Clara BISALLI - Caroline ROUSSEAU - Nour EKHLAS - Laurine VERNIZEAU

IMA4

Projet P18

Dispositif d'aide au déplacement pour jeune malvoyant

Alexandre BOE - Marion BINNINGER - Xavier REDON - Thomas VANTROYS

Année 2019-2020

Sommaire

Introduction

Dans le cadre de notre formation, le projet “Dispositif d’aide au déplacement pour jeune malvoyant” nous a été proposé. Celui-ci consiste à réaliser un dispositif permettant le déplacement en toute sécurité et autonomie d'une personne malvoyante. L’objectif est de permettre à cette personne malvoyante de réaliser une balade en forêt, sur terrain plat ou accidenté. Le dispositif reproduisant la forme d’une main sera positionné sur l’épaule de la personne concernée et lui indiquera par le biais d’une oreillette et des indications vocales, ainsi que par des pressions et des vibrations, les dangers potentiels devant lui.

Le travail des deux derniers semestres était principalement un travail de fondations : sélection et commande des composants, premiers jets d’algorithmes . S'ensuit l’objectif principal de notre travail ce semestre qui était de concrétiser toutes les décisions prises et la création d’un second prototype complètement fonctionnel (en posant quelques conditions sur son utilisation).

Le projet se scinde en deux : les réalisations de codes et du matériel formant le dispositif.

Dans ce rapport récapitulatif, nous allons dans un premier temps explorer les principes de fonctionnement du code du capteur et du code gérant les actionneurs de la main. Puis dans une seconde partie nous aborderons la conception du Hat puis la modélisation et la réalisation de la main et du harnais supportant le dispositif.

I. Partie Software

L'enjeu principal de ce projet était de pouvoir indiquer, par l'intermédiaire de capteurs optiques et d'actionneurs, la présence d'un obstacle dans le champs de vision de Florian.
Nous nous sommes, suites aux recommandations des accompagnatrices de l'enfant, focalisées sur les reliefs présents sur le sol.
L'enjeu était donc de capter et d'informer l'utilisateur, de la manière la plus fiable, de la présence de reliefs marqués pouvant engendrer un accident.
Pour cela nous avons divisé notre code en deux parties distinctes, code capteur et code actionneurs, communicant entre eux à l'aide d'une file de messages IPC (Inter-Process Communication).
Pour ne pas surcharger ce rapport nous allons principalement exposer la logique et l'algorithmique des codes élaborés (plus de détails sont présents sur le wiki : https://projets-ima.plil.fr/mediawiki/index.php/IMA3/IMA4_2018/2020_P18).

1. Installation du SDK

1.1. Choix de la Raspberry

L'installation du SDK a été faite sur deux Raspberry Pi. La Raspberry Pi 4 a été choisie pour le projet final en raison de sa plus grande rapidité, du fait de sa plus grande capacité mémoire RAM et de la présence de ports USB 3. La deuxième installation s'est faite sur une Raspberry Pi 3 pour cause de confinement.
Le début d'installation a commencé sur la RPi 3 avec Ubuntu 18.04 comme l'indiquait le tutoriel d'installation du SDK de la D435 sur le site d'Intel. Le choix de la Raspberry 3 était dû au fait qu'Intel indiquait que la dernière version d'Ubuntu n'était pas compatible avec la RPi 4. Entre temps, des mises à jour de Raspibian ont permis d'installer le SDK de la D435 sur la RPi 4. Il y a donc eu changement de RPi avec la RPi 4 dont l'OS Rasbian a été installé.

1.2. Raspberry Pi 3 Ubuntu 18.04

Le travail sur Raspberry Pi 3 a été effectué pour le test du hat et de la caméra D435. L'installation d'Ubuntu sur la RPi 3 était un peu plus longue du fait de sa capacité mémoire RAM limitée qui a nécessité la création et l'utilisation d'un fichier swap. Grâce à son environnement de Bureau Mate, l'utilisation d'Ubuntu était plus simple pour la modification de codes ou la visualisation des images et des traitements de la D435.

1.3. Configuration et installation du SDK

L'installation du SDK s'est faite en ligne de code que l'on retrouve dans le Wiki. Il fallait cloner le répertoire librealsense sur git, installer des paquets pour construire les binaires librealsense, exécuter des scripts.
Une fois ceci fait on crée un répertoire *build* ou l'on exécute *cmake* qui va construire les démos et les tutoriels. Ensuite on compile et on finit l'installation avec *make && make install.* Lors de la compilation il peut se produire une erreur du fait des 1Go de RAM. Pour cela le tutoriel propose d'ajouter 1Go de swap. J'ai rajouté 3Go pour plus de rapidité. Après avoir créé ce fichier swap, la compilation a été effectuée avec succès en à peu près 2 heures.

2. Caméra D435

2.1. Traitement des données

Points délicats:

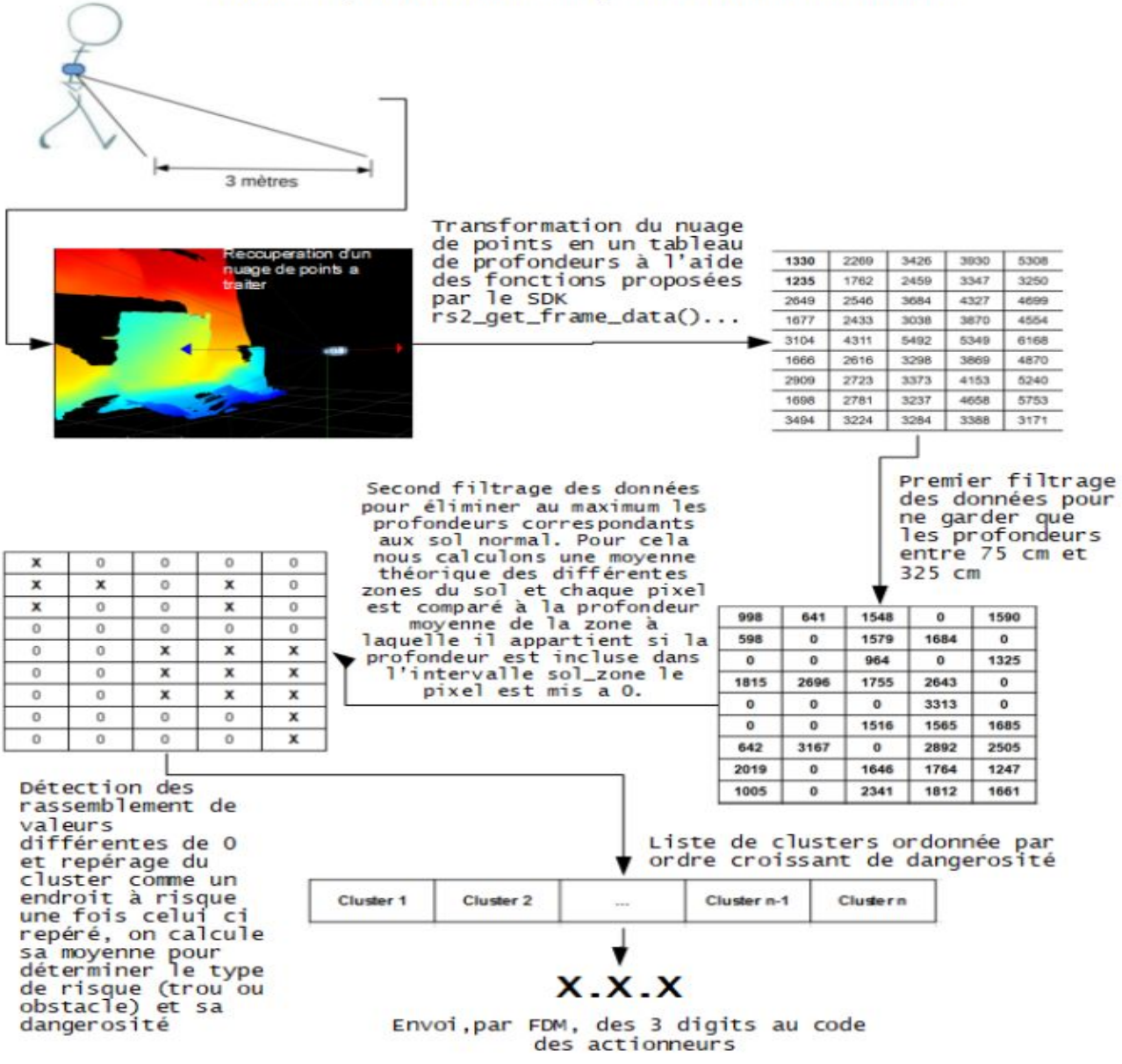
- **Elimination du sol :**
  Nous sommes parties de l'hypothèse que le capteur sera correctement fixé sur le harnais pour vérifier les distances fixées sur le croquis ci-dessous (Valeurs fixées en macros). Nous avons ensuite divisé l'image de profondeur perçue en NB_ZONES différentes. Chaque zone est espacée de l'autre d'un pas_zone qui nous permet en utilisant une simple règle de Pythagore de déterminer la profondeur moyenne de la zone. Cette méthode est en réalité peu pratique car elle repose sur des calculs théoriques dans un milieu dynamique qui bouge en continu. Il faudrait donc dans le futur envisager une élimination dynamique de sol en utilisant des fonctionnalités d'élimination de plans par exemples (très complexe).

- **Détection des Clusters dans la matrice et leur traitement:**
  L'approche récursive de la détections de clusters (voisins non nuls) a très vite montré ses limites en générant un segfault due à la saturation de la pile

d'exécution. Pour pallier à ce problème nous nous sommes tournées vers une alternative non récursive avec une création d'une pile d'exécution dont l'espace mémoire est allouée dans le tas nous offrant donc plus de marge et de puissance pour effectuer notre traitement.
Pour chaque point répondant à nos critères (profondeur != 0), ce dernier est ajouté sur la pile que nous avons créé, puis quand tous les voisins du points initial sont testés on "pop" le dernier élément présent sur la pile et on teste tous ses voisins. Cette étape est répétée jusqu'au moment où tous les voisin du point sont nuls (stack vide).

- **Calcul de la dangerosité:**
  L'indice de dangerosité est calculé grâce à la taille du cluster et à l'écart entre la moyenne du cluster et le sol théorique de la zone dans laquelle il a été détecté.

Schéma récapitulant le déroulement global du traitement des données:



2.2. Pistes exploratoires

Il existe de nombreux algorithmes pour la détection du contour des objets dans une image, et une solution serait d'utiliser ces méthodes afin de localiser les bords des obstacles.
Une méthode très classique consiste à utiliser un filtre de Sobel pour détecter les contours des objets. Celui-ci va calculer le gradient de l'intensité (de la distance) sur chaque pixel. Un contour se caractérise par une rupture d'intensité dans l'image, ces changements d'intensités se traduisent par des discontinuités dans la profondeur.
Ce filtre comporte deux dérivées, une dérivée pour la direction horizontale et une dérivée pour la direction verticale. Ceci permet d'éliminer l'inclinaison du sol par rapport à la caméra et conserver uniquement les contours des objets. Il suffit de détecter pour chaque pixel sa variation par rapport à ses voisins de gauche et de droite, et de dessus et dessous. La méthode consiste à calculer le gradient en H puis en V autour du pixel.

Donc à chaque position (x, y) d'un pixel, il faut calculer la différence entre les 3 pixels à sa gauche avec les 3 à sa droite selon les matrices de pondération pour déterminer le gradient en x, les 3 pixels au dessus avec les 3 pixels en dessous pour avoir le gradient en y. Le pixel fait partie d'un contour si sa valeur est élevée.

Le test du code a été fait sur la Raspberry Pi 3, on remarque du bruit dans le résultat obtenu, mais nous pouvons facilement identifier les contours (Le code détaillé est sur le wiki).

3. Gestion des actionneurs de la main

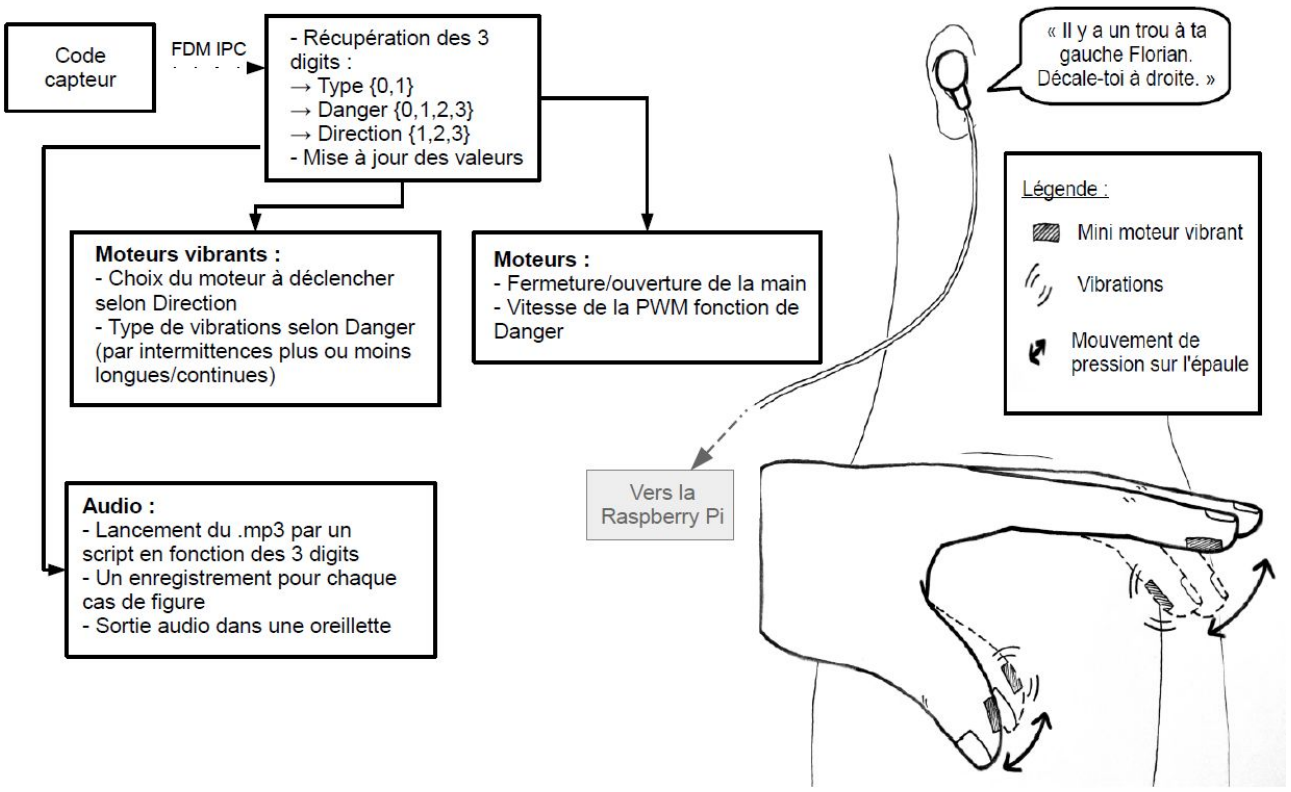
La main centralise l'utilisation des actionneurs et de l'audio. Elle réalise parallèlement les trois tâches suivantes :

– la fermeture puis l'ouverture de la main avec trois moteurs;

– le déclenchement de vibrations via des mini moteurs vibrants;

– le lancement d'un enregistrement sonore adapté à la situation rencontré par l'utilisateur du dispositif.

Schéma récapitulant le déroulement global des actions de la main



La réalisation de ces actions dépend de l'analyse de l'environnement faite par le capteur résumée sous forme de trois paramètres : le type d'obstacle, sa position par rapport au capteur et la dangerosité, selon la proximité de l'obstacle et sa profondeur pour les trous.

Les réactions de la main s'articulent donc selon les valeurs suivantes :

direction	
Valeur	Signification
1	Obstacle À gauche
2	Obstacle À droite
3	Obstacle En face

danger	
Valeur	Signification
0	Pas de danger → Position de Repos pour la main
1	Faible danger
2	Danger intermédiaire
3	Danger important

type	
Valeur	Signification
0	Trou/dénivelé Important
1	Obstacle En hauteur

Ainsi la configuration choisie ci-dessus forme dix-huit cas de figure différents. Un enregistrement a été réalisé pour chacun d'eux (cf. annexe) avec une voix de jeune garçon comme explicitement demandé par Florian. Ces indications reprennent le type d'obstacle, sa position et sont graduelles en fonction du niveau de danger.

Pour plus de lisibilité, le code est scindé en un main (gestionMain) et une bibliothèque de fonctions (fonctionMain). Le lancement de l'enregistrement audio adéquat est réalisé via un script shell.

Les différentes tâches devant être réalisées simultanément, le code se base sur l'utilisation de threads. Comme expliqué précédemment dans le code du capteur, les paramètres sont envoyés par une file de messages IPC et sont réceptionnés dans un thread. Les valeurs sont accessibles à tous les autres threads et mises à jour toutes les sept secondes environ. Cela permet à un enregistrement de se jouer en entier. En effet ils durent de trois à six secondes pour le plus long. Cela permet également de faire un cycle de vibration complet (au maximum six secondes également).

Le thread moteur permet la fermeture puis l'ouverture de la main. Le niveau de danger conditionne la vitesse fournie aux PWM des moteurs et le temps entre la fermeture et l'ouverture. Un ajustement aurait sûrement été nécessaire à trouver entre la vitesse donnée au moteur contrôlant le pouce et ceux reliés aux autres doigts pour avoir un mouvement le plus naturel possible.

Le thread moteurs vibrants utilise trois moteurs sur les quatre disponibles pour indiquer la direction de l'obstacle. Pour un danger faible ou moyen, les vibrations se font par intermittence ; continues dans le cas d'un danger important.

Le quatrième moteur disponible à l'arrière de la main pourrait être déclenché ou non selon que l'obstacle soit au sol ou en hauteur.

La partie audio est gérée par le lecteur multimédia omxplayer. Ce dernier fonctionne en ligne de commande. Le thread audio lance un script shell avec un paramètre qui permet de démarrer le fichier audio correspondant à la situation. Le lancement du script et du lecteur se fait en arrière plan. L'envoi d’une commande permet de fermer omxplayer lorsque que l'enregistrement arrive à son terme. Néanmoins cela empêche le bon fonctionnement des moteurs en parallèle malgré plusieurs approches au problème (utilisation d'un pipe, envoi de commandes, lancement en arrière plan, ajout d'options dans la commande omxplayer, recherche d’autres lecteurs). Dès lors que l'enregistrement démarre, les moteurs restent bloqués dans leur état précédent.

Nous n'avons pas pu tester le code avec la totalité des actionneurs. La partie gestion des différents moteurs s’exécute correctement en l’absence de la partie audio. Le lancement

d'une piste audio et son arrêt est possible, mais ne fonctionne qu'une seule fois par exécution du code.

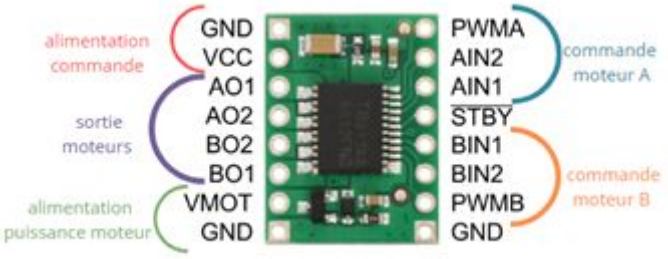
II. Partie Hardware

1. Réalisation d'un Hat pour la Raspberry Pi

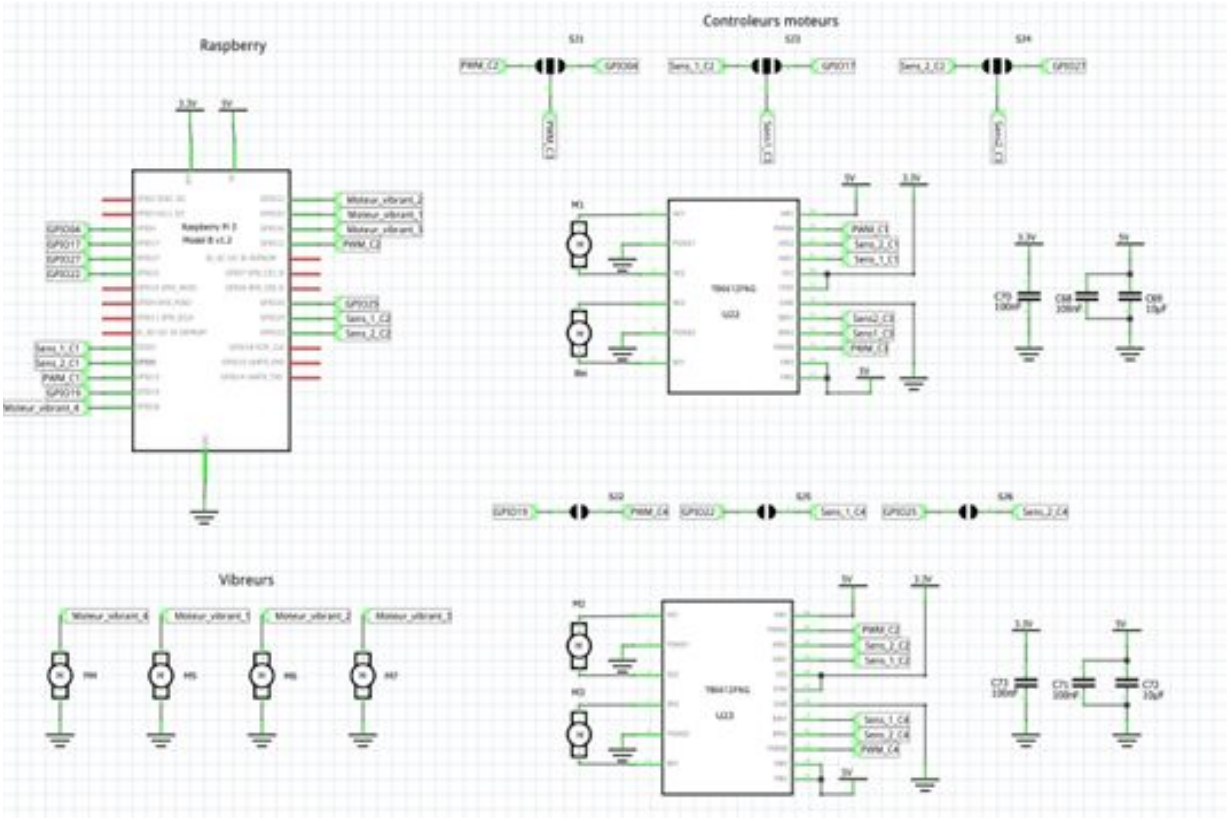
1.1. Conception du hat

Ainsi, la première étape fut de rechercher les composants adéquats, en commençant par les actionneurs. Nous avons choisi d'utiliser 3 motoréducteurs et 4 moteurs vibrants. Les motoréducteurs serviront à mettre en mouvement les doigts de la main. Les moteurs vibrants, quant à eux, seront disposés sur la main afin de représenter, chacun, une direction : devant, derrière, droite et gauche.

De plus, nous avons sélectionné le contrôleur moteur TB6612FNG. Ce dernier permet de contrôler deux moteurs. Les pins iIN1 et iIN2 permettent de choisir le sens de rotation du moteur. La PWM permet de réguler la vitesse de rotation. Nous retrouvons alors la commande sur les pins iO1 et iO2. L'alimentation de la partie puissance moteur doit être comprise entre 4.5V et 13.5V. L'alimentation de la partie commande doit être, quant à elle, comprise entre 2.7V à 5V. Le contrôleur moteur ne fonctionne que si /STBY est au niveau de logique 1.



Après réception des composants et un premier montage sur RPi 3 B+, nous avons décidé de créer une carte électronique qui viendra se fixer sur cette dernière. Ainsi, nous avons modélisé le hat sur le logiciel Fritzing.



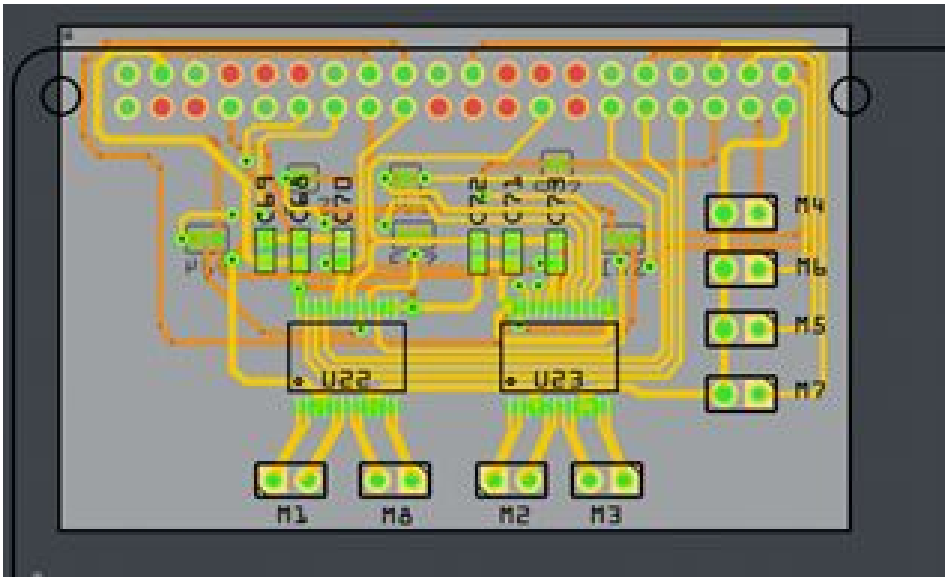
Vue schématique du hat sur Fritzing

Pour un gain de place le module TB6612FNG a été remplacé par le contrôleur TB6612FNG et des capacités en correspondance avec le diagramme d'application disponible sur la datasheet du contrôleur moteur. Ainsi, nous retrouvons les 4 moteurs vibrants et les motoréducteurs. A noter qu'un quatrième moteur a été ajouté pour offrir plus de possibilité pour d'autres projets. Les 4 motoréducteurs sont contrôlés par deux contrôleurs moteurs. Ces derniers seront reliés aux pins d'une RPi 4. L'attribution des GPIOs est la suivante :

Schéma de l'attribution des GPIOs sur une Raspberry Pi 4

Nous pouvons aussi remarquer la présence de 6 composants, ce sont des « solder jumper ». Ces derniers permettent de choisir, après l'impression de la carte, les liaisons désirées avec une simple soudure. Pour notre dispositif, nous choisirons de contrôler les moteurs M8 et M2 de la même façon.

Une fois la vue schématique créé, nous avons pu effectuer le routage sur deux faces.



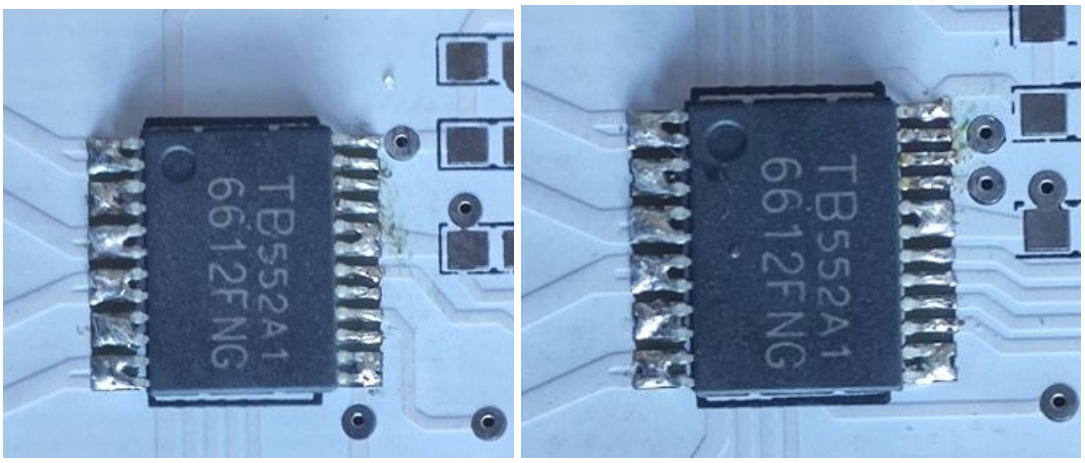
Vue du circuit imprimé du hat sur Fritzing

1.2. Soudure et test

Réalisation de la soudure

La partie soudure du hat était délicate du point du vue de la précision de soudure afin ne pas déborder sur les pins du même composant. Plusieurs essais ont dû être faits afin d'acquérir la technique de soudure d'un CMS. La soudure a été faite avec de l'étain contenant du plomb, celui-ci est plus facile à manipuler que de l'étain sans plomb.

Après plusieurs séances d'entraînement, les CMS sont soudés proprement. Le plus dur a été de souder les contrôleurs moteur. Il fallait être rapide pour ne pas trop chauffer le composant, mais aussi précis. Le résultat est le suivant :



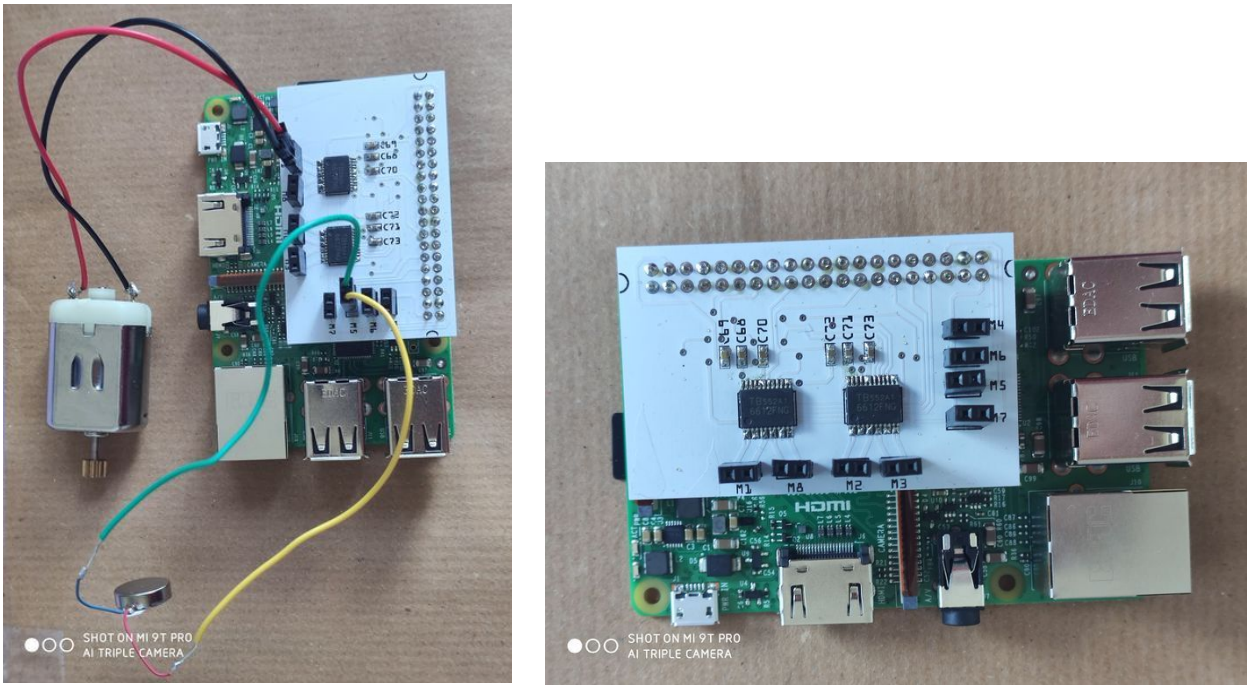
Contrôleurs moteur pour les moteurs M1, M8, M2 et M3

Les contrôleur étant soudés, l'ajout des condensateurs, des connecteurs pour les moteurs et moteurs vibrants, et les connecteurs pour la Raspberry ont été effectués.

On remarque la présence de ponts de soudure qui sont normaux car ils correspondent à des broches reliées ensemble sur le circuit.

On note que d'après le schématique vu précédemment, il faut faire des ponts de soudure sur l'arrière de la plaque pour mettre les moteurs M3 et M8 en service.

Grâce à un moteur DC et de vibreurs, le hat a pu être testé avec une Raspberry Pi 3 Model B, ainsi que d'autres matériels et composants.



Hat et Raspberry Pi 3

Tests de fonctionnement

La partie fonctionnement du hat consiste à implémenter un code de test dans la Raspberry Pi 3 et de vérifier si la carte fonctionne après son montage et la soudure. Le test a été fait avec un moteur afin de vérifier le premier contrôleur puis avec le deuxième moteur pour le second contrôleur.

Le paramétrage des moteurs est important dans le code. La librairie de commande des GPIO que nous avons utilisée avec la Raspberry Pi, nécessite non pas d'initialiser avec les numéros des pins de celle-ci, mais avec les numéros des GPIO qui se trouvent sur la Datasheet de la Raspberry. L'initialisation des moteurs avec les GPIO correspondants sont les suivants :

```
/* === PARAMETRES === */

#define VITESSE_DANGER_1  100
#define VITESSE_DANGER_2  200
#define VITESSE_DANGER_3  255

/* === GPIO === */
// Moteur M1
#define M1A 5   // Pin 29
#define M1B 6   // Pin 31
#define M1P 13  // Pin 33

// Moteur M2
#define M2A 24  // Pin 18
#define M2B 23  // Pin 31
#define M2P 12  // Pin 33
/*
// Moteur M8
#define M8A 24  // Pin 18
#define M8B 23   // Pin 31
#define M8P 12  // Pin 33
*/
// Moteur M3
#define M3A 22  // Pin 15
#define M3B 25  // Pin 22
#define M3P 19  // Pin 35

// Moteur M4 === Vibreur
#define M4  26  // Pin 37

// Moteur M5 === Vibreur
#define M5  20  // Pin 38

// Moteur M6 === Vibreur
#define M6  21  // Pin 40

// Moteur M7 === Vibreur
#define M7  16  // Pin 36
```

Lors du lancement du code il est nécessaire de préciser sudo avant l'exécution du programme.

Lorsque le programme est lancé sans sudo une erreur indique la non reconnaissance des fonctions GPIO utilisées dans le code.

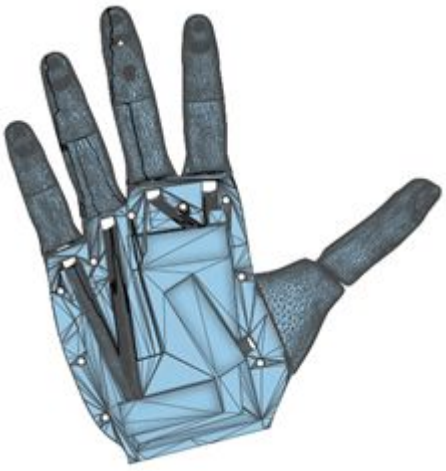
Ceci est une particularité de la librairie pigpio qui a été utilisée.

Certains problèmes ont été observés au niveau de la soudure. Lors du nettoyage de la panne sur une éponge humidifiée, la panne s'oxydait rapidement et devenait noire. Il devenait beaucoup plus difficile voire impossible de l'étamer. Il aura fallu recommencer plusieurs fois l'étamage de la panne pour pouvoir souder les composants. Le nettoyage de la panne s'est fait grâce à une éponge métallique qui a permis d'étaler l'étain sur la panne.

3. Modélisation et Impression 3D

Suite à notre rencontre avec Florian, nous avons décidé de conserver l'idée d'une forme de main pour notre dispositif. Ainsi, dans un premier temps nous avons cherché un modèle de main qui pourrait correspondre à notre cahier des charges en terme de confort et d'un point de vue fonctionnel. En effet, ce dernier devait, par exemple, être capable de se mettre en mouvement par le biais de moteurs, respecter une taille proche d'une main réelle, confortable à porter sur l'épaule, etc.

Une fois le modèle choisi, nous avons dû en modifier les fichiers pour l'adapter à nos motoréducteurs, mais aussi ajouter des emplacements pour les moteurs vibrants.



Dessus de la main

Paume de la main

Une fois le modèle de la main sélectionné, nous nous sommes penchées sur la structure qui soutiendra la caméra, la RPi 4 avec le hat et la batterie. Nous avons décidé de créer un harnais modulable, c'est-à-dire que chaque pièce est indépendante et pourra être retravaillée afin de s'adapter à d'autres versions de RPi, de caméra, batterie, etc. Pour une question de répartition des poids, nous avons décidé de positionner la RPI et la caméra au niveau du buste et la batterie au niveau du dos.

Afin d'adapter notre harnais à tout utilisateur, nous voulions que la position de la caméra soit réglable. Ainsi, nous avons commencé par créer une coque pour la caméra puis le bras qui la soutient. Ce bras sera fixé sur la plaque avant du harnais et possède 3 points de réglages.

Ensuite, nous avons modélisé la boite qui contiendra la RPi 4 et le hat, ainsi qu'un ventilateur.

Nous voulions que notre dispositif soit nomade, de ce fait nous disposons d'une batterie externe pour la source d'alimentation. Nous avons donc créé une autre boite pour la batterie, que l'on fixera à l'arrière du harnais.

Enfin, nous avons modélisé la plaque avant et arrière du harnais. La plaque arrière soutient la boîte de la batterie. La plaque avant, quant à elle, soutient le bras de la caméra et la boîte de la RPi.

Conclusion

Finalement, le semestre 8 a été la période de la réalisation des codes et du prototype dont les algorithmes et composants ont été décidés au semestre précédant. La mise en place de la partie software comprenant les codes du capteur et des actionneurs de la main a pu être testée dans un premier temps et s'est poursuivie à distance par la suite. Ce qui explique que l'ensemble des codes n'ont pas pu être testés simultanément. En parallèle, la création d'un hat a été réalisé pour avoir un dispositif plus compact. Pour finir, la modélisation du dispositif et l'impression du harnais a été effectuée.

La situation exceptionnelle que nous avons vécu lors de ce projet ne nous a pas permis de le finaliser correctement. Beaucoup de points et d'améliorations reposaient sur la phase de tests qui n'as pas eu lieu.

L'assemblage et les tests devront donc attendre la fin du confinement suite à quoi nous pourrons proposer à Florian et aux enfants dans une situation similaire à la sienne, un dispositif qui leur permettra d'avoir un plus d'indépendance et d'autonomie.

Nous souhaitons, bien évidemment, remercier les enseignants encadrants pour l'aide qu'ils nous ont fourni tout au long du projet. Ainsi que Florian et ses accompagnatrices, pour nous avoir donné l'opportunité de travailler sur un projet unique qui pourrait un jour aider des personnes en besoin.

Annexe

Script écrit des enregistrements audio :

Indications audio

- **Obstacle :**
  - **danger faible :**
    - **à gauche :** « Il y a un petit obstacle à ta gauche Florian. »
    - **à droite :** « Il y a un petit obstacle à ta droite Florian. »
    - **au centre :** « Il y a un petit obstacle en face de toi Florian. Décale-toi légèrement à gauche ou à droite. »
  - **danger intermédiaire :**
    - **à gauche :** « Il y a un obstacle à ta gauche. Décale-toi à droite »
    - **à droite :** « Il y a un obstacle à ta droite. Décale-toi à gauche »
    - **au centre :** « Il y a un obstacle en face de toi. Décale-toi à gauche ou à droite. »
  - **danger important :**
    - **à gauche :** « Attention Florian, il y a un obstacle à ta gauche. Arrête-toi et décale-toi à droite. »
    - **à droite :** « Attention Florian, il y a un obstacle à ta droite. Arrête-toi et décale-toi à gauche. »
    - **au centre :** « Attention Florian, il y a un obstacle en face. Arrête-toi et décale-toi bien à gauche ou à droite. »
- **Trou :**
  - **danger faible :**
    - **à gauche :** « Il y a un petit trou à ta gauche Florian. »
    - **à droite :** « Il y a un petit trou à ta droite Florian. »
    - **au centre :** « Il y a un petit trou en face de toi Florian. Décale-toi légèrement à gauche ou à droite. »
  - **danger intermédiaire :**
    - **à gauche :** « Il y a un trou à ta gauche. Décale-toi à droite »
    - **à droite :** « Il y a un trou à ta droite. Décale-toi à gauche »
    - **au centre :** « Il y a un trou en face de toi. Décale-toi à gauche ou à droite. »
  - **danger important :**
    - **à gauche :** « Attention Florian, il y a un trou à ta gauche. Arrête-toi et décale-toi à droite. »
    - **à droite :** « Attention Florian, il y a un trou à ta droite. Arrête-toi et décale-toi à gauche. »
    - **au centre :** « Attention Florian, il y a un trou en face. Arrête-toi et décale-toi bien à gauche ou à droite. »

Site utilisé pour le filtre de Sobel:

https://en.wikipedia.org/wiki/Sobel_operator