit l'utilisateur est en butée, c'est à dire qu'il pousse au maximum le joystick vers une direction. Afin de différencier ces deux cas, on va comparer la valeur de la position de repos actuelle avec $\mathscr{E}$. Si l'écart est grand, c'est que l'utilisateur est en butée, sinon, c'est que le joystick faible force est au repos. Dans ce dernier cas, l'espérance devient la nouvelle position de repos.

3.3.5 Ajustement de l'intervalle de valeurs possibles

Une fois ces améliorations codées, nous sommes allés expérimenter le fauteuil. Nous nous sommes rapidement rendu compte que son contrôle n'était pas intuitif et devait être revu. Après réflexions, nous nous sommes posé la question suivante. Le JSM et notre joystick faible force ont ils la même portée ? Nous savons que notre joystick peut atteindre et envoyer des positions en X Y égales à [100,100] car il est déplaçable dans un carré. Mais est-ce que le JSM, qui possède un socle circulaire peut en faire autant ? Après vérifications, il s'avère que non. Le problème est décrit visuellement sur la figure 3.7

Afin de correspondre au mieux aux positions que pourrait envoyer un JSM, nos avons décider d'appliquer une fonction à nos valeurs X Y. Cette fonction change la disposition carrée en une disposition circulaire. Elle est tracée sur la figure 3.8. Cette fonction aplatit l'espace carré initial dans le cercle circonscrit à celui-ci.

3.3.6 Ajustement de X en fonction de Y

Enfin, lors de l'utilisation du fauteuil, nous avons remarqué que ce dernier réagissait trop violemment à l'axe des X. Il tournait trop vite et trop fort au moindre petit décalage en X. C'est pourquoi nous avons décidé de réduire artificiellement la valeur de X retenue en fonction de celle de Y. Nous voulions garder une forte maniabilité lorsque le fauteuil n'avance pas (Y faible), mais la réduire quand le fauteuil roule (Y élevé).

Nous avons donc appliqué la formule suivante à X.

$$X_f = X \times \frac{m}{m + |Y|}$$

Avec m un entier supérieur à 1, dit coefficient de maniabilité. X_f est la valeur final de X. On voit que plus Y est grand, plus X_f diminue. Cette diminution est pilotée par m. Quand m tend vers l'infini, la fraction tend vers 1 et X n'est que peu modifié.

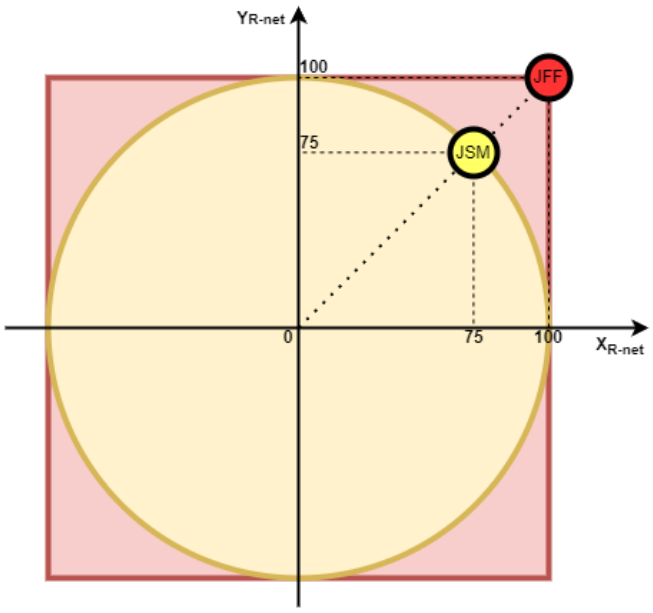


FIGURE 3.7 – Représentation graphique de la portée atteignable par le JSM (zone jaune) et notre joystick faible force (zone rouge) sur l'ensemble des valeurs de position R-net en X et Y.

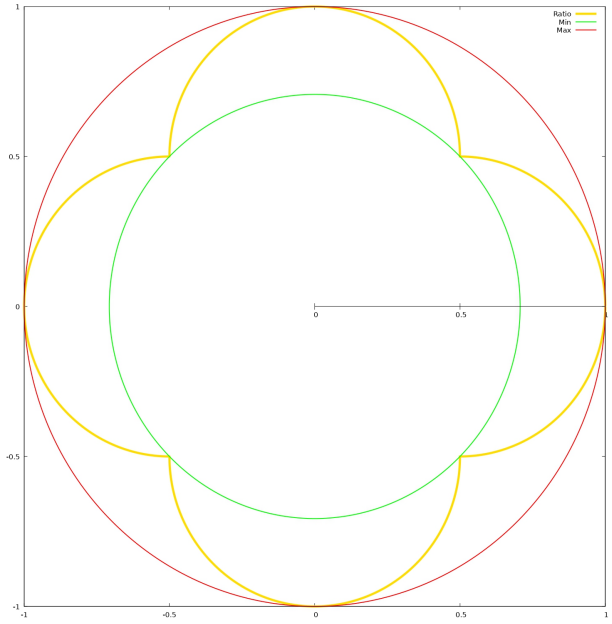


FIGURE 3.8 – Courbe de la fonction polaire appliquée (jaune). Elle coefficiente la coordonnée radiale r d'un point en fonction de sa coordonnée angulaire θ. Tous les points situés sur le carré de largeur 2 centré à l'origine sont envoyés sur le cercle trigonométrique. Valeur minimale de la fonction en vert, valeur maximale en rouge.

3.4 Souris Bluetooth

La gestion de la connexions Bluetooth ainsi que l'émulation de la souris sur le téléphone furent des challenges plus difficiles que les autres. C'est ma collègue Roxanne Xu qui s'est occupée de coder ces parties, mais la complexité du problème nous a forcé à demander l'aide active de Christophe, notre maître de stage.

Finalement, la connexion entre les messages MQTT et les messages Bluetooth a pu se faire. Les données de positions sont écrites bit à bit dans le message Bluetooth envoyé au téléphone. Le joystick faible force est en mesure de piloter la souris sur celui-ci après avoir été connecté à la Raspberry Pi.

3.5 Infrarouge

Pour donner plus d'autonomie à l'utilisateur final du fauteuil, nous avons ajouté un contrôleur infrarouge. Ce petit boîtier contient une rangée de LED infrarouges branchées en parallèle, ainsi qu'un récepteur. Tout cela est branché aux pins d'entrée-sortie encore inoccupés de la Raspberry Pi. Notre but est de pouvoir capturer, puis rejouer des séquences infrarouges, sans pour autant les comprendre. C'est ma collègue Roxanne qui s'est occupée de cette partie. Mais j'ai suivi ses avancées pour rester à jour.

Une nouvelle section de pages a été créée pour le contrôle infrarouge sur l'interface Web. La page principale présente une grille de tuiles cliquable. Pour sauvegarder une action d'une télécommande infrarouge sur la Raspberry, il faut réaliser la suite d'actions suivantes.

Les tuiles rouges sont vides et déclenchent un enregistrement si elles sont activées. L'utilisateur (ou une auxiliaire de vie) doit alors appuyer sur le bouton de la télécommande en visant le capteur infrarouge du fauteuil. Une fois la séquence infrarouge capturée, la même tuile devient verte et la séquence est répétée à chaque clic suivant. Ainsi, l'utilisateur peut programmer n'importe quel appareil infrarouge, indépendamment du protocole utilisé par ce dernier. Nous avons effectué nos tests concluants avec une télévision et sa télécommande.

Chapitre 4

Bilan

Ces trois mois à l'Inria ont été une expérience forte. Je sais qu'elle va me guider dans le choix de mon futur emploi. Je vais décrire dans ce chapitre ce que je tire de ce stage, aussi bien humainement que professionnellement.

4.1 Compétences technique

Lors de ce stage, j'ai eu souvent recours à mes connaissances systèmes, et je les ai améliorées. Il y a en particulier les configurations pour gérer le point d'accès Wifi, la gestion du bluetooth ou encore celle de l'infrarouge. J'ai du également traiter la reproductibilité de ces configurations en les automatisant par un script Shell ; ainsi l'installation du système sera plus simple pour l'utilisateur final.

J'ai découvert aussi les problématiques de rétro-ingénierie, que l'on aborde peu durant une formation académique. L'ouverture et l'agilité d'esprit dont il faut faire preuve dans ce domaine sont particulièrement stimulantes.

J'ai du faire aussi du prototypage électronique pour relier différents composants à la Raspberry Pi. Cela a été l'occasion de m'initier à l'utilisation du fer à souder en suivant les bonnes pratiques et les règles de sécurité. C'est encore un aspect que je n'avais pas vu lors de mon parcours académique.

J'ai eu l'occasion de mettre en pratique des connaissances basiques en électronique comme par exemple lors de l'élaboration de la carte d'interfaces pour les LED infrarouges ou la fabrication du joystick faible force.

J'ai pu approfondir des principes mathématiques avec le traitement des informations du joystick faible force. Je maîtrisais les concepts d'espérance et de variances, mais je n'avais pas étudié les fonctions polaires lors de ma licence.

J'ai appris l'existence d'outils très utiles et dont je me resservirai très certainement. *Gnuplot* est un logiciel interactif en ligne de commande qui m'a servi à tracer des courbes afin de visualiser certains problèmes. Par exemple, nous avons identifié une trame R-net envoyée par le moteur comme étant le retour de vitesse linéaire et angulaire du fauteuil grâce à nos courbes sur Gnuplot. Le fait qu'il soit en ligne de commande permet d'avoir un tracé en très peu de temps.

Je retiens l'utilisation de *FreeCAD* pour la modélisation de pièce en 3D. Je retiens aussi *Supervidord* pour le suivi, la surveillance et le contrôle de processus.

Enfin, ce stage m'a permis de confirmer mon niveau avec le langage Python. J'ai été à l'aise avec son utilisation. Nous nous sommes notamment servi de threads dans nos programmes, et de sockets pour la communication avec l'extérieur sans que ces principes n'obscurcissent ma compréhension.

4.2 Impression générale

J'adore la symbiose électronique informatique de ma filière. Je souhaite en faire mon métier et j'ai trouvé lors de ce stage la satisfaction de toucher du doigt ce que je souhaite faire plus tard. Mon impression générale sur ce stage est donc très positive. Au-delà de ça, je me suis senti inclus dans le service SED dès la première semaine.

L'organisation du Hackathon au tiers du stage a permis de se rapprocher de tous les membres du projet. J'y ai rencontré Jonathan, mais j'ai aussi vu ce que peu de monde peuvent réussir à faire en peu de temps.

Je garde un très bon souvenir de mon duo avec Roxanne Xu. La présence d'une collègue étudiante m'a permis d'avoir un appui pour m'intégrer, mais aussi de casser la monotonie qui s'installe lorsqu'on travaille seul.

Je finirai en soulignant l'ambiance de travail particulièrement agréable. Je n'ai senti ni pression ni stress, tout en restant productif. J'espère trouver une future expérience professionnelle aussi agréable que celle-ci.