UMONS
Université de Mons

Faculté Polytechnique

Rapport du projet

Localisation du son en temps-réel

US-B3-SCINFO-016-M Traitement du signal

Réalisé par Roméo IBRAIMOVSKI & Simon MICHEL

POLYTECH MONS

Supervisé par Hugo BOHY

3e Bachelier en Sciences Informatiques
Année 2022-2023

ACADÉMIE UNIVERSITAIRE WALLONIE-BRUXELLES

Table des matières

Introduction **1**

1. Système hors ligne **3**
1.1 Génération des données et ensemble de données 3
1.2 Mise en mémoire tampon 5
1.3 Pré-traitement 6
1.3.1 Normalisation 6
1.3.2 Déséchantillonnage 7
1.4 Corrélation croisée 9
1.5 Localisation 10
1.5.1 Différence de temps d'arrivée (TDOA) 10
1.5.2 Système d'équation de localisation 10
1.6 Précision et rapidité des systèmes 10

2. Localisation en temps réel **11**
2.1 Équipement matériel 11
2.2 Acquisition et traitement des données 11

Conclusion **11**

Introduction

Ce rapport porte sur le projet, supervisé par Mr.Hugo Bohy, du cours de Traitement du Signal donné par le Prof. Thierry Dutoit.

Le but de ce projet est de donner aux étudiants une intuition de l'importance des systèmes de traitement du signal en temps réel, ainsi que d'implémenter un système capable de traiter des signaux audio afin de localiser directionnellement un son émit à proximité, c'est à dire de donner l'angle avec lequel le son est émit dans le plan par rapport au système.

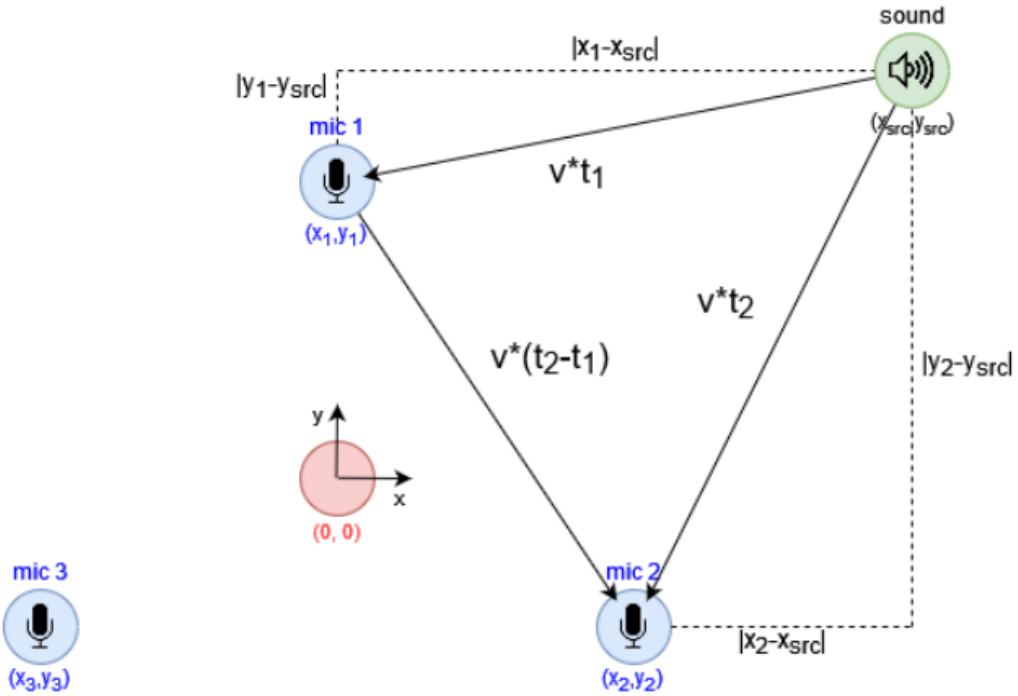

Figure 1 – Illustration du propos.

Remarque : cette image provient de l'énoncé du projet.

Afin de résoudre cette problématique, nous avons écrit, en Python 3, un code exécutable sur un Raspberry PI 3B connecté à un système de microphones circulaire (voir Figure 2).

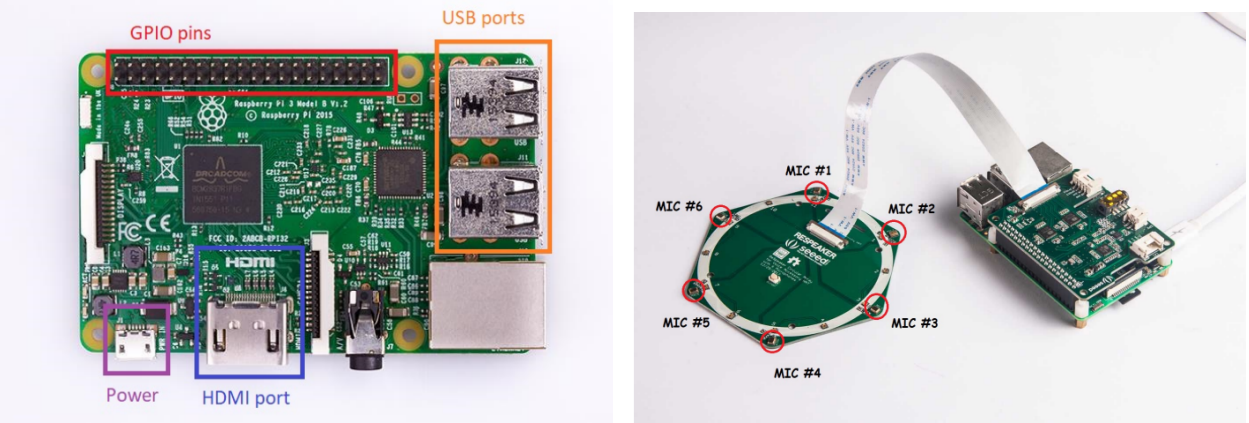

FIGURE 2 – Un Raspberry PI 3B et le système de microphones connecté à celui-ci.

Remarque 1 : le système de microphones est le Respeaker 6-Mic Circular Array Kit from Seeed Studio.
Remarque 2 : ces images proviennent de l'énoncé du projet.

Le principe utilisé afin de localiser les sons est la "Différence de Temps d'Arrivée" (plus tard parfois référencé comme TDOA, de l'anglais *Time Difference Of Arrival*).
Cette technique, également utilisée dans les sonars pour ne citer qu'un exemple, consiste à mesurer la différence de temps de détection du signal par différents microphones synchronisés et de calculer ainsi la position de sa source.

Méthodologie générale

Lors du développement du code, celui-ci sera d'abord testé *offline* sur des fichiers pré-enregistrés, auxquels sont joints les données vérifiées à reproduire avec notre application, afin de mesurer la précision et la rapidité d'exécution de l'application.
Ensuite, une fois les tests susmentionnés concluants, le code sera exécuté *online* sur le système matériel de la Figure 2 afin de vérifier son fonctionnement en temps réel.

Afin de faire les tests *offline* du code, celui-ci sera séparé en différentes fonctions semi-indépendantes afin de traiter séparément chaque étape et aspect du travail à effectuer.

1. Système hors ligne

Cette partie du projet concerne l'implémentation de chaque fonction séparément et de leur enchaînement afin de former un système capable de répondre à la problématique ainsi que l'évaluation de ses performances.

1.1 Génération des données et ensemble de données

La première étape de ce projet consiste à produire des signaux connus afin d'obtenir des résultats prévisibles.

D'une part nous avons écrit une fonction (*create_sine_wave*) afin de produire des signaux de manière contrôlée. Afin de la tester nous avons simplement suivi les consignes de l'énoncé et ensuite affiché le signal.

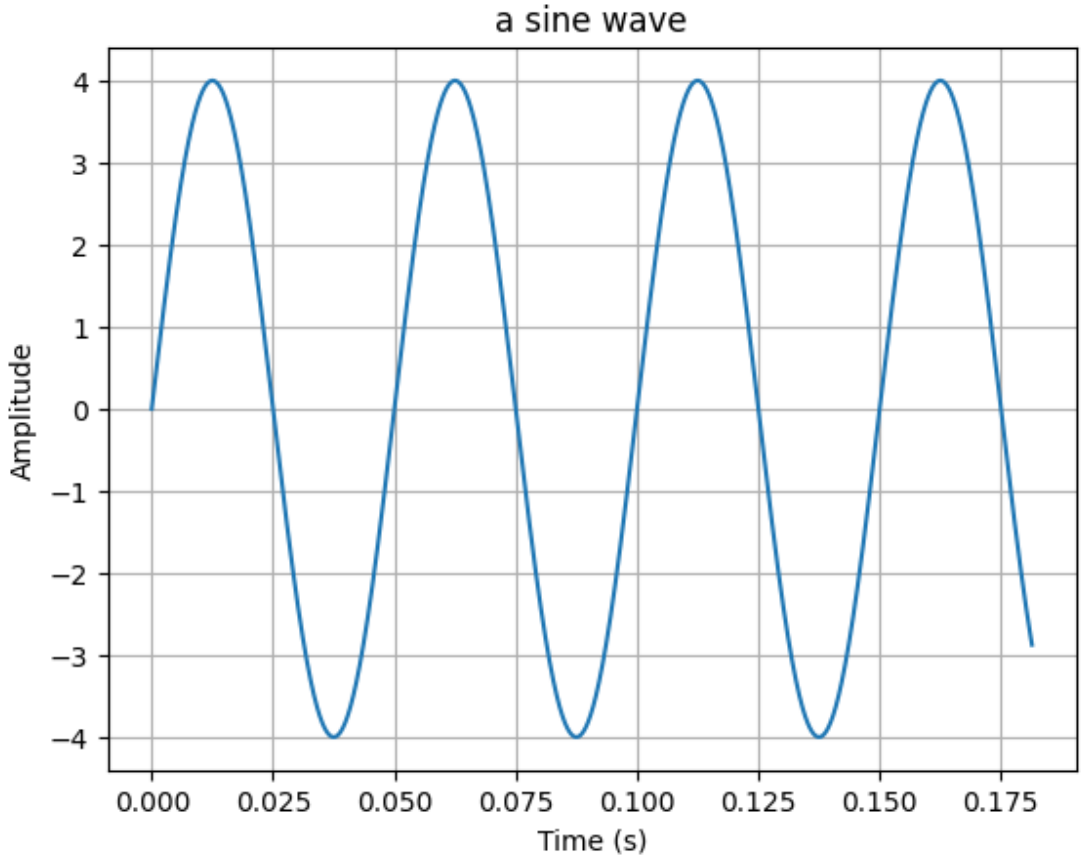

FIGURE 3 – Signal sinusoïdal de $A_{pp} = 8$, f = 20Hz et N = 8000 à $f_s = 44{,}1$ kHz créé pour la section 1.1 du projet

D'autre part nous avons écrit une fonction (*read_wavefile*) pour lire les fichiers contenant les sons pré-enregistrés.

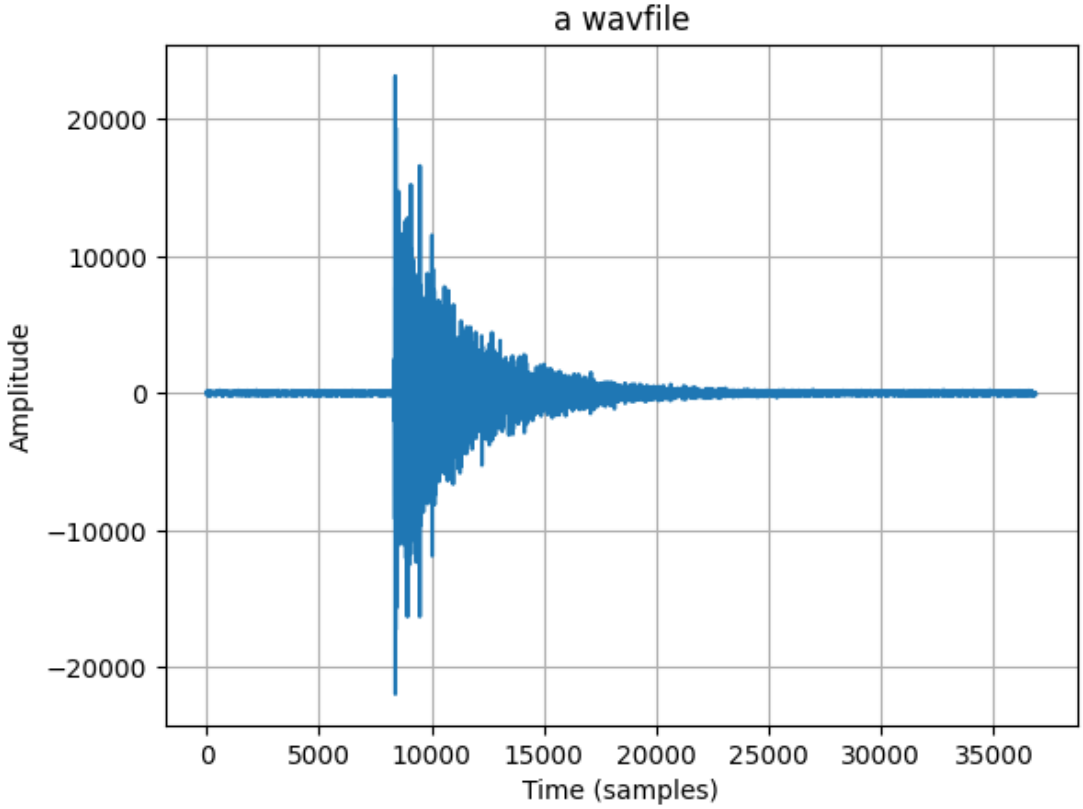

FIGURE 4 – Signal lu depuis un fichier pour la section 1.1 du projet.

Remarque : l'axe des abscisses représente le temps en nombre d'échantillons écoulés.

1.2 Mise en mémoire tampon

Pour gérer les données enregistrées en continu par un microphone, nous utilisons une mémoire tampon pour stocker ces données et les traiter ultérieurement.

Comme nous n'avons pas l'utilité de conserver les données traitées, nous utilisons plus particulièrement un tampon circulaire (ou *ringbuffer* en anglais) qui est un type de tampon de taille limitée dans lequel nous remplaçons les échantillons les plus anciens par de nouveaux échantillons. Lorsqu'un certain nombre d'emplacements ont été mis à jour, nous traitons à nouveau le tampon.

Pour créer un tampon circulaire, nous avons écrit une fonction (*create_ringbuffer*) qui utilise la fonction *deque* des collections de la bibliothèque Python, pour créer un objet *deque* qui se comporte comme un tampon circulaire.

Pour vérifier l'efficacité de notre implémentation, nous avons créé un tampon circulaire avec une longueur maximale de 750 et utilisez la fonction *extend* pour sauvegarder un par un les échantillons de notre signal. Ensuite, nous avons affiché le contenu du tampon lorsqu'il atteint sa taille maximale, puis lorsque le 1000ème échantillon a été sauvegardé. Et nous avons bien obtenu les résultats attendus.

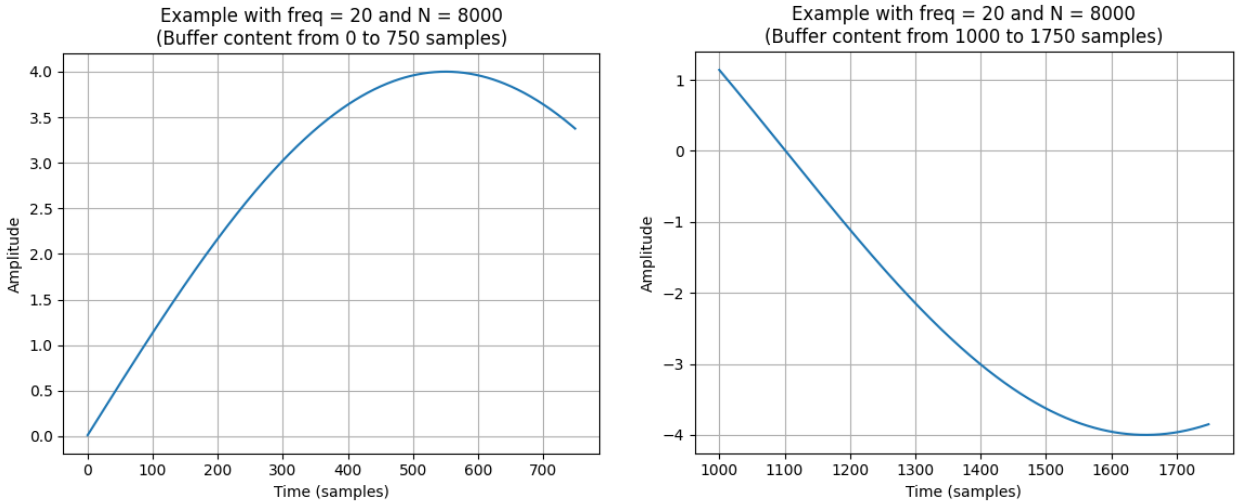

FIGURE 5 – Exemple de vérification pour un signal sinusoïdal de $A_{pp} = 8$, f = 20Hz et N = 8000 à $f_s = 44{,}1$ kHz.

Remarque : l'axe des abscisses représente le temps en nombre d'échantillons écoulés.

1.3 Pré-traitement

Avant d'utiliser les signaux, il est nécessaire de les pré-traiter, c'est-à-dire d'effectuer un travail préliminaire de standardisation sur le signal.

1.3.1 Normalisation

L'étape de normalisation du signal consiste à ramener l'amplitude sur un intervalle [-1 ; 1]. Pour cela, il suffit de diviser toutes les valeurs du signal par sa valeur la plus grande (en valeur absolue).

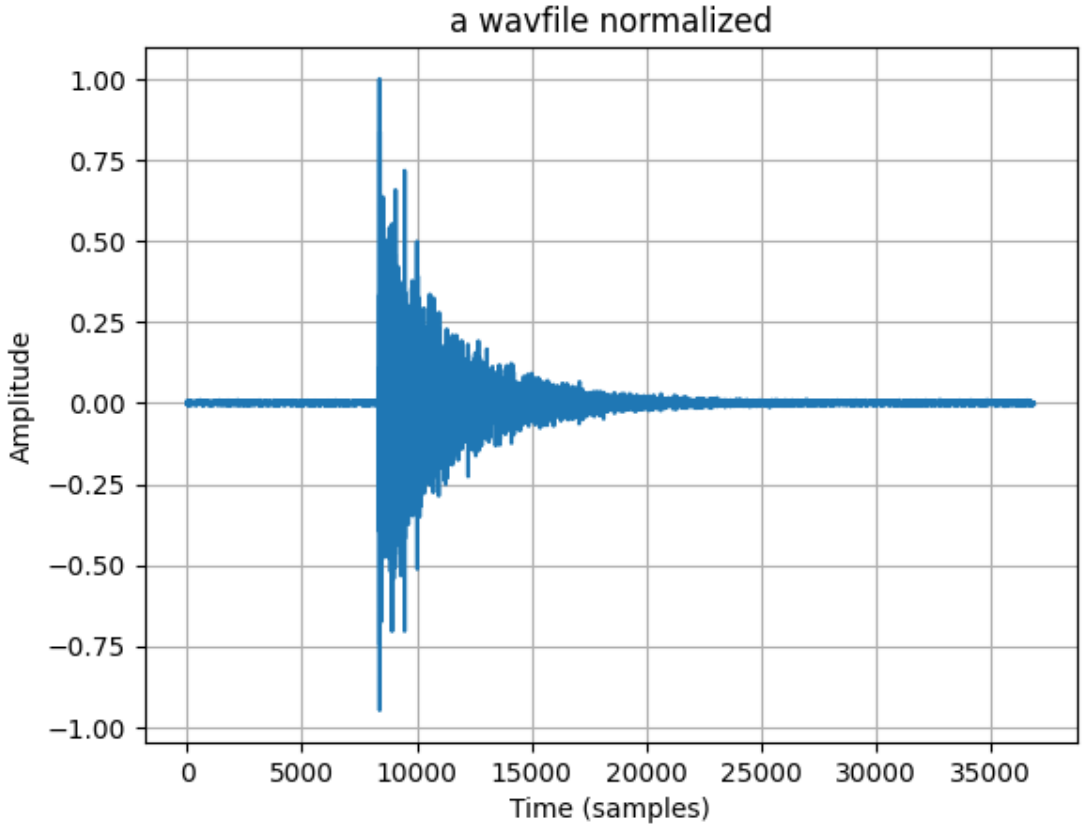

FIGURE 6 – Signal de la Figure 3 de la Section 1.1 normalisé.

Remarque : l'axe des abscisses représente le temps en nombre d'échantillons écoulés.

1.3.2 Déséchantillonnage

L'étape suivante, le déséchantillonnage (ou *downsampling* en anglais), s'est avérée la plus fastidieuse lors du développement de notre projet.

Dans un premier temps, il nous a été nécessaire d'analyser le signal via un spectrogramme.

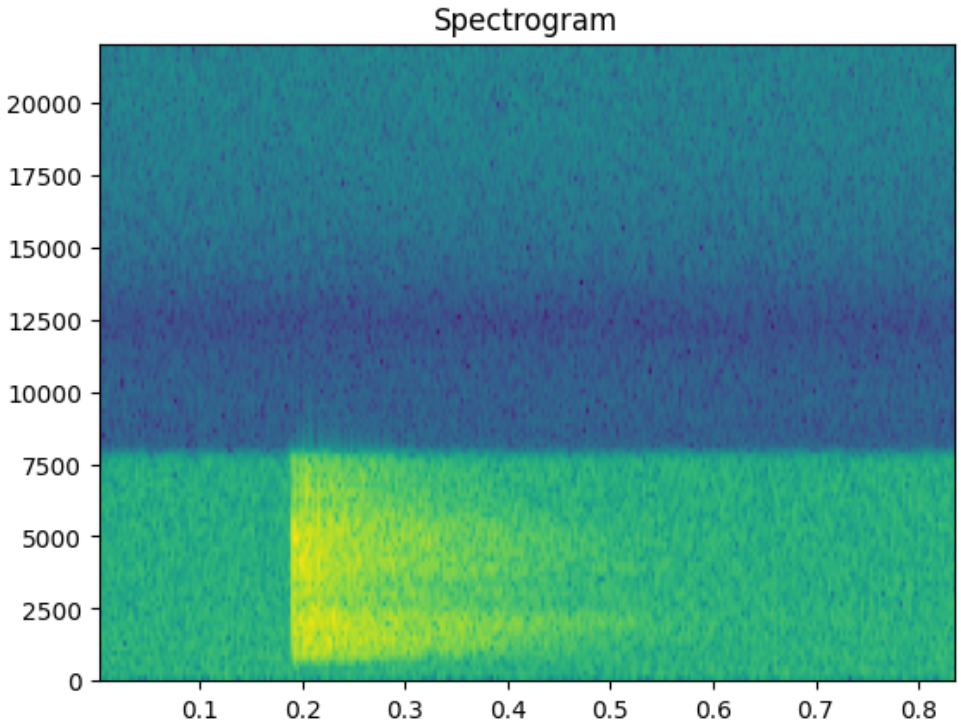

FIGURE 7 – Spectrogramme représentatif des "claps" en général.

On peut y voir qu'une fréquence d'environ 8000Hz n'y est pas dépassée. De fait, elle ne l'est pour aucun des "claps". La fréquence d'échantillonnage choisie pour le *downsampling* sera dès lors 16000Hz, du fait du Théorème de Shannon qui nous impose une fréquence d'échantillonnage égale à un minimum de deux fois la fréquence maximum du signal.

Ensuite, afin de limiter les irrégularités dans le signal déséchantillonné, il nous a été demandé d'implémenter deux filtres *anti-alliasing* : un filtre de Cauer et un filtre de Chebychev, puis de les tester sur un signal composé d'un sinus de d'amplitude 1000V et de fréquence 8500Hz et d'un sinus d'une amplitude de 20V et de fréquence 7500Hz afin de déterminer lequel des deux filtres entraînerait le minimum de délai et de déformation du signal. Après investigation, il apparaît que le filtre de Cauer est le signal conservant au mieux le signal. C'est donc celui-ci que nous avons utilisé par la suite.

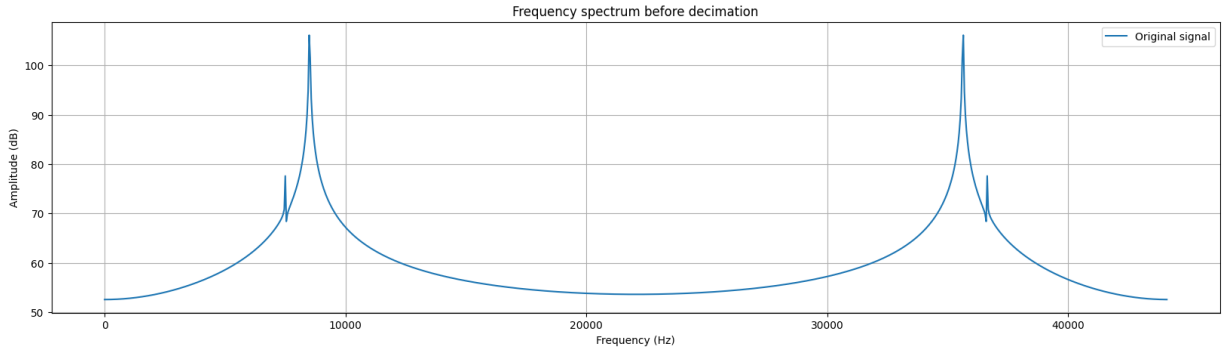

FIGURE 8 – Spectre en fréquence du signal originel.

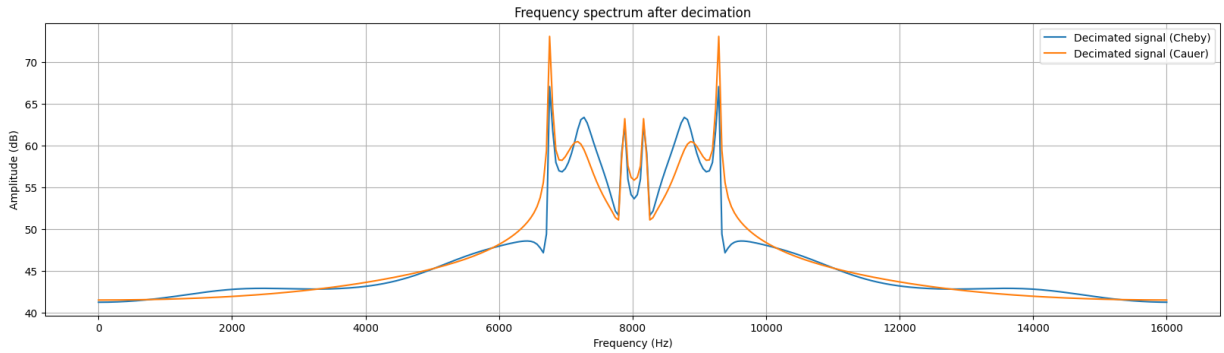

FIGURE 9 – Spectre en fréquence du signal passé dans le filtre de Chebychev (bleu) et de Cauer (orange).

Remarque : L'étape de décimation reprise ci-dessous est déjà prise en compte dans la Figure 8.

Enfin, l'étape finale du processus de déséchantillonnage consiste à appliquer une décimation sur le signal filtré en enlevant un échantillon sur M où M est le facteur de décimation demandé ; dans le cadre de ce projet, M vaut 3.

1.4 Corrélation croisée

Pour mesurer la similarité de deux signaux, nous avons utilisé la corrélation croisée, une variante de la convolution.

Dans les signaux 2D, elle calcule la surface commune entre deux signaux lorsque l'on fait glisser le premier sur le second (signal rouge sur signal bleu). La corrélation croisée ne renverse pas le second signal comme le ferait la convolution.

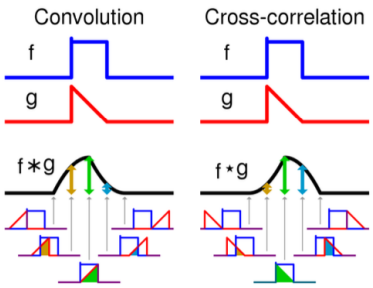

FIGURE 10 – Comparaison entre la convolution et la corrélation croisée. La différence entre les deux est que le signal g n'est pas inversé dans la corrélation croisée.

Remarque : cette image provient de l'énoncé du projet.

Nous avons implémenté notre propre corrélation croisée (*fftxcorr*) à partir de sa définition inspirée de la convolution utilisant la transformée de Fourier afin de profiter de l'avantage de celle-ci. La transformée de Fourier d'une convolution devient une simple multiplicité de la transformée de Fourier de chaque signal.

Nous avons obtenu le résultat attendu :

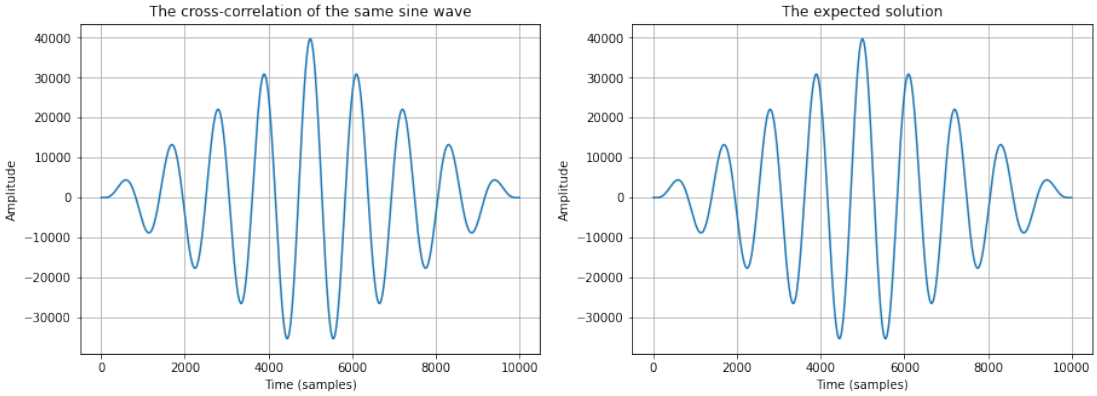

FIGURE 11 – Comparaison entre le résultat obtenu avec notre fonction *fftxcorr* (à droite) et le résultat obtenu avec la fonction fournie pour le test (à gauche).

Remarque : l'axe des abscisses représente le temps par rapport au nombre d'échantillons écoulés.

1.5 Localisation

Pour notre système de localisation, nous n'avons dû implémenter que deux fonctions : une fonction *TDOA* basée sur la corrélation croisée et une fonction *source_angle* pour obtenir l'angle pour des coordonnées données.

1.5.1 Différence de temps d'arrivée (TDOA)

Comme indiqué dans l'introduction, la TDOA est basée sur le décalage temporel entre un même signal enregistré par différents capteurs. Une façon de mesurer ce décalage temporel est d'utiliser la corrélation croisée vue précédemment et d'obtenir l'indice de l'échantillon de plus grande amplitude.

1.5.2 Système d'équation de localisation

Notre fonction *source_angle* fournit l'angle entre l'axe des abscisses et la position de la source audio. Pour cela, il suffisait simplement de calculer l'angle (en degrés) entre les deux vecteurs unitaires correspondant à l'axe des abscisses et la demi-droite partant de l'origine vers la source.

Nous n'avons effectué qu'un seul test, le calcul de l'angle d'un des micros se trouvant à 90 degrés.

1.6 Précision et rapidité des systèmes

Pour tester la précision du système, il nous a été demandé d'implémenter une fonction *accuracy* afin de comparer la différence entre l'angle prédit (obtenu par nos fonctions) et l'angle réel avec un seuil pour détecter les prédictions erronées. Pour cela, nous avons utilisé une expression booléenne élégante inspirée d'un nos précédents projets :

$$min(abs(pred\_angle - gt\_angle), 360 - abs(pred\_angle - gt\_angle)) < threshold$$

où *pred_angle* est l'angle prédit, *gt_angle* est l'angle réel et *threshold* est le seuil.

Pour définir la valeur du seuil, nous avons utilisé *accuracy* sur les 12 angles disponibles dans l'ensemble de données *LocateClaps* avec un seuil arbitraire et après observation, nous avons remarqué qu'avec un seuil de 11, notre précision moyenne était de 100%. Cette valeur nous paraissait grande mais satisfaisante car elle n'est pas plus élevée que la moitié de la différence entre les angles possibles (tous espacés de 30 degrés). Cela garanti l'impossibilité que la marge d'erreur nous amène à diagnostiquer un angle différent de l'angle réel.

Afin de tester le temps d'exécution, une fonction (*time_delay*) nous a été fournie, utilisant la librairie *time*. Les résultats obtenus lors de ces tests *offline* montrent de bonnes performances de notre système.

2. Localisation en temps réel

2.1 Équipement matériel

Dans cette section du rapport dédié au matériel, il nous est demandé de trouver une application pratique du Raspberry PI, de l'expliquer brièvement, ainsi que son utilisation générale.

Après recherches, il apparaît qu'une des applications intéressantes d'un Raspberry PI est l'automatisation des systèmes d'une maison, ainsi que de contrôler ceux-ci à distance via une application ou des via commandes vocales de manière similaire à l'Assistant Google.

Ainsi, il est possible de contrôler la luminosité ou même la couleur de lumières intelligentes, de contrôler son thermostat, de l'électroménager intelligent, comme une machine à laver, et même des systèmes de sécurité comme une alarme, par exemple.

Une telle automatisation est rendue possible par une application comme Home Assistant [1], une application open-source codée en Python.

2.2 Acquisition et traitement des données

Bien que notre temps d'interaction avec le Raspberry PI lui-même ait été limité, il nous est apparu que, lors des tests en temps-réel, l'application se comporte moins bien du fait de la mémoire limitée de l'application et de la sensibilité du matériel.

Conclusion

Pour conclure ce rapport portant sur le projet, supervisé par Mr.Hugo Bohy, du cours de Traitement du Signal donné par le Prof. Thierry Dutoit, nous allons revenir sur les différents points et étapes avec une vision globale.
Ce projet a été très instructif sur le traitement du signal, bien que nos interactions avec la partie temps-réel (l'accès au Raspberry PI lui-même) aient été très limitées.

Concernant les résultats obtenus lors du test avec le matériel, comme mentionné dans la section 2.2 concernant le traitement des données en temps réel, ceux-ci ont été peu concluants de par la nature de l'environnement, la sensiblité du matériel, et le temps d'accès réduit à celui-ci. Cependant, au vu de la précision obtenue lors des tests *offline* tels que décrit dans la section 1.6 sur la précision et la rapidité de notre application, nous pouvons avancer que notre implémentation possède un certain degré de fiabilité.

1. Pour des informations complètes et approfondies : le site internet de Home Assistant