

UE Prototypage Industriel

Développement d'une expérience audio interactive par la création d'un boitier programmé pour envoyer sa position, son orientation et son accélération en temps réel

Rapport de projet

Auteur :

ISHIMARU Geoffrey  
*MA1 - Électronique*

A l'attention de :  
Mr. Le Vaillant  
Mr. Rodrigues  
Mr. Giot

# Table des matières

<b>Table des figures</b>	iii
<b>1 Présentation du projet</b>	1
1.1 Contexte . . . . .	1
1.2 Objectif . . . . .	1
1.3 Liste du matériel . . . . .	1
1.4 Choix de développement . . . . .	2
1.4.1 Programmation . . . . .	2
1.4.2 Conception 3D . . . . .	2
1.5 Méthode de travail . . . . .	2
<b>2 Récupération de la position en intérieur</b>	3
2.1 Introduction . . . . .	3
2.1.1 Technologie Ultra-Wide Band (UWB) . . . . .	3
2.1.2 Système Pozyx . . . . .	3
2.1.3 Protocole de communication MQTT . . . . .	3
2.2 Récupération des données . . . . .	4
2.2.1 Principe . . . . .	4
2.2.2 Câblage du tag . . . . .	4
2.2.3 Programmation . . . . .	4
<b>3 Récupération de l'orientation</b>	5
3.1 Introduction . . . . .	5
3.1.1 Présentation du capteur et précision de langage . . . . .	5
3.1.2 Protocole de communication SPI . . . . .	5
3.1.3 Câblage du BNO080 et du tag Pozyx au RPi . . . . .	6
3.2 Récupération des données . . . . .	6
3.2.1 Choix préalables . . . . .	6
3.2.2 Programmation . . . . .	7
<b>4 Communication des données</b>	10
4.1 Unification des 2 scripts . . . . .	10
4.2 Introduction à la transmission OSC en UDP . . . . .	10
4.2.1 Généralités sur l'OSC . . . . .	10
4.2.2 Généralités sur l'UDP . . . . .	11
4.3 Implémentation du protocole OSC . . . . .	11
<b>5 Conception 3D du boîtier</b>	12
5.1 Cahier des charges . . . . .	12
5.2 Idées et croquis . . . . .	12
5.2.1 Idée 1 . . . . .	12
5.2.2 Idée 2 . . . . .	13
5.2.3 Idée 3 . . . . .	14
5.3 Modélisation sur Fusion 360 . . . . .	15
5.4 Réalisation . . . . .	17
5.4.1 Première impression 3D . . . . .	17
5.4.2 Analyse critique et corrections . . . . .	19
5.4.3 Seconde impression 3D . . . . .	21
5.4.4 Gravure des supports internes pour PCB . . . . .	21
5.4.5 Assemblage final . . . . .	22

<b>6 Application de démonstration</b>	<b>24</b>
6.1 Outil de développement . . . . .	24
6.2 Objectif . . . . .	24
6.3 Démonstration "Proof of Concept" . . . . .	25
<b>7 Conclusion générale</b>	<b>26</b>
<b>Bibliographie</b>	<b>27</b>

# Table des figures

1.1	Spirale de développement rapide de prototypes [5] . . . . .	2
2.1	Schéma de la solution pour entreprise de Pozyx [4] . . . . .	3
3.1	Schéma de principe maître-esclave du SPI [14] . . . . .	6
3.2	Schéma de câblage du BNO080 à gauche et du tag Pozyx à droite au RPi . . . . .	6
3.3	Plan utilisé pour le développement de la librairie en Python pour le BNO080 . . . . .	8
5.1	Croquis de la première idée de design 3D . . . . .	12
5.2	Croquis de la deuxième idée de design 3D . . . . .	13
5.3	Croquis de dimensionnement du plateau interne . . . . .	14
5.4	Croquis du concept de la troisième idée de design 3D . . . . .	14
5.5	Croquis de la troisième idée de design 3D avec cotation . . . . .	15
5.6	Calculs préalables de dimensionnement des plateaux décentrés . . . . .	15
5.7	Sketch pour les extrusions et trous d'emboîtement de l'hémisphère . . . . .	16
5.8	Extrusions et trous d'emboîtement de l'hémisphère réalisés . . . . .	16
5.9	Résultat de l'hémisphère symétrique dans Fusion 360 . . . . .	16
5.10	Assemblage de 2 hémisphères symétriques dans Fusion 360 . . . . .	17
5.11	Photo durant l'impression d'une hémisphère en résine . . . . .	17
5.12	Photo du résultat de l'impression de l'hémisphère en résine (le support n'a pas encore été retiré) . . . . .	18
5.13	Photo à la fin de l'impression de l'hémisphère en PLA . . . . .	18
5.14	Photo de l'assemblage des 2 hémisphères symétriques . . . . .	19
5.15	Erreur avec le premier design imprimé . . . . .	19
5.16	Conséquence de l'erreur avec le premier design, coupe dans le boîtier fermé . . . . .	19
5.17	Design final corrigé, hémisphère symétrique (emboîtement possible avec lui-même) . . . . .	20
5.18	Design final corrigé, hémisphère asymétrique (emboîtement impossible avec lui-même) . . . . .	20
5.19	Coupe dans l'assemblage du design final pour vérification sur Fusion 360 afin d'éviter toute erreur . . . . .	20
5.20	Design (hémisphère symétrique) corrigé et imprimé en résine . . . . .	21
5.21	Modèle en résine dans le bain UV peu après son impression . . . . .	21
5.22	Vue virtuelle des 3 plateaux avec cartes électroniques . . . . .	21
5.23	Les 3 plateaux gravés dans le bois . . . . .	21
5.24	Assemblage des plateaux dans la sphère sans cartes électroniques . . . . .	22
5.25	Assemblage des plateaux dans la sphère avec cartes électroniques et batterie . . . . .	22
5.26	Vue sur le boîtier fermé avec tous les composants électroniques intégrés . . . . .	22
5.27	Intégration du boîtier dans une balle en mousse pour sa protection lors du lancé, vue entre-ouverte . . . . .	23
5.28	Intégration du boîtier dans une balle en mousse pour sa protection lors du lancé, vue finale . . . . .	23
6.1	Exemple de programme avec Puredata . . . . .	24

# 1. Présentation du projet

## 1.1 Contexte

Ce projet tire son origine dans le projet nommé "AHIA" initié par l'entreprise "*Demute.*" soutenue par le LAboratoire de Recherche en Arts et Sciences (LARAS) de l'Institut de Recherche de l'ISIB (IRISIB). Le travail de fin d'études de Sylvain Huraux a constitué la première base d'informations pour l'élaboration de ce projet [1]. "*Demute.*" est un studio de son mais aussi un centre de recherche et développement spécialisé dans les expériences audio innovantes [2]. Leur projet AHIA consiste à offrir une expérience de réalité augmentée avec un casque audio intégrant de la spatialisation 3D en temps réel.

Le casque en cours de développement pour le projet AHIA communique, pour ce faire, sa position et son orientation en temps réel. L'idée ici est de pouvoir élargir l'horizon d'applications des modules capables de donner leur position et leur orientation en temps réel car ils représentent un pilier fondamental pour interagir dans les expériences de réalité mixte.

## 1.2 Objectif

Le but de ce projet est de concevoir un boîtier indépendant capable de renvoyer en temps réel sa position et son orientation en intérieur. La conception se fera depuis la communication entre le cœur du boîtier et ses capteurs à la création d'une application de démonstration en passant par le design 3D du boîtier qui sera imprimé au laboratoire.

Le tracker du VIVE ("VIVE Tracker" [3]) est voué à un but similaire mais son mode de communication est propriétaire. Il fait donc partie de l'objectif de ce projet d'être libre sur le contrôle de la communication entre le boîtier développé ici (émetteur) et le récepteur. Cet objectif est le plus élaboré et sera réalisé ici.

Il serait également envisageable que le boîtier puisse fonctionner de manière totalement autonome produisant lui-même, par exemple, des sons et/ou lumières en fonction des informations rapportées par ses capteurs. Il suffirait alors de supprimer la partie d'envoi des données et de développer l'application finale directement dans le boitier.

## 1.3 Liste du matériel

Le choix du matériel a bénéficié des conclusions tirées pour le projet AHIA [1] principalement concernant le choix des capteurs de position et d'orientation. Pour le positionnement, le système de Pozyx a été choisi [4] et est déjà installé et fonctionnel au laboratoire. Pour l'orientation, une étude très poussée a été menée pour AHIA [1] et le choix final s'est porté sur le capteur BNO080 de Sparkfun. Le fonctionnement de ces capteurs sera développé aux points 2 et 3.

Dans le boîtier se trouveront :

- 1 Raspberry Pi (RPi) Zero W ;
- 1 Tag de Pozyx ;
- 1 Capteur BNO080 ;
- 1 Batterie 5V 5000mA ;

Il sera également nécessaire d'avoir dans la pièce des ancre Pozyx ; dans ce cas-ci, quatre ancre sont utilisées. Un serveur MQTT sera aussi nécessaire pour la communication de la position du tag sur le réseau Wi-Fi.

## 1.4 Choix de développement

### 1.4.1 Programmation

Pour l'uniformité du projet ainsi que pour le rendre plus accessible à de futurs développements, tous les scripts seront réalisés en Python (version 3.7). Ce langage simple a principalement été choisi pour faciliter la communication finale du boîtier avec une autre machine. En effet, il est très aisément de réaliser un serveur avec Python pour démarrer un échange de données avec d'autres appareils. De plus, Python est un langage très populaire et de nombreuses librairies sont disponibles, facilitant de nombreuses tâches.

Cependant, le choix de Python représentera un certain challenge dans ce travail pour la communication relativement bas niveau avec le capteur BNO080, normalement codée en C/C++.

### 1.4.2 Conception 3D

Pour la conception 3D du boîtier, le logiciel Fusion 360 de Autodesk a été choisi. Ce logiciel est gratuit pour les étudiants et ceux-ci peuvent demander et obtenir une licence aisément. Ses fonctionnalités sont très proches de logiciels professionnels tels que SolidWorks. Fusion 360 est donc parfaitement adapté à notre cas.

## 1.5 Méthode de travail

Le travail sera réalisé méthodiquement en séparant les tâches afin de les tester individuellement avant de les implémenter dans le projet complet. Cela est crucial lorsque le code de communication avec le BNO080 prendra de l'ampleur.

La spirale de prototypage rapide présentée à la figure 1.1 servira de référence pour les séquences de développement.

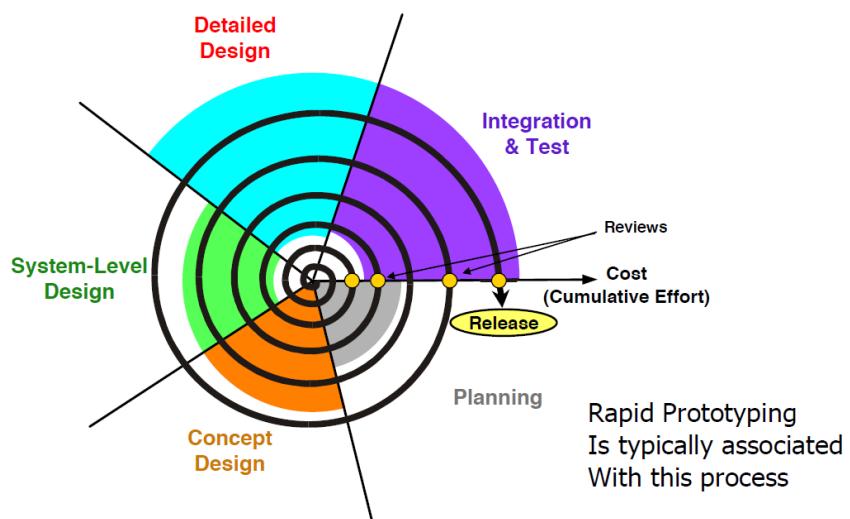


FIGURE 1.1 – Spirale de développement rapide de prototypes [5]

## 2. Récupération de la position en intérieur

### 2.1 Introduction

#### 2.1.1 Technologie Ultra-Wide Band (UWB)

La technologie Ultra-Wide Band consiste à utiliser des impulsions dont la largeur de bande vaut au minimum 500MHz sinon au minimum 20% de sa fréquence centrale [6]. L'intérêt de cette technologie pour ce projet est de pouvoir localiser notre boîtier avec une précision de l'ordre de 10cm. Cette précision est moindre qu'avec le système utilisé par le *VIVE Tracker*. Cependant, pour obtenir son niveau de précision le système du *VIVE Tracker* a besoin qu'au moins l'une de ses ancre "voie" constamment et en ligne directe le Tracker. L'UWB, lui, fonctionne à travers les parois. La balise UWB pourra donc être "cachée" dans le boîtier et l'aspect visuel de ce dernier n'est pas contraint.

La société Pozyx introduite au point 2.1.2 a choisi la technologie UWB car elle répond exactement aux besoins de localisation d'objets connectés dans de grands espaces intérieurs [7] : l'UWB nécessite peu d'énergie et est très peu influencé par le bruit électromagnétique ambiant.

#### 2.1.2 Système Pozyx

La société Pozyx fournit des solutions de localisation en intérieur (indoor) pour créateurs ("Creator series") et entreprises ("Enterprise solutions") [7]. Dans notre cas, le système en place au laboratoire correspond à la solution pour entreprises.

Ce système consiste à localiser les différentes balises (tags) grâce à un réseau d'ancres fixes (anchors), comme visible à la figure 2.1. Les balises émettent des signaux UWB. De par la distance entre chaque balise et les ancre, ces dernières reçoivent les signaux UWB à des moments différents. Les ancre sont toutes synchronisées à la passerelle ("Local gateway" sur la figure 2.1) et lui communiquent les identifiants des balises perçues ainsi que les temps mesurés à la réception des signaux UWB de chacune des balises. Sur base de ces informations, la passerelle est capable de calculer la position de la ou les balises perçues par les ancre. Cette technique de localisation (indépendante de l'UWB) est appelée TDOA (pour Time Difference Of Arrivals) [4].

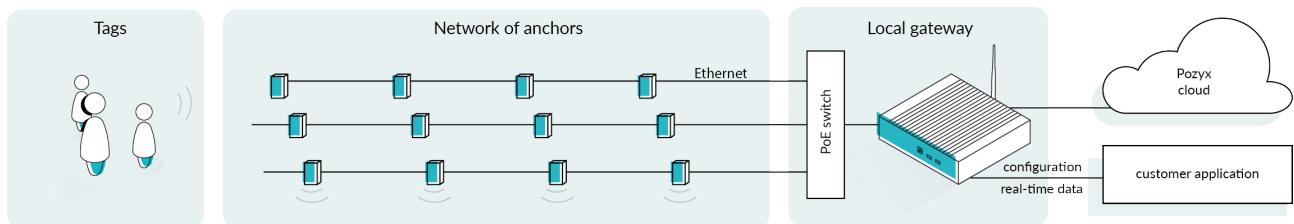


FIGURE 2.1 – Schéma de la solution pour entreprise de Pozyx [4]

La passerelle communique les résultats de ses calculs via son interface MQTT. Le protocole MQTT est introduit au point 2.1.3 et la récupération des données est décrite au point 2.2.

#### 2.1.3 Protocole de communication MQTT

MQTT signifie "Message Queuing Telemetry Transport" et tire son origine chez IBM [8]. Il s'agit d'un protocole de communication allégé utilisant le protocole plus connu TCP/IP. Il permet une com-

munication fiable et économe en énergie entre des appareils portables et autonomes comme des capteurs [9].

Son fonctionnement est basé sur un système de publication/souscription (*publish/subscribe* en anglais). Un "broker" désigne l'appareil, assimilable à un serveur, auquel des appareils appelés "clients" se connectent pour envoyer (mode publisher) et/ou récupérer (mode subscriber) des données. Un *publisher* doit préciser un "topic" associé à la donnée qu'il envoie au broker. De même, lorsqu'un *subscriber* se connecte au broker, il s'inscrit à un ou plusieurs topic et ne recevra que les données associées au(x) topic(s) en question. Les topics sont librement choisi par le développeur des clients.

Dans le cas du système Pozyx, les données utiles sont envoyées selon le format *.json* sur le topic "tags". Le broker est la passerelle (local gateway de la figure 2.1).

## 2.2 Récupération des données

### 2.2.1 Principe

Comme introduit au point 2.1.2, le code ici devra récupérer la position de la balise (tag dans le boîtier) en se connectant via WiFi à la passerelle (liée aux ancrès). La passerelle publie via le protocole MQTT (on l'appellera donc "broker") la position des balises perçues par ses ancrès.

### 2.2.2 Câblage du tag

Le tag Pozyx a seulement besoin d'être alimenté en 5V. Pour l'instant, il peut simplement être alimenté par un transfo 5V avec un câble micro-USB. Plus tard, ses 2 broches d'alimentation (5V et GND) seront utilisées. Le câblage complet (avec le BNO080) sera donné au point 3.1.3.

### 2.2.3 Programmation

Le code relatif à cette partie se trouve dans le répertoire *RPi\_Code\1.Tag\_Pozyx\_MQTT* du GitHub du projet [10].

Pour que le Raspberry Pi (RPi) Zero W connaisse sa position, il faut donc suivre les étapes globales suivantes :

1. Souscription au topic "tags" du broker MQTT (passerelle)
2. Récupération des données voulues dans le paquet reçu (format *.json*)
3. Mise à jour des données via callback pour le temps réel

De par le choix du langage de programmation Python, nous bénéficions d'une librairie simple pour la communication en MQTT. La librairie utilisée est *paho.mqtt.client* [11]. Pour la récupération des données dans le paquet *.json*, une librairie éponyme [12] existe aussi en Python. La manière de les récupérer les données de position ont été déterminées par observation des *.json* reçus.

À fin de développer des prototypes de code rapidement, le code a été écrit en suivant progressivement ces étapes :

- Implémenter la communication MQTT avec la librairie *paho.mqtt.client*
- Analyser la structure des paquets *.json* reçus
- Extraire les données voulues des paquets avec la librairie *json*

Chaque étape étant un cycle complet de la spirale de développement rapide (figure 1.1).

Par exemple, un paquet *.json* a été copié et un fichier Python dédié à son traitement a été réalisé pour tester le code indépendamment. Une fois fonctionnel, ce code a été implémenté dans le code principal. Tous les codes réalisés sont disponibles sur le GitHub du projet : [10].

### 3. Récupération de l'orientation

#### 3.1 Introduction

##### 3.1.1 Présentation du capteur et précision de langage

Le capteur utilisé ici est le BNO080 de Sparkfun. Il est composé d'un accéléromètre (3 axes), d'un gyroscope (3 axes) et d'un magnétomètre (3 axes). Il possède donc 9 degrés de liberté. Il contient également un microprocesseur 32-bit ARM Cortex-M0+ qui lui permet de traiter les données brutes de ses 3 capteurs internes [13]. Ainsi, il peut fournir des valeurs directement utilisables (comme par exemple son accélération linéaire en  $m/s^2$ ) via son interface SPI. Le fait que le BNO080 soit composé des 3 capteurs cités précédemment, et qu'il traite leurs valeurs brutes, fait de lui un AHRS (Attitude and Heading Reference System) [1].

De manière plus générale, on appelle IMU (Inertial Measurement Unit) ou encore "centrale inertielle" en français, des capteurs dotés d'un accéléromètre et d'un gyroscope. Dans l'étude réalisée pour le projet AHIA, il a été choisi de désigner le BNO080 comme un IMU (à 9 degrés de liberté) [1]. Pour éviter toute confusion de langage, ce choix sera gardé ici et dans le code.

##### 3.1.2 Protocole de communication SPI

SPI signifie Serial Peripheral Interface [14]. Comme son nom l'indique, il s'agit d'une interface de communication série entre différents périphériques, identifiés comme maître ou esclave. Comme sur la figure 3.1, on retrouve 4 bus de communication entre un maître et un esclave [14] :

- SCK = Serial Clock
- MOSI = Master Output / Slave Input
- MISO = Master Input / Slave Output
- CS = Chip Select

Il ne peut y avoir qu'un maître : il génère seul le signal d'horloge (SCK ou CLK) servant à la synchronisation de tous les périphériques. Le maître envoie ses données sur le bus MOSI. Les esclaves sur le bus MISO. Pour réveiller un esclave, le maître utilise le bus CS (un par esclave). Lorsqu'il y a plusieurs esclaves, cela permet aussi au maître de ne sélectionner et de ne communiquer qu'avec un seul esclave à la fois afin d'éviter les interférences sur le bus MISO [14]. Dans le cadre de ce projet, nous n'avons qu'un esclave SPI : le BNO080, le maître étant le RPi.

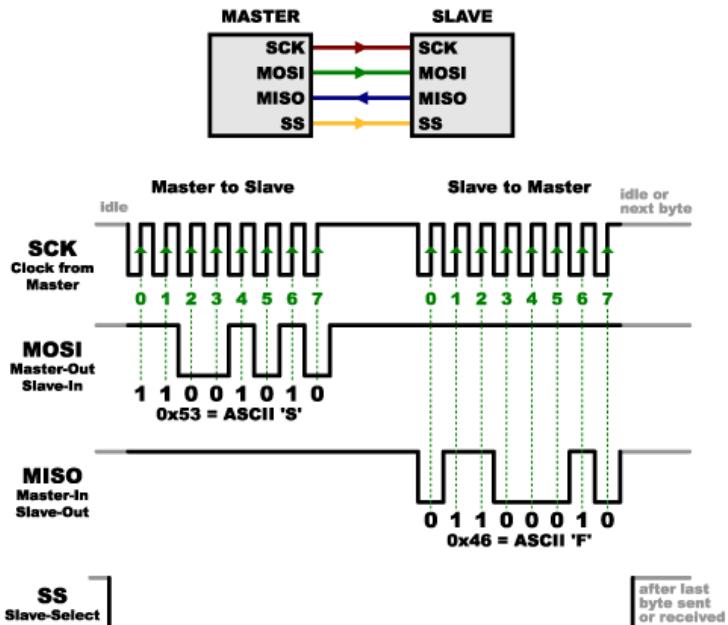


FIGURE 3.1 – Schéma de principe maître-esclave du SPI [14]

### 3.1.3 Câblage du BNO080 et du tag Pozyx au RPi

Le schéma de câblage du BNO080 et du tag Pozyx est donné à la figure 3.2. Ce schéma a été fourni par Guillaume Villée, chercheur à l'IRISIB travaillant sur le projet AHIA. Il n'a pas été nécessaire de modifier le schéma de câblage pour ce projet.

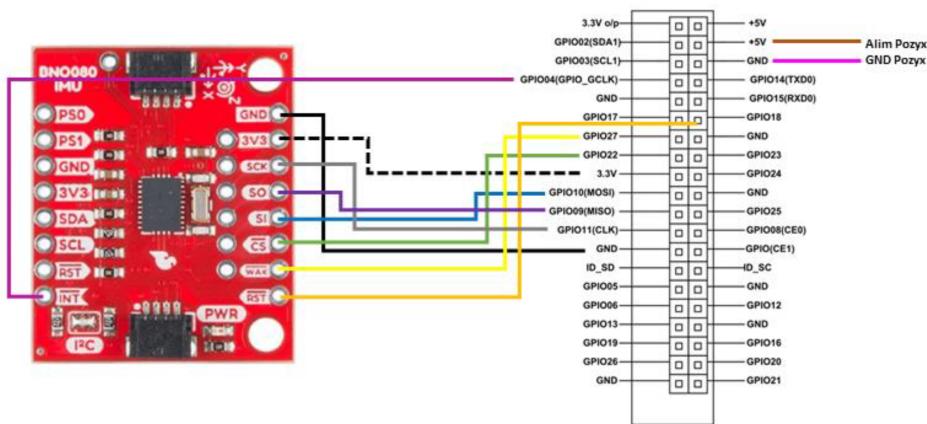


FIGURE 3.2 – Schéma de câblage du BNO080 à gauche et du tag Pozyx à droite au RPi

## 3.2 Récupération des données

### 3.2.1 Choix préalables

Un récapitulatif des moyens de communications possibles entre le RPi et le BNO080 a été établi :

1. Arduino  $\leftarrow$  I2C  $\rightarrow$  BNO (max 400kHz)
2. C++ + WiringPi  $\leftarrow$  SPI  $\rightarrow$  BNO (solution prise pour le projet AHIA)
3. Python + WiringPi  $\leftarrow$  SPI  $\rightarrow$  BNO
4. Python + Interfaçage GPIO et SPI  $\leftarrow$  SPI  $\rightarrow$  BNO

Le BNO080 est capable de communiquer jusqu'à 3MHz en SPI contre 400kHz en I2C [13]. Le premier élément de la liste ci-dessus a donc rapidement été mis de côté pour bénéficier d'un minimum de latence. Le second élément ne correspondait pas aux choix réalisés en terme de programmation : le projet a été restreint au Python uniquement. De plus, bien que WiringPi ait été une librairie très populaire pour la gestion des GPIO d'un Raspberry, cette librairie est maintenant dépréciée [15], surtout en Python (non officielle, l'officielle étant codée en C/C++). Cela donc écarté les solutions 2 et 3 de la liste. La solution restante consistait donc à réaliser l'interfaçage SPI entièrement en Python. Ce qui signifie également de réécrire la librairie permettant de gérer le BNO080 facilement.

L'avantage d'avoir choisi le langage Python est ici encore justifié pour la communication SPI et le paramétrage des GPIO du RPi. Il existe des librairies simples d'utilisation pour réaliser cela. Les librairies utilisées sont :

- spidev pour la communication SPI [16]
- RPi.GPIO pour la configuration des GPIO du RPi [17]

Il y a néanmoins des inconvénients majeurs :

- La réécriture d'une librairie (protocole SHTP, relativement bas niveau, à gérer avec la communication SPI pour le BNO080) est un travail de longue haleine
- La librairie spidev utilisée sert d'interface Python pour le driver du kernel Linux nommé également "spidev" (ce dernier est codé en C) [16]. On ne sait pas vraiment comment l'OS gère les instructions Python et la fréquence du SPI peut en être impactée. D'ailleurs, la fréquence générée par le driver est limitée à des divisions (par 2) d'une horloge cadencée à 250MHz. Comme la fréquence SPI du BNO080 s'élève à 3MHz, la fréquence la plus proche inférieure supportée par le driver est à la division d'horloge 128, soit 1,953MHz [18]. Sinon, la fréquence du RPi serait trop élevée pour le BNO080.

### 3.2.2 Programmation

Le code relatif à cette partie se trouve dans le répertoire *RPi\_Code\2.IMU* du GitHub du projet [10]. Les principales références (codes et documentations) utilisées s'y trouvent aussi dans le répertoire *RPi\_Code\2.IMU\resrc*.

Pour récupérer directement les données du BNO080 en SPI nous ne disposons cette fois, d'aucune librairie toute prête en Python. Comme expliqué au point précédent, la communication SPI devra être générée entièrement (avec envoi et réception de paquets SHTP [13] [19]). Cependant, une librairie a été éditée en C++ par Sparkfun (fournisseur du BNO080 utilisé). Cette dernière a été exploitée dans le cadre du projet AHIA et les codes de Nathan Seidle (Sparkfun), Guillaume Villée (IRISIB) et de Sylvain Huraux (ISIB/*Demute*. ) ont été pris comme référence. Ces dernières sont présentes sur le GitHub dans le répertoire annoncé ci-dessus. Le travail ici va consister à reproduire cette librairie et son utilisation, le tout en Python.

Écrire une librairie représente un travail de longue haleine et qui demande beaucoup de vigilance. Nathan Seidle (Sparkfun) a rigoureusement commenté sa librairie en C++ et l'exemple a été suivi au mieux pour sa transposition en Python. Afin d'éviter tout travail inutile et les erreurs, les fonctions strictement nécessaires au bon de la communication du BNO080 en SPI avec le RPi ont été déterminées.

Pour ce faire, la librairie en C++ de Sparkfun a été bien analysée : les fonctions indispensables ont été identifiées et comprises. Ensuite un plan de développement a été établi (voir figure 3.3). Ce fichier est présent sur le GitHub du projet [10].

La librairie de Sparkfun a également été relue d'un point de vue critique : maintenant que le langage utilisé est le Python, d'autres étapes pourraient être simplifiées. Par exemple, le code C++ possède une fonction "waitForSPI()" qui permet de savoir quand le BNO080 est prêt à délivrer des données [13]. La

librairie RPi.GPIO possède une fonction callback permettant de détecter des événements sur les pins paramétrées en input [20]. Par exemple, l'implémentation d'un fonction INT\_callback a été testée. Cette fonction lance la lecture de données avec la fonction dataAvailable() (pour plus de détails, voir le GitHub du projet [10]). Cette fonction est ajoutée aux événement de RPi.GPIO via cette ligne de code [20] :

```
GPIO.add_event_detect(bcm_INTPin, GPIO.FALLING, callback=INT_callback, bouncetime=1)
```

De cette manière, la fonction est exécutée à chaque front descendant sur la pin Interrupt du BNO080. Cependant, comme annoncé dans les inconvénients majeurs (au point 3.2.1) des problèmes liés à la gestion du script par l'OS ont été constatés : plusieurs callback INT\_callback avaient le temps de se produire avant que la fonction dataAvailable() ne soit terminée. Il en résultait une désynchronisation de la communication et un comportement général incontrôlé... La structure utilisée en C++ par Sparkfun a donc été conservée.

```
Fonctions_C_interessantes.txt - Bloc-notes
Fichier Édition Format Affichage Aide
Chemins intéressants pour target les fonctions à traduire impérativement:

Le but: pouvoir exécuter une fct similaire à imuFunc() dans IMUManager
imuFunc(); : "fonction récupérant les quaternions, exécutée sur un thread en permanence"

Fonctions primordiales dans cette imuFunc():
OK1. beginSPI
primordial pour initier la comm' SPI entre RPi et BNO080
Dedans on a besoin de :
OK-- waitForPinResponse (initialement nommé waitForSPI)
OK-- receiveSPIPacket (initialement nommé receivePacket)
OK-- sendSPIPacket (init nommé sendPacket)
OK-- printHeader (init nommé printPacket, moi je print que le header pcq le packet se print en une seule ligne de cmd)

2. dataAvailable --> pour récup les données
OK- parseInputReport
OK- parseCommandReport
OK- getQuatX pour X= I, J, K et Real
OK-- qToFloat(rawQuatX, rotationVector_QY)
OK- getQuatAccuracy()
OK- getMagAccuracy()

3.OK enableX --> pour que le BNO envoie les données
OK- X= Magnetometer
OK- X= RotationVector
OK-- pour les 2 : setFeatureCommand

4.OK calibration
-OK calibrateIMU()
--OK calibrateAll()
---OK sendCalibrateCommand
----OK sendCommand
--OK checkCalibration()
--OK saveCalibration()
--OK requestCalibrationStatus()
--OK calibrationComplete()
--OK endCalibration()

+ OK StartIMU()

5. reset //pour pouvoir recommuniquer post calibration
OK- softReset()
```

FIGURE 3.3 – Plan utilisé pour le développement de la librairie en Python pour le BNO080

Le plan présenté à la figure 3.3 est l'original ayant servi durant de la programmation. Le plan distingue des blocs cohérents à traduire en groupe. Dès lors, ces blocs pouvaient être testés les uns après les autres afin de trouver les éventuelles erreurs et les déboguer le plus rapidement et simplement possible. L'indentation des fonctions représente leurs imbrications réelles (fonction(s) appelant

d'autre(s) fonction(s)). La mention "OK" à gauche de l'indentation a été ajoutée à chaque fois que la fonction en question a été correctement implémentée. À nouveau, la spirale de prototypage rapide a été suivie (figure 1.1). Des commentaires avec références vers les sources d'informations (codes C++, manuel de référence, datasheet, ...) ont été ajoutés directement dans le code pour aider à sa bonne compréhension. Le tout est disponible sur le GitHub du projet [10].

Comme le montre la figure 3.3 , le nombre de fonctions à "traduire" en Python a été conséquent. De plus, beaucoup de notions supplémentaires ont dû être acquises (spécificités du C++, manière de coder une librairie utilisant le protocole SHTP, ...). La méthode de travail utilisée s'est révélée cruciale ici. Elle a permis de garder une vision globale de l'avancement du développement et des relations entre les nombreuses fonctions appelées. Grâce à cela, un maximum de temps et d'énergie ont été économisés. Le but étant également d'en faire bénéficier les futures personnes qui se pencheront sur ce travail.

Notons que la récupération de l'accélération linéaire a également été implémentée dans la librairie écrite ici pour le BNO080 en Python. Cela a été possible grâce à la compréhension du fonctionnement de la récupération des autres données. Les références principalement utilisées ont joué un rôle important dans cet acquis : la librairie Sparkfun (en C++) et le manuel de référence pour le BNO080 [19].

## 4. Communication des données

### 4.1 Unification des 2 scripts

Le code relatif à cette partie se trouve dans le répertoire *RPi\_Code\3.Unify* du GitHub du projet [10].

Selon la méthode de travail introduite au point 1.5, les scripts décrits aux points 2.2.3 et 3.2.2 ont été réalisés par étapes successives et donc indépendamment l'un de l'autre.

À présent, l'objectif est de pouvoir récupérer les 2 informations (position et orientation) en même temps. Les étapes suivies sont alors les suivantes :

1. Transformer le script récupérant la position en une classe python
2. Importer cette classe dans un nouveau script
3. Tester son fonctionnement dans le nouveau script en vérifiant que les données sont bien affichées dans la console
4. Transformer le script de l'IMU en une classe Python
5. Importer cette classe dans le nouveau script
6. Tester le fonctionnement parallèle de l'unification des 2 classes en vérifiant que les données des 2 classes sont correctement affichées dans la console
7. Stocker les variables qui nous intéressent dans le nouveau script afin de rendre facilement accessibles leurs dernières mises à jour
8. Implémenter une condition de mode Debug à tout affichage dans la console de variables non-expressément demandées
9. Simuler par une action (par exemple, appui sur une touche) une requête soumise au script : lorsque l'action est exécutée, le script doit simuler l'envoi (par affichage dans la console) des variables stockées

Une fois cette liste complétée, tout est prêt pour l'implémentation du protocole de communication voulu afin d'utiliser le boîtier de manière autonome.

Le dernier point a généré un problème lié au fonctionnement de Linux : la fonction callback permettant de récupérer les touches du clavier rendait trop lente la communication avec le BNO qui finissait par ne plus envoyer de données. Ce problème a été réglé par le choix d'envoyer les données en OSC via le protocole UDP. Il n'y a pas de requête reçue par le RPi : ce dernier envoie ses données tout le temps sur le réseau.

### 4.2 Introduction à la transmission OSC en UDP

#### 4.2.1 Généralités sur l'OSC

L'Open Sound Control (OSC) est un protocole de communication simple d'utilisation pour transmettre des données en temps réel. Initialement conçu pour transmettre données audio [21], il peut être utilisé dans bien d'autres domaines, comme celui de ce projet. L'avantage principal ici est de pouvoir envoyer des données associées à des noms ayant un style URL. Par exemple, un message contenant la position en x peut être envoyé avec le nom *position/x*, idem avec les valeurs y et z. Le récepteur pourra dès lors directement reconnaître et aiguiller la donnée reçue. Une autre facilité que propose le protocole est de grouper les messages à envoyer en un "bundle". Ainsi, pour reprendre l'exemple, il sera possible d'envoyer en une fois, les 3 messages portant les noms *position/x*, *position/y*, *position/z* avec leurs valeurs associées.

#### 4.2.2 Généralités sur l'UDP

L'UDP est l'abréviation de "User Datagram Protocol". Il s'agit aussi d'un protocole qui se veut léger et simple (version simplifiée du protocole plus connu TCP). Pour cela, il ne réalise aucune vérification au niveau de la qualité de la transmission [22]. On se contentera simplement d'envoyer les données le plus rapidement possible et de laisser gérer le récepteur.

### 4.3 Implémentation du protocole OSC

Le code relatif à cette partie se trouve dans le répertoire *RPi\_Code\4.OSC\_Implementation* du GitHub du projet [10].

La librairie utilisée pour envoyer des messages en OSC avec le RPi est osc4py3 [23]. Nous utiliserons surtout des bundles pour transmettre :

- la position selon les 3 axes : x, y, z → 3 messages,
- l'orientation sous forme angle d'Euler en degrés autour des 3 axes : x, y, z → 3 messages,
- l'accélération linéaire (en  $m/s^2$ ) aussi selon les 3 axes : x, y, z → 3 messages.

Les bundles envoyés contiennent donc 9 messages au total. La librairie utilisée permet l'envoi de tels bundles [24]. La réception de ces bundles, sera abordée au point 6.

# 5. Conception 3D du boîtier

## 5.1 Cahier des charges

Le cahier des charge se résume aux points suivants :

- Design sphérique
- Volume suffisant pour garder la batterie (et ses émissions électromagnétiques) à l'écart du BNO080 pour ne pas interférer avec son magnétomètre
- Lier le design du boitier à une application à inventer
- Boitier final suffisamment résistant pour pouvoir être lancé

Les modèles 3D créés dans cette partie se trouvent dans le répertoire *Conception\_3D* du GitHub du projet [10].

## 5.2 Idées et croquis

Pour répondre au cahier des charges, plusieurs idées de croquis ont été soumises. Elles seront présentées aux points suivants. Toutes ces idées visent à intégrer l'impression 3D dans une balle en mousse pour respecter le dernier point du cahier des charge.

### 5.2.1 Idée 1

La première idée montre la focalisation sur la façon dont la sphère pourrait tenir dans une balle en mousse. Pour cela des piques pour frottement ont été mises sur la surface externe. Des trous avaient aussi été prévus pour une aération minimale du boitier.

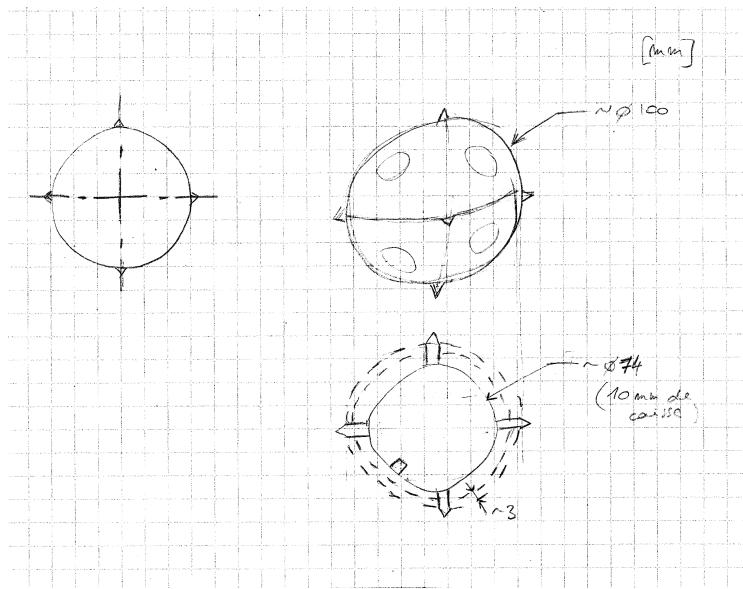


FIGURE 5.1 – Croquis de la première idée de design 3D

Ce premier jet d'un point de vue externe a rapidement été mis de côté pour se concentrer sur l'intérieur de la sphère. La problématique d'ouverture/fermeture de la sphère s'est rapidement révélée cruciale sur le design et ce premier n'offrait pas d'option réaliste pour y répondre.

### 5.2.2 Idée 2

Le système qui a été pensé ici s'est confronté à la problématique principale d'ouverture/fermeture de la sphère. Les hémisphères ont été pensées de manière symétrique afin qu'il n'y ait qu'un seul modèle 3D qui, imprimé en 2 exemplaires, s'emboiterait avec lui-même. Un système de pièces à emboiter a été pensé et mis sur papier. Le but était d'arriver à concentrer la complexité et les contraintes sur de petites pièces faciles à réimprimer et donc à réparer/remplacer.

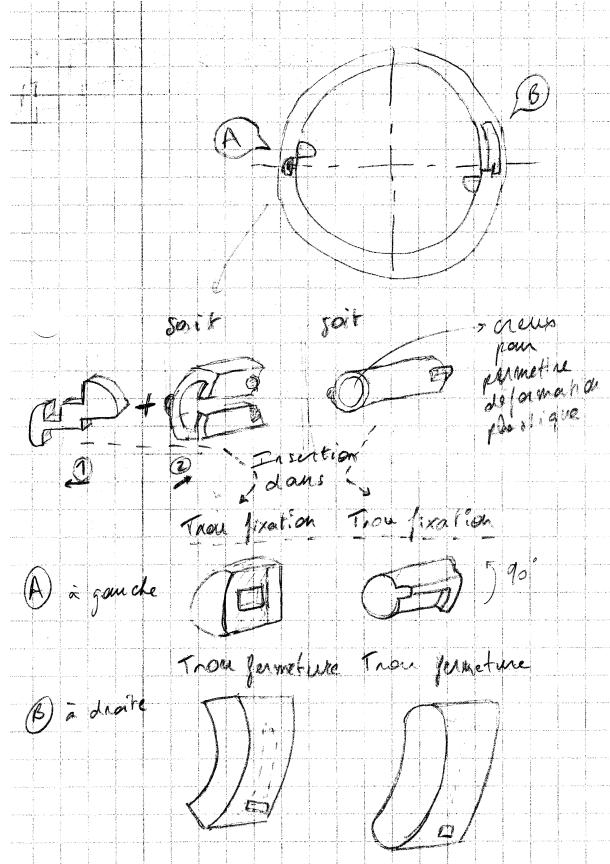


FIGURE 5.2 – Croquis de la deuxième idée de design 3D

Le problème fatal ici s'est imposé par les contraintes d'impression : des concavités telles que prévues pour y glisser les pièces ne sont pas réalisables. En effet, elles nécessitent d'imprimer du support à l'intérieur des concavités. Ce dernier aurait été très difficile à retirer.

De plus, augmenter le nombre de pièces n'est pas forcément une bonne idée d'un point de vue production. La modularité/réparabilité doit être mieux justifiée.

Du point de vue interne, l'idée de fixer les composants électroniques sur une plaque en bois (à l'aide d'entretoises) a été retenue. Il y a des plaques de 3mm d'épaisseur au laboratoire et elles peuvent y être gravées au laser. Un premier dimensionnement pour avoir une idée de l'encombrement maximal s'est fait via le croquis de la figure 5.3.

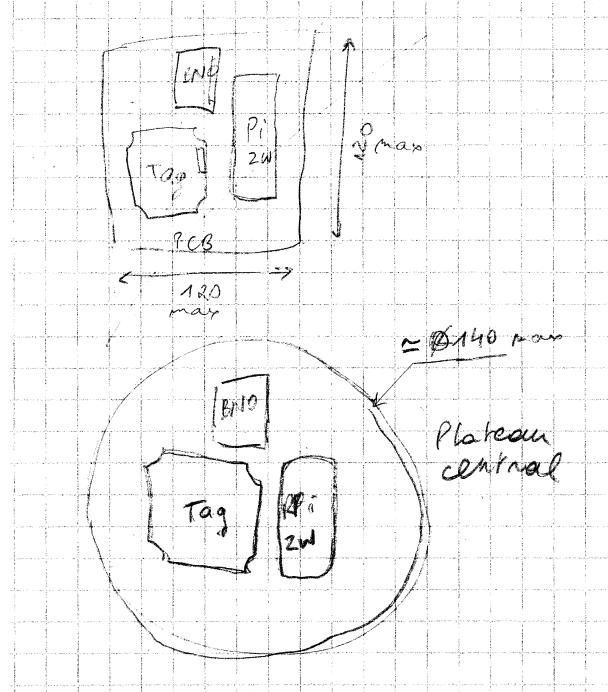


FIGURE 5.3 – Croquis de dimensionnement du plateau interne

### 5.2.3 Idée 3

Cette 3<sup>e</sup> idée est une révision de l'idée 2 afin qu'elle soit réalisable. Les éléments d'emboîtement ont directement été intégrés aux 2 hémisphères. L'idée de symétrie pour emboîter l'hémisphère avec une autre identique a été conservée. Les croquis sont présentés aux figures 5.4 et 5.5.

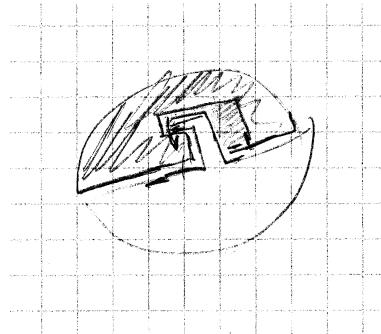


FIGURE 5.4 – Croquis du concept de la troisième idée de design 3D

Afin d'éloigner les composants pour respecter le cahier des charges, le plateau interne vu à la figure 5.3 a été allégé (bien que ses dimensions aient été préservées) : on y laissera uniquement la batterie et le RPi Zero W, chacun sur une face différente du plateau. On placera 2 autres plateaux supplémentaires : l'un accueillera le tag et l'autre le BNO080. Ces plateaux seront décentrés à une hauteur 'd' (figure 5.5). Cette hauteur a été calculée pour s'assurer que le tag (élément décentré le plus gros) soit sur un plateau suffisamment large pour l'accueillir (figure 5.6). Du point de vue fixation, les entretoises sont toujours de mise pour les cartes électroniques et pour la batterie, des "scratch" (Velcro) seront utilisés. Les mesures adéquates seront prises et un dimensionnement sur mesure pour les pré-trouer (à l'image des cartes électroniques) sera effectué sur Fusion 360.

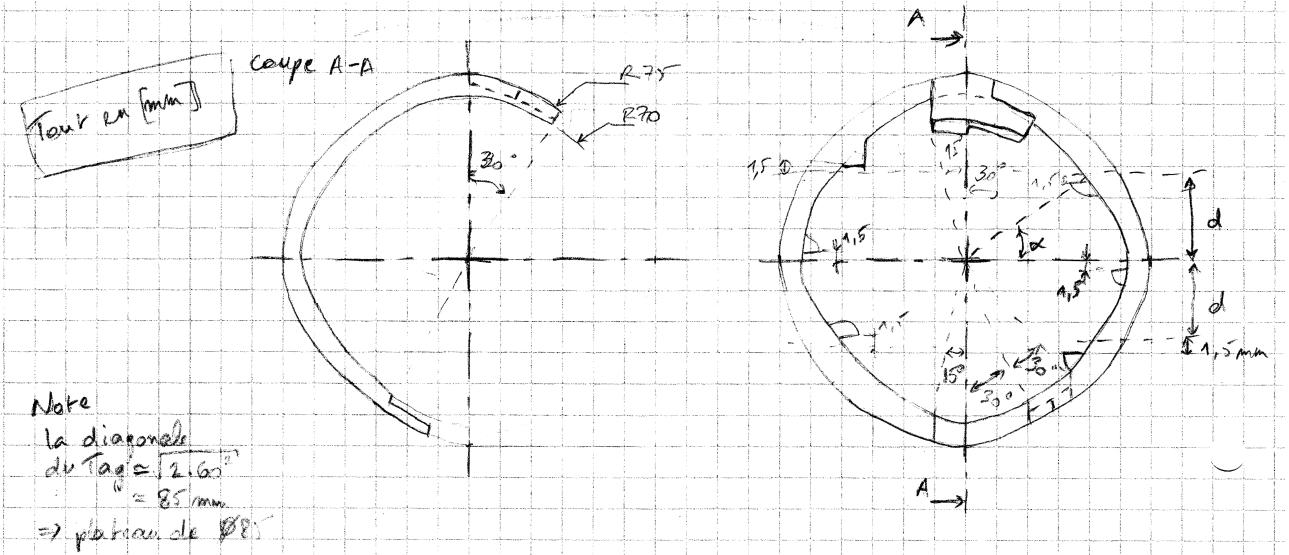


FIGURE 5.5 – Croquis de la troisième idée de design 3D avec cotation

$$\begin{aligned}
 &\Rightarrow \text{plateau de } \varnothing 8 \\
 &\Rightarrow \text{rapport : } \frac{85}{2} = 0,607 \\
 &\cos \alpha = 0,607 \\
 &\Leftrightarrow \alpha = 52,6^\circ \\
 &\Leftrightarrow \operatorname{tg} \alpha = 1,31 = \frac{d}{52,5} \\
 &\Leftrightarrow d_{\max} = 1,31 \cdot 52,5 = 55,6 \text{ mm} \\
 &\rightarrow d \text{ laissé } \% \text{ au modèle : } d = 53,5 \Rightarrow \sin \alpha = \frac{\sqrt{33,2}}{50} \Leftrightarrow \alpha = 49,84^\circ \\
 &\text{Contrainte} \\
 &\text{dans : minimum} \\
 &: d + 1,5 \\
 &\text{de + contrainte} \\
 &\Leftrightarrow \frac{10}{R} \geq 55 \\
 &\Leftrightarrow R = \frac{55}{10} = 5,5 \text{ mm} \\
 &\Leftrightarrow R = \frac{55}{\operatorname{tg} \alpha} = 43,3 \text{ mm} \Rightarrow \varnothing = 86,6 \text{ mm}
 \end{aligned}$$

FIGURE 5.6 – Calculs préalables de dimensionnement des plateaux décentrés

Ce modèle a été réalisé sur Fusion 360.

### 5.3 Modélisation sur Fusion 360

Concernant la méthode de modélisation, 3 sketchs ont été utilisés : le premier pour extruder un hémisphère. Le deuxième pour les extrusions et trous d'emboîtement -arcs de fixation- (figures 5.7 et 5.8). La dernière pour les supports des plateaux internes.

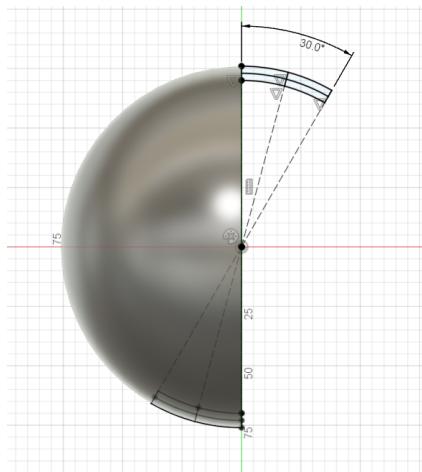


FIGURE 5.7 – Sketch pour les extrusions et trous d'emboîtement de l'hémisphère



FIGURE 5.8 – Extrusions et trous d'emboîtement de l'hémisphère réalisés

Le résultat des extrusions finale du dernier sketch est affiché à la figure 5.9.

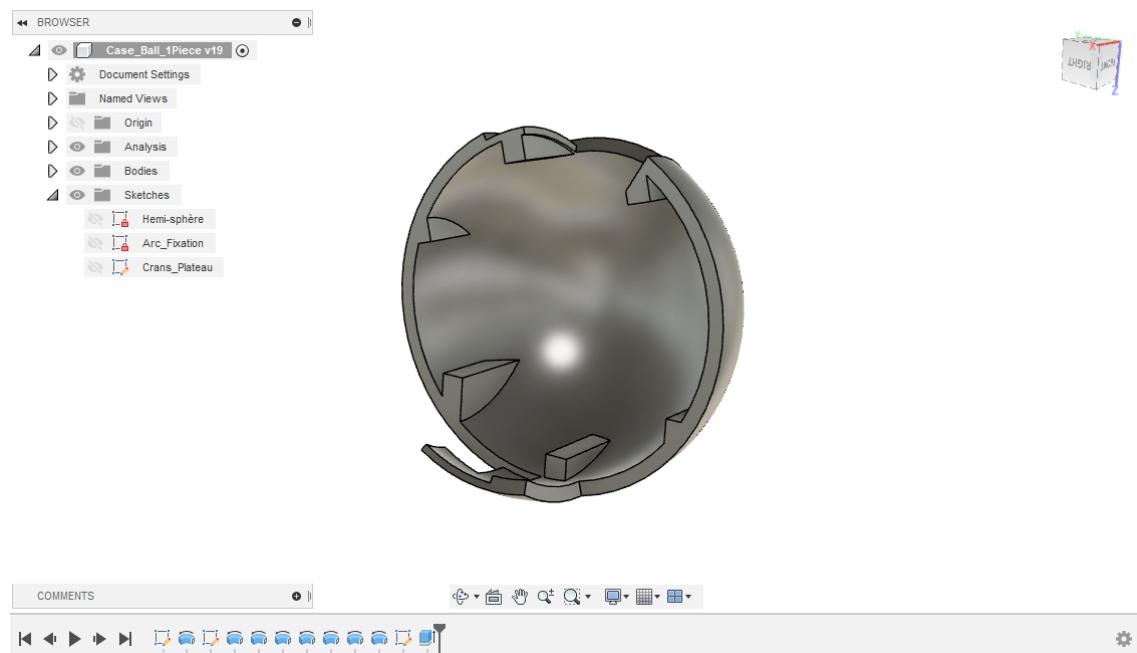


FIGURE 5.9 – Résultat de l'hémisphère symétrique dans Fusion 360

La figure 5.10 donne une prévisualisation de l'assemblage avec emboîtement.

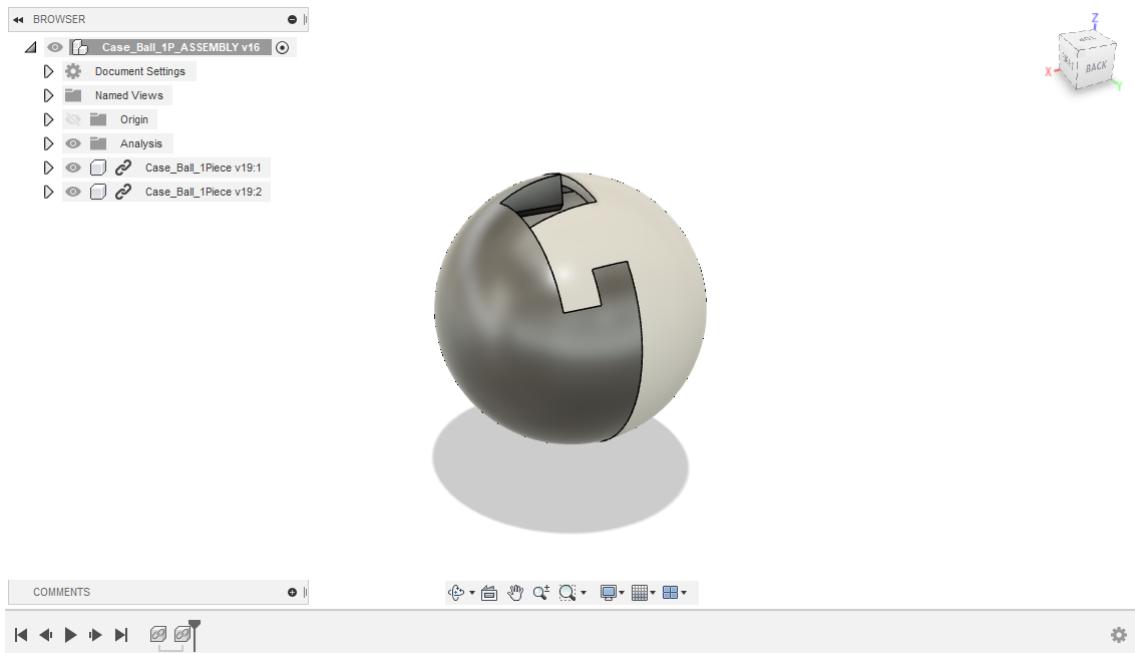


FIGURE 5.10 – Assemblage de 2 hémsiphères symétriques dans Fusion 360

## 5.4 Réalisation

### 5.4.1 Première impression 3D

L'impression 3D avec de la résine est assez particulière : un laser (vert sur la figure 5.11) vient polymériser la résine fluide dans le bac. La structure est construite de haut en bas.



FIGURE 5.11 – Photo durant l'impression d'une hémisphère en résine



FIGURE 5.12 – Photo du résultat de l'impression de l'hémisphère en résine (le support n'a pas encore été retiré)

L'impression en PLA est classique : on vient déposer des couches successives d'un filament de plastique (PLA) fondu. La pièce est construite de bas en haut.

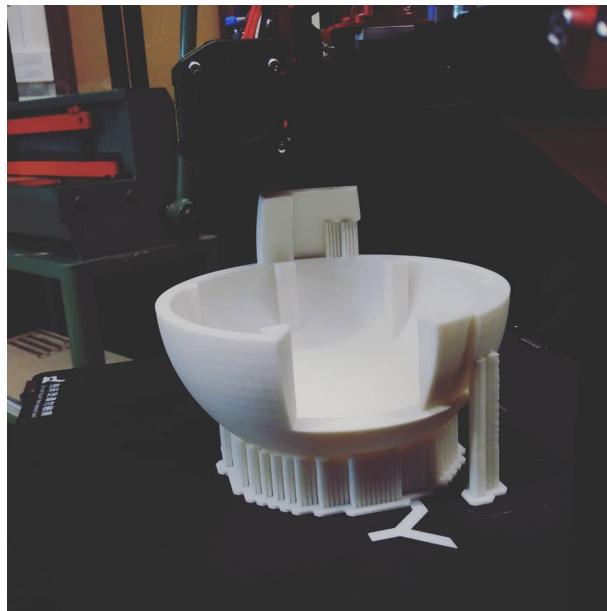


FIGURE 5.13 – Photo à la fin de l'impression de l'hémisphère en PLA

Après avoir retiré le support des 2 hémisphères, ces 2 dernières ont été assemblées. Le support imprimé pour le PLA n'a pas été jeté : une fois décollé, il fait office d'un excellent support sur mesure pour venir déposer l'assemblage. Le résultat est visible à la figure 5.14.



FIGURE 5.14 – Photo de l'assemblage des 2 hémisphères symétriques

#### 5.4.2 Analyse critique et corrections

Une erreur de dimensionnement a été constatée après impression : un décalage de  $15^\circ$  n'a pas été pris en compte dans la rotation de l'emboitement. Ce qui a eu une répercussion fatale sur le glissement des plateaux. Le problème est montré explicitement aux figures 5.15 et 5.16 suivantes.

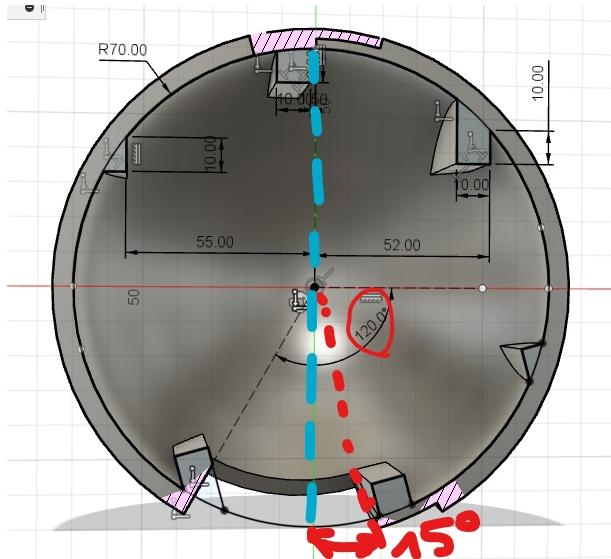


FIGURE 5.15 – Erreur avec le premier design imprimé

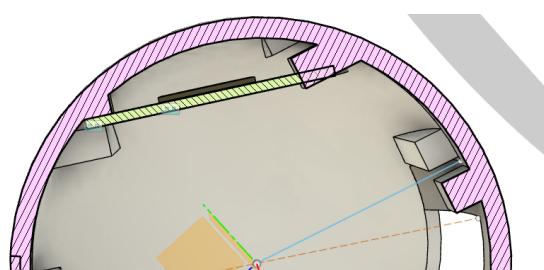


FIGURE 5.16 – Conséquence de l'erreur avec le premier design, coupe dans le boîtier fermé

Cette erreur a été rapidement corrigée sur Fusion, il suffisait de changer une cote d'angle (figure 5.15  $120^\circ \rightarrow 135^\circ$ ) et d'appliquer la correction aux supports de plateaux... D'autres modifications ont été apportées dans le but d'améliorer la résistance mécanique et la facilité d'utilisation. Pour la résistance mécanique, des congés ont été ajoutés aux angles droits situés sur l'extrusion qui sert à l'emboîtement. Pour la facilité d'utilisation lors de la fermeture de la sphère, la parfaite symétrie a été sacrifiée pour avoir l'une des 2 hémisphères avec des doubles encoches. Ainsi, les plateaux ne pourront pas bouger lors de la fermeture de la sphère. Les 2 nouveaux modèles d'hémisphère sont présentés aux figures 5.17 et 5.18.

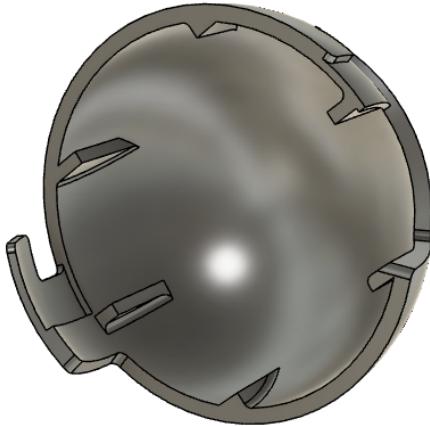


FIGURE 5.17 – Design final corrigé, hémisphère symétrique (emboîtement possible avec lui-même)

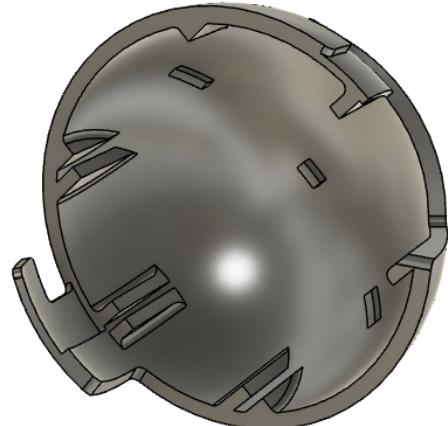


FIGURE 5.18 – Design final corrigé, hémisphère asymétrique (emboîtement impossible avec lui-même)

L'assemblage des 2 a été complètement re-vérifié sur Fusion (avec plateaux et PCB, comme sur la figure 5.19) pour s'assurer qu'il n'y aurait plus de problème à l'impression suivante. Étant donné la taille du modèle et le temps imparti pour réaliser le prototype, il n'est pas idéal de faire plus de cycles de prototypage (en lien avec la méthode de travail, figure 1.1).

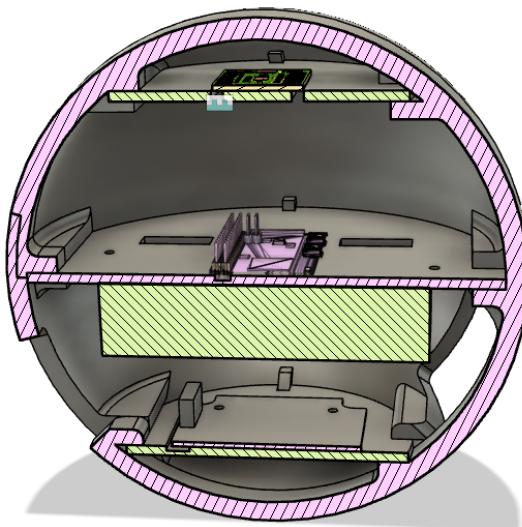


FIGURE 5.19 – Coupe dans l'assemblage du design final pour vérification sur Fusion 360 afin d'éviter toute erreur

#### 5.4.3 Seconde impression 3D

Notons que la résine est durcie dans un bain UV pendant plusieurs dizaines de minutes après son impression. La résine utilisée est maintenant incolore.



FIGURE 5.20 – Design (hémisphère symétrique) corrigé et imprimé en résine

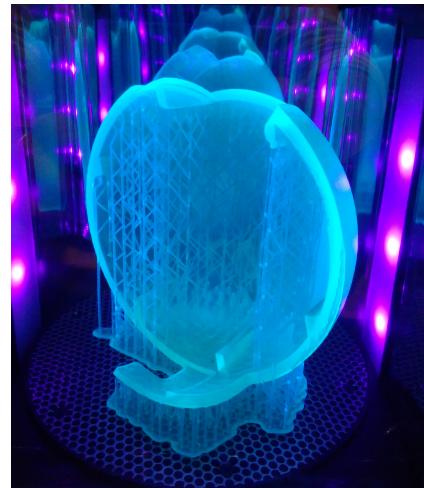


FIGURE 5.21 – Modèle en résine dans le bain UV peu après son impression

L'hémisphère asymétrique a été imprimé en PLA blanc.

#### 5.4.4 Gravure des supports internes pour PCB

Comme aperçu à la figure 5.19, les plateaux ont également été modélisés et assemblé. Un visuel supplémentaire est donné à la figure 5.22.

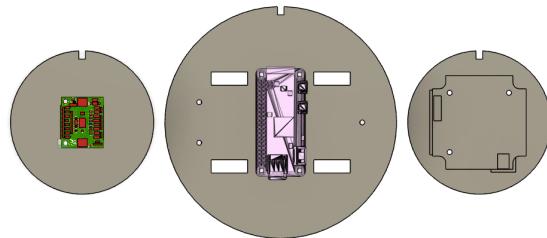


FIGURE 5.22 – Vue virtuelle des 3 plateaux avec cartes électroniques

Ils ont été gravés dans du bois de 3mm d'épaisseur au labo. Le résultat est donné à la figure 5.23.

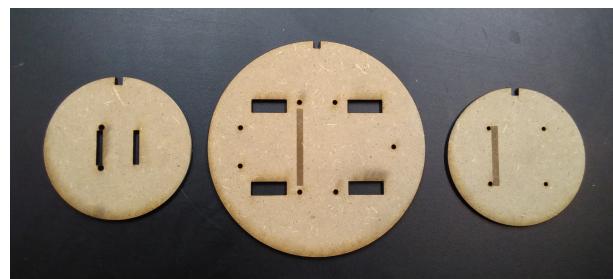


FIGURE 5.23 – Les 3 plateaux gravés dans le bois

Quelques coups de lime et de dremel ont été apportés pour assurer un emboîtement parfait avec les hémisphères et les cartes électroniques.

#### 5.4.5 Assemblage final

La figure 5.24 montre les plateaux calés dans leurs encoches respectives et la figure 5.25 montre le résultat avec les cartes électroniques utilisées et la batterie.



FIGURE 5.24 – Assemblage des plateaux dans la sphère sans cartes électroniques

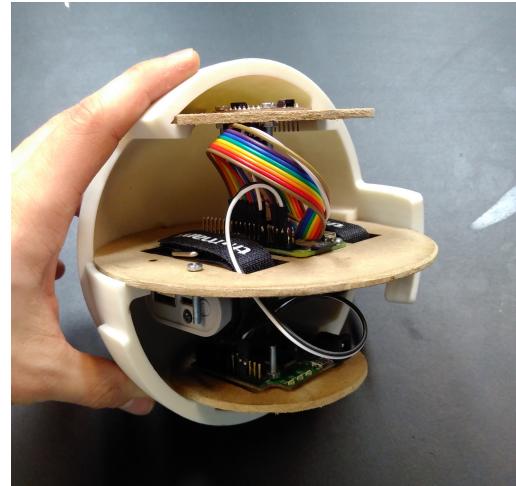


FIGURE 5.25 – Assemblage des plateaux dans la sphère avec cartes électroniques et batterie

La figure 5.26 donne une idée du boitier lorsqu'il est fermé.



FIGURE 5.26 – Vue sur le boitier fermé avec tous les composants électroniques intégrés

Finalement, pour respecter la possibilité de lancer le boitier (prévue par le cahier des charges), le boitier a été intégré dans une balle en mousse. Le résultat final est montré aux figures 5.27 et 5.28.



FIGURE 5.27 – Intégration du boitier dans une balle en mousse pour sa protection lors du lancé, vue entre-ouverte



FIGURE 5.28 – Intégration du boitier dans une balle en mousse pour sa protection lors du lancé, vue finale

# 6. Application de démonstration

## 6.1 Outil de développement

L'application de démonstration tirera parti des données récupérées avec tout le travail réalisé aux points 2, 3 et 4. Elle sera développée sur PureData, un logiciel de traitement sonore et graphique. PureData permet de développer à travers une interface de programmation graphique utilisant des noeuds.

La version Extended de Puredata possède beaucoup de librairies. Notamment une nommée "mr-peach" lui permettant d'envoyer et de recevoir des messages en OSC [26]. Cette librairie est même capable de traiter des bundles (paquets contenant plusieurs messages) : exactement ce dont nous avons besoin. La figure 6.1 donne un exemple visuel d'un programme réalisé dans Puredata. Il s'agit du programme de démonstration en cours de développement.

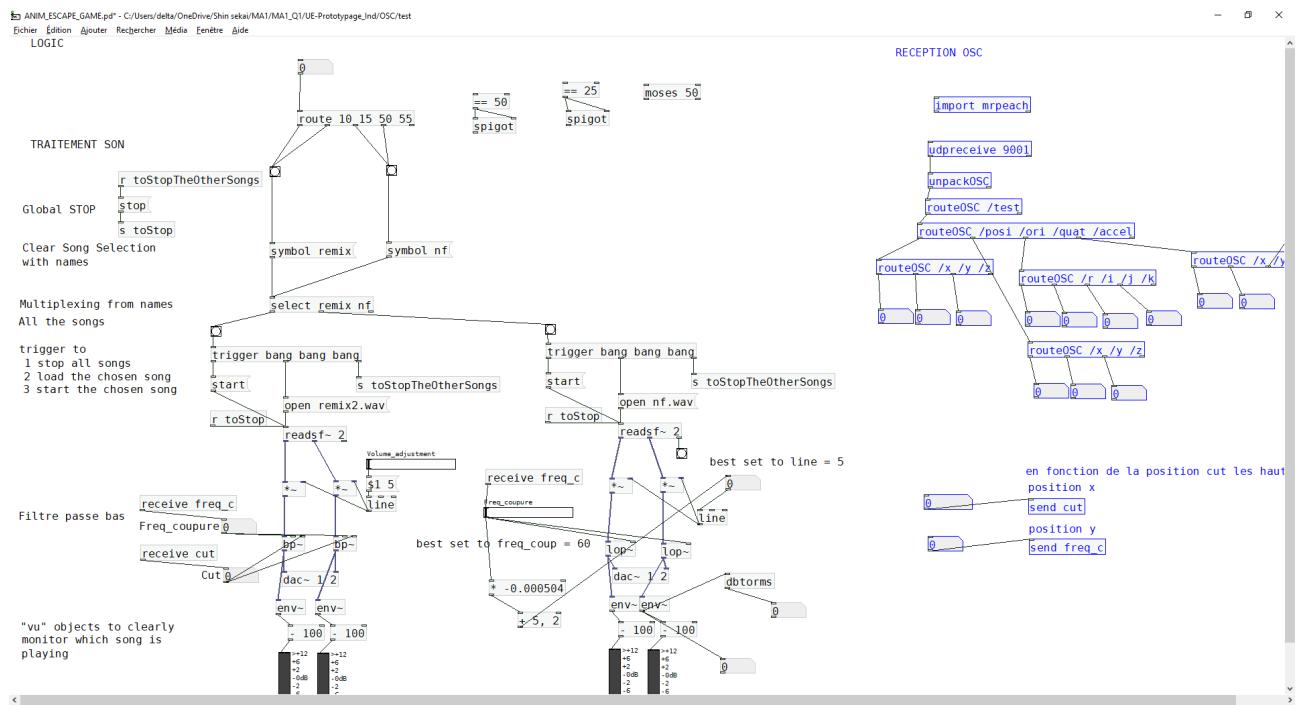


FIGURE 6.1 – Exemple de programme avec Puredata

## 6.2 Objectif

Le but de la démonstration est d'établir un "Proof of concept" en démontrant le potentiel du boîtier développé. L'idée est de montrer que le boîtier est utilisable pour des applications de réalité mixte. Pour cette démonstration, il a été choisi de créer une expérience interactive liant la spatialisation du boîtier et des effets sonores. Ce genre d'expérience pourrait parfaitement s'intégrer dans un jeu de type "Escape Game" mais ce n'est qu'un exemple qui sera pris ici. Les possibilités d'application sont presque illimitées.

### 6.3 Démonstration "Proof of Concept"

Lorsque le boîtier est pris en main, une accélération est détectée (seuil géré dans PureData) et l'animation se lance avec un avertissement sonore. Ensuite, la personne tenant le boîtier est amenée à se déplacer dans le labo. En fonction de sa position, des effets sonores sont déclenchés. La personne est amenée à se rendre à des endroits précis dans le laboratoire. Le fait de se trouver à l'endroit demandé enclenche une nouvelle interaction : des instructions ou explications orales sont données. La personne sera alors amenée à jouer avec l'orientation du boîtier pour moduler un filtre appliqué à de la musique jouée en direct. Cela étant fait, la personne est amenée à réaliser une dernière action (ramener le boîtier sur son socle initial) mettant fin à la démonstration.

La démonstration sera effectuée lors de la présentation du projet. Le fichier PureData (.pd) et les fichiers sonores nécessaires à la présentation se trouvent dans le répertoire *PC\_PureData\DEMO\_FINAL* du GitHub du projet [10].

La visualisation en temps réel des données reçues (quaternions et accélération linéaire) a montré que celles-ci prennent parfois des valeurs incohérentes : quaternions hors de l'intervalle [0,1] et accélération linéaire sautant à plus de 200 lorsque les valeurs oscillent autour de 0. Ce problème (seul restant) est sans doute lié à la gestion des types de données (fixedPointValue → float) par Python. Il fera l'objet d'une vérification supplémentaire après la fin de ce travail. Tant qu'il ne sera pas réglé, le problème sera mis en évidence dans le code par un "# TODO" (mis à jour au chemin *RPI\_Code\4.OSC\_Implementation\BNO080\_SPI.py* du GitHub du projet [10]).

## 7. Conclusion générale

Ce projet très complet a permis d'aborder énormément de notions différentes en utilisant de la méthode de travail en spirale de prototypage rapide (cfr. point 1).

Une librairie pour exploiter le BNO080 directement en Python a été totalement écrite dans ce langage (cfr. point 3). Ce travail d'ampleur fut très enrichissant d'un point de vue programmation et sur l'utilisation concrète de manuels très techniques décrivant la manière de communiquer avec un capteur AHRS. Cet acquis a permis d'implémenter en plus la récupération de l'accélération linéaire (absente dans les codes reçus du projet AHIA à titre d'exemple). Beaucoup de protocoles de communication ont été utilisés et assimilés avec leur librairie respective en Python : le MQTT, le SPI couplé au SHTP ou encore l'OSC couplé à l'UDP (cfr. points 2, 3, 4).

Des designs 3D complexes ont été élaborés sur papier et réalisés sur Fusion 360. Ils ont ensuite été réellement imprimés, en résine et en PLA pour y intégrer les cartes électroniques qui ont été précédemment programmées. Des erreurs ont été commises et ont été corrigées. Cela a encore permis d'exercer la spirale du cycle de prototypage et d'améliorer le design. Le cahier des charges a été respecté (cfr. point 5).

Le logiciel Puredata a été pris en main pour réaliser une expérience audio interactive sur base des données envoyées par le boîtier réalisé. La démonstration a été faite lors de la présentation du projet (cfr. point 6).

Une dernière mise à jour du code sera nécessaire pour corriger les valeurs de quaternions et d'accélération linéaire parfois incohérentes. Celle-ci sera sans doute apportée peu de temps après la fin de ce projet (cfr. 6).

# Bibliographie

- [1] Sylvain HURAU. *Tests et implémentation de centrales inertielles et d'un système de gestion d'alimentation autonome pour un casque audio immersif binaural.* ISIB, 2019.
- [2] Demute. Site web officiel de *Demute*.  
<http://www.demute.studio/drupal/>.
- [3] Vive. Site web officiel de *Vive*.  
<https://www.vive.com/us/vive-tracker/>.
- [4] Pozyx. Site web officiel de *Pozyx* - Leur offre produit *Pozyx Enterprise*.  
<https://www.pozyx.io/product-info/enterprise>.
- [5] Olivier de Weck. Engeineering Design and Rapid Prototyping.
- [6] IEEE Computer Society Sponsored by the LAN/MAN Standards Committee. *IEEE Standard for Low Rate Wireless Network - version : 802.15.4<sup>TM</sup>-2015*. The Institute of Electrical and Electronics Engineers, Inc., 2015.
- [7] Pozyx. Our products.  
<https://www.pozyx.io/product-info>.
- [8] IBM. Introduction to IBM MQ.  
[https://www.ibm.com/support/knowledgecenter/SSFKSJ\\_8.0.0/com.ibm.mq.pro.doc/q001020\\_.htm](https://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.pro.doc/q001020_.htm).
- [9] IBM. IBM MQ Telemetry Transport format and protocol.  
[https://www.ibm.com/support/knowledgecenter/SSFKSJ\\_8.0.0/com.ibm.mq.ref.doc/q049190\\_.htm](https://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.ref.doc/q049190_.htm).
- [10] Geoffrey ISHIMARU. UE Prototypage - Répertoire GitHub de tous les codes.  
[https://github.com/Geoff55555/UE\\_Protypage](https://github.com/Geoff55555/UE_Protypage).
- [11] Roger Light. (librairie python) paho-mqtt 1.5.0.  
<https://pypi.org/project/paho-mqtt/>.
- [12] Bob Ippolito. (librairie python) json.  
<https://docs.python.org/2/library/json.html>.
- [13] Hillcrestlabs. *BNO08X Data Sheet v1.6*. Hillcrest Laboratories, Inc., April 2019.
- [14] Mike Grusin. Serial peripheral interface (spi).  
<https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all#introduction>.
- [15] Gordon. wiringPi - deprecated...  
<http://wiringpi.com/wiringpi-deprecated/>, August 6, 2019.
- [16] Stephen Caudle. (Librairie Python) spidev.  
<https://pypi.org/project/spidev/>.
- [17] Ben Croston. (Librairie Python) RPi.GPIO.  
<https://pypi.org/project/RPi.GPIO/>.
- [18] Raspberry Pi Documentation. SPI.  
<https://www.raspberrypi.org/documentation/hardware/raspberrypi/spi/README.md>.
- [19] Hillcrestlabs. *SH 2 Reference Manual v1.2*. Hillcrest Laboratories, Inc., 05/19/2017.
- [20] RasPi.TV. How to use interrupts with Python on the Raspberry Pi and RPi.GPIO – part 3.  
<https://raspi.tv/2013/how-to-use-interrupts-with-python-on-the-raspberry-pi-and-rpi-gpio-part-3>.
- [21] opensoundcontrol.org. Introduction to OSC.  
<http://opensoundcontrol.org/introduction-osc>.
- [22] ipv6.com. UDP – User Datagram Protocol.  
<https://www.ipv6.com/general/udp-user-datagram-protocol/>.

- [23] osc4py3.readthedocs.io. Docs - 1. Usage.  
<https://osc4py3.readthedocs.io/en/latest/userdoc.html>.
- [24] osc4py3.readthedocs.io. Docs - 2. Messages and Bundles.  
<https://osc4py3.readthedocs.io/en/latest/msgbund.html>.
- [25] Puredata Manual. Chapter 2 : Theory Of Operation.  
<https://puredata.info/docs/manuals/pd/x2.htm>.
- [26] Floss Manuals. Open Sound Control (OSC) -for Puredata-.  
<http://write.flossmanuals.net/pure-data/osc/>.