Projet : localisation du son en temps réel

Traitement du

signal Nov-Déc

2022

Introduction

Le traitement du signal en temps réel consiste à construire des systèmes capables de réaliser une tâche spécifique dans le temps imparti. Dans de nombreux domaines, ces types de systèmes sont très utiles car ils utilisent la *rétroaction*, c'est-à-dire qu'ils s'adaptent eux-mêmes pour contrôler la sortie.

L'objectif de ce projet est de vous donner une intuition de l'importance des[1] systèmes de traitement du signal en ligne. Vous allez mettre en œuvre un système qui traite les signaux audio pour estimer la position d'un son. À la fin de ce projet, vous devriez être en mesure de déployer votre travail sur un Raspberry Pi équipé d'un réseau de microphones pour localiser la position d'un son en temps réel.

Différence de temps d'arrivée

De nombreuses techniques existent pour comprendre d'où vient un son. Nos oreilles par exemple sont capables de nous donner une idée de la position grâce en partie à la différence de temps entre leur perception de pression[2] [1]. Sur le même principe, des systèmes tels que les sonars ou les systèmes de surveillance sismique utilisent une technique appelée différence de temps d'arrivée (TDOA) pour trouver la localisation du son. Ils mesurent la différence de temps entre les signaux provenant de plusieurs microphones synchronisés et calculent la position de leur source.

Ensemble de données

Les applications en ligne sont toujours évaluées deux fois : une fois hors ligne pour mesurer la capacité de l'application à atteindre des performances acceptables (par exemple, l'exactitude, la précision) ; puis en ligne, au moment de l'exécution, pour vérifier qu'elle fonctionne en temps réel. Pour la première partie, vous aurez besoin de données créées pour les besoins de l'application que vous mettez en œuvre. Le jeu de données **LocateClaps qui** vous est fourni a été enregistré dans la salle Numediart du laboratoire ISIA et contient l'enregistrement par trois microphones de sons de claquements de mains situés à 12 positions différentes.

[1]c'est-à-dire en temps réel, par opposition au hors ligne.
[2]dans la gamme des basses fréquences, au-dessous de 10 kHz

La configuration de l'enregistrement est la suivante : trois microphones enregistrent simultanément des sons aigus. La position de chaque son est indiquée par son angle par rapport à l'axe des x, tout en maintenant une distance d'environ 3 mètres du centre du réseau de microphones. Les informations relatives à la position d'une source et du microphone utilisé sont intégrées dans le nom du fichier correspondant (par exemple : *M1 30.wav* est le fichier relatif au microphone 1 et la source est à 30°).

Procédure

Dans ce qui suit, vous allez diviser votre projet en plusieurs fonctions placées dans un seul fichier python (appelé module) et nommer ce module utils.py. Créez un second module nommé main.py dans lequel vous importerez vos fonctions et les utiliserez, construirez votre projet et ferez votre analyse.

Il vous est demandé de tester chacune des fonctions que vous construisez pour vérifier que la sortie correspond à vos prédictions lorsque vous injectez une entrée spécifique. Lors de la dernière session de laboratoire, vous pourrez pousser votre code sur un Raspberry Pi et évaluer les performances en ligne de votre système.

Votre rapport contiendra l'analyse et les interprétations requises (ce qui inclut la réponse aux questions posées explicitement ainsi que vos propres choix de mise en œuvre et les performances de votre système). Chaque résultat que vous obtenez doit être analysé et interprété, même s'il ne vous est pas explicitement demandé de le faire.

Bibliothèques Python utiles

Comme Python dispose de nombreuses bibliothèques utiles, certaines d'entre elles possèdent des fonctions que vous pourriez vouloir utiliser pour réaliser ce projet. Vous pouvez trouver les entrées et sorties ainsi que des exemples pour chaque fonction dans la documentation disponible en ligne. Voici une liste de quelques bibliothèques qui pourraient vous être utiles :

- scipy (.signal et .optimize) : est une bibliothèque bien connue pour de nombreuses opérations de traitement du signal ou de mathématiques respectivement.
- audiofile : contient la fonction **read** qui permet de lire un fichier wav en Python.
- glob : contient la fonction **glob** qui permet d'enregistrer plusieurs chemins de dossiers et/ou de fichiers dans la même liste.
- collections : gère différents types de vecteurs/listes.
- time : contient les fonctions **time** ou **time ns** qui permettent d'enregistrer l'heure.

- PyAudio : est principalement utilisé pour accéder aux fonctionnalités matérielles liées à l'audio (microphones ou haut-parleurs) par le biais de la **classe PyAudio.**[3].

1 Système hors ligne

Cette partie du projet concerne l'implémentation de chaque fonction séparément, leur mise en cascade pour construire le système et l'évaluation de ses performances hors ligne. Elle comprend la gestion des données, le prétraitement et le calcul de la position basée sur la corrélation croisée. La figure 1 résume les étapes du projet en blocs.

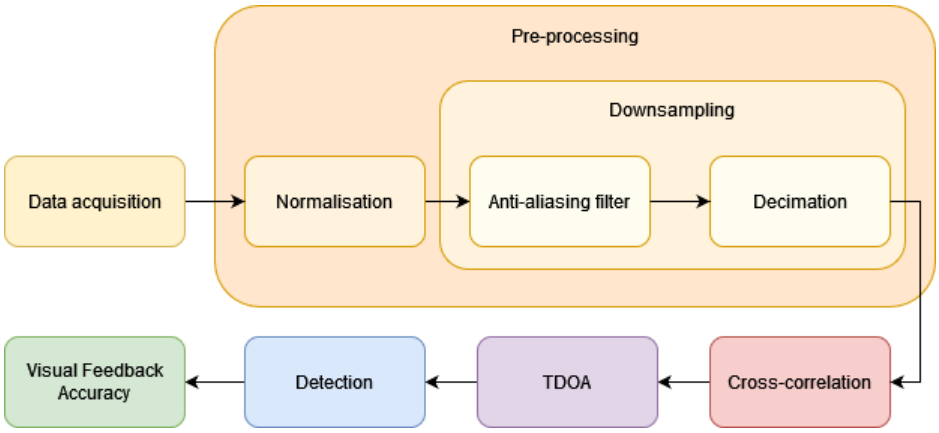


Figure 1 : Schéma de principe du projet.

1.1 Génération des données et ensemble de données

Afin d'évaluer le système, vous devez utiliser des signaux qui vous donneront des résultats prévisibles. D'une part, vous allez générer un signal connu qui sera utilisé pour démontrer l'efficacité de la plupart de vos fonctions. D'autre part, vous lirez les signaux de l'ensemble de données **LocateClaps** afin d'évaluer la précision de votre système pour localiser les sons réels.

1. Signal sinusoïdal : **Créez un signal sinusoïdal** construit avec 8000 sam- ples, avec une amplitude crête à crête de 8 et une fréquence de 20 Hz. La fréquence d'échantillonnage est de 44,1 kHz.

2. Localisez les données de Claps : **Complétez la fonction read wavefile pour qu'elle produise des données à partir d'un chemin de fichier donné.** Une option est d'utiliser la lecture de la bibliothèque scipy.signal.wavefile.

Affichez le contenu d'au moins un fichier .wav dans une figure, ainsi que votre onde sinusoïdale dans une autre.

[3]Vous n'utiliserez pas cette bibliothèque vous-même au cours de ce projet.

1.2 Mise en mémoire tampon

Pour gérer le flux continu de données (stream) enregistrées par un microphone, nous les stockons dans une mémoire tampon. Un tampon est similaire à une salle d'attente pour les données à traiter. Lorsque vous n'avez pas besoin de conserver des données déjà traitées, un tampon en anneau peut être utilisé. Un tampon en anneau est un type spécifique de tampon de taille limitée dans lequel nous mettons à jour les emplacements d'échantillons les plus anciens avec de nouveaux échantillons. Lorsqu'un certain nombre d'emplacements ont été mis à jour, nous traitons à nouveau le tampon (cf. Figure 2).

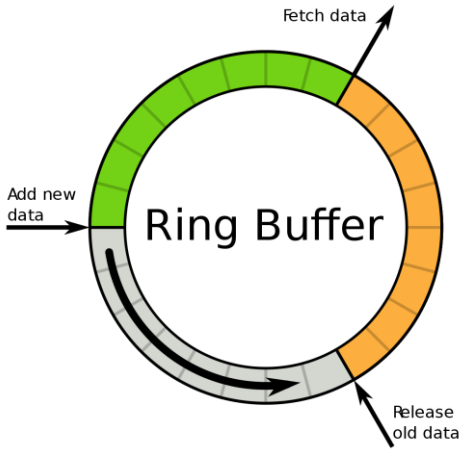


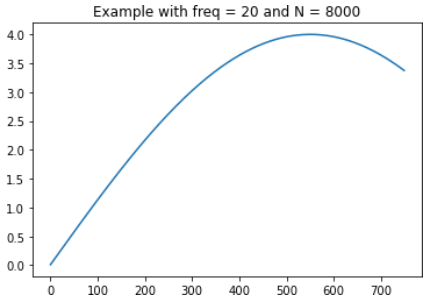
Figure 2 : Schéma d'un tampon en anneau [de RingBuffer]. La section verte représente les nouvelles données, la section orange les données à traiter et la section grise les données qui ont été traitées.

Pour créer un tampon circulaire en Python, on peut utiliser la fonction **deque**[4] qui crée un objet deque qui se comporte comme un tampon circulaire. Cela signifie qu'il enregistre les données jusqu'à ce que tous ses emplacements aient été remplis. Puis, lorsque de nouvelles données arrivent, il déplace son contenu pour supprimer l'échantillon le plus ancien à la fin de la file d'attente et enregistre le nouveau au début.

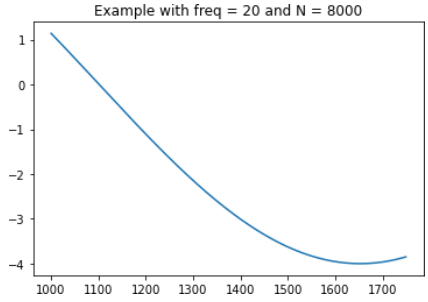
1. **Complète la création de ringbuffer pour qu'il utilise deque pour créer un tampon en anneau de longueur maximale *maxlen* échantillons**.

2. Pour vérifier l'efficacité de votre implémentation, **créez d'abord un tampon en anneau de *maxlen* 750** et **utilisez la fonction extend pour sauvegarder votre signal sinusoïdal un échantillon à la fois** (c'est-à-dire dans une boucle *for*). Ensuite, à l'aide de matplotlib, affichez le contenu du tampon lorsqu'il atteint sa taille maximale, puis lorsque le 1000ème échantillon a été sauvegardé (cf. Figure 3).

[4] à partir des collections de la bibliothèque Python



(a) Teneur en tampon de 0 à 750 samples



(b) Teneur en tampon de 1000 à 1750 échantillons

Figure 3 : Exemple de vérification pour un signal sinusoïdal de $A_{pp} = 8$, $f = 20$Hz et $N = 8000$ à $f_s = 44{,}1$ kHz.

1.3 Pré-traitement

La plupart du temps, les données provenant de capteurs peuvent nécessiter un prétraitement pour s'adapter à un format qui augmente l'efficacité de l'application. Dans cette section, il vous est demandé de modifier certains aspects des signaux comme l'amplitude ou la quantité d'échantillons.

1.3.1 Normalisation

L'amplitude du signal varie entre ses valeurs maximale et minimale qui dépendent des caractéristiques du capteur et de l'intensité du signal. Pour normaliser l'amplitude entre [-1, 1], **implémentez la fonction normalise afin qu'elle produise la forme normalisée du signal d'entrée.** Appliquez-la à votre signal sinusoïdal et montrez l'avant et l'après normalisation sur la même figure.

1.3.2 Déséchantillonnage

Les capteurs tels que les microphones ont leurs propres spécifications de fréquence d'échantillonnage qui peuvent ne pas convenir à vos besoins (par exemple, les données fournies dans **LocateClaps** sont échantillonnées à 44,1 kHz). En examinant le spectrogramme d'un signal, vous pouvez voir quelle partie du spectre de fréquences est occupée. Si les fréquences utiles sont suffisamment basses pour respecter le théorème de Shannon, on peut sous-échantillonner le signal pour traiter moins de données (donc moins de calculs) sans perte importante d'informations. Comme indiqué dans le chapitre 3 de Traitement du signal, le sous-échantillonnage s'effectue en deux étapes : le filtre anti-repliement élimine d'abord les composantes de fréquence supérieures à la moitié de la nouvelle fréquence d'échantillonnage, afin d'éviter le repliement lors de l'étape suivante de décimation. Cette deuxième étape ne conserve qu'un seul échantillon sur M (M étant le facteur de décimation) (Cf. Figure 4).

1. Analyse du signal : **Analysez** d'abord **les fréquences utiles des données de LocateClaps à l'aide de specgram.**[5] sur la base de ce que vous avez observé,

[5]de matplotlib.pyplot

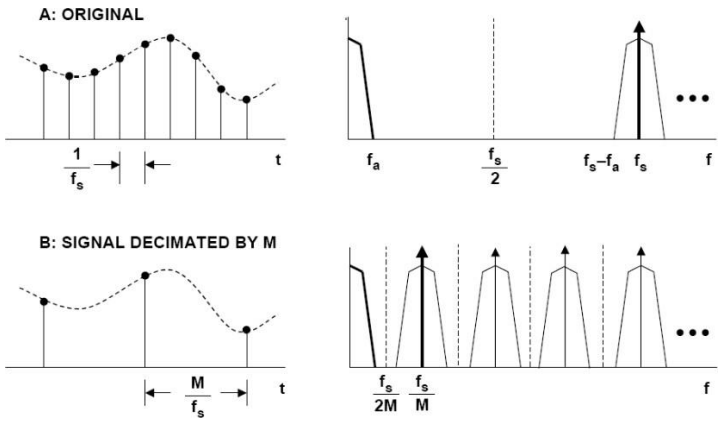


Figure 4 : Décimation par un facteur $M = 4$. [Tiré de Signal Processing, chapitre 3, Fig. 3.18, p. 13].

choisir la fréquence d'échantillonnage la plus appropriée f_s entre les suivantes : 8k, 16k, 24k ou 44.1k Hz et justifiez ;

2. Filtre anti-repliement : Examinez ensuite les filtres de Chebychev et de Cauer pour éliminer toutes les composantes de fréquence supérieures à $f_s/2$ (f_s étant la fréquence d'échantillonnage finale). Le filtre doit éliminer les fréquences supérieures à $f_s/2$ Hz. Le filtre retenu doit avoir un retard minimal et une déformation minimale sur le signal. **Testez les deux filtres sur un signal qui contient une onde sinusoïdale d'amplitude 1000 V et de fréquence 8500 Hz et une autre onde sinusoïdale d'amplitude 20 et de fréquence 7500 Hz.**

3. Décimation : Comme indiqué, la décimation est l'action de garder un échantillon parmi M. En python, vous pouvez effectuer la décimation facilement avec la syntaxe suivante :

   vecteur décimé = votre vecteur [: :M]

   Vérifiez la décimation avec un signal contenant une onde sinusoïdale d'une amplitude de 1000 V et d'une fréquence de 8500 Hz et une onde sinusoïdale d'une amplitude de 20 V et d'une fréquence de 7500 Hz avec un facteur de décimation $M = 3$. En outre, montrez dans le domaine fréquentiel la différence entre un signal décimé et un signal non décimé lorsque nous utilisons le filtre anti-repliement.

1.4 Corrélation croisée

La corrélation croisée est une méthode permettant de mesurer la similarité de deux signaux. La figure 5 montre que la corrélation croisée est une variation 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). Alors que la convolution

inverse le second signal, la corrélation croisée le conserve tel quel. Pour cette raison, il faut faire attention au fait que la convolution a la propriété de commutativité mais que la corrélation croisée dépend de l'ordre des signaux.

Les équations 1 et 2 montrent les formules de transformation de Fourier de la convolution et de la corrélation croisée respectivement. L'avantage de l'utilisation de la transformée de Fourier est que la transformée de Fourier d'une convolution devient une simple multivariation de la transformée de Fourier de chaque signal[6].

$$\text{Convolution : } \{g * h\} = F^{-1}\{F\{g\} \cdot F\{h\}\} \tag{1}$$

$$\begin{aligned}\text{Corrélation croisée : } \{g(t) \star h(t)\}(t) &= g(t) * h(-t) \\ &= F^{-1}\{F\{g\} \cdot \overline{F\{h\}}\}\end{aligned} \tag{2}$$

où F est la transformée de Fourier et $\bar{h}$ le conjugué complexe de h.

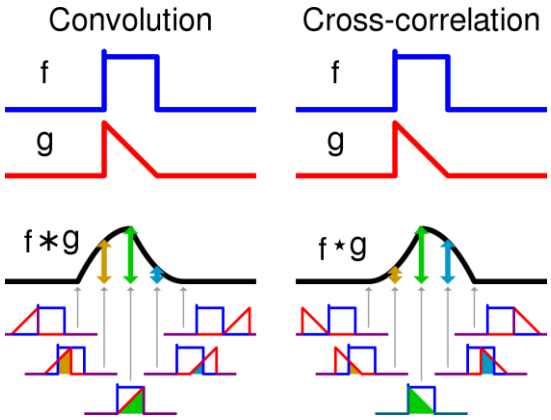


Figure 5 : Comparaison entre la convolution (à gauche) et la corrélation croisée (à droite). La différence entre les deux est que le signal **g** n'est pas inversé dans la corrélation croisée. [Modifié à partir de Wikipedia]

Sur la base de l'équation 2, implémentez la fonction fftxcorr de manière à ce qu'elle produise la corrélation des deux signaux. Utilisez la bibliothèque numpy.fft (ou np.fft) pour obtenir la transformée de Fourier de vos signaux. Vérifiez votre implémentation, comparez votre sortie avec celle de la fonction fftconvolve[7] lors du calcul de l'auto-corrélation[8] de votre signal sinusoïdal. N'oubliez pas que vous devez contrer le retournement que la convolution effectue sur le second signal.

[6] cf. Traitement des signaux, chapitre 4
[7] de la bibliothèque Python scipy.signal.
[8] c'est-à-dire la corrélation croisée d'un même signal

1.5 Localisation

Dans cette section, vous allez implémenter deux fonctions pour votre système de localisation : une fonction TDOA basée sur la corrélation croisée et un solveur pour les équations de localisation.

1.5.1 TDOA

Comme indiqué dans l'introduction, la TDOA est basée sur le décalage temporel entre le 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. **Implémentez la fonction TDOA qui prend en entrée le vecteur de corrélation de deux signaux et sort la valeur du décalage temporel.**

1.5.2 Système d'équation de localisation

Le décalage temporel nous permettrait de comprendre quel capteur reçoit le signal en premier. Nous pouvons combiner deux décalages temporels pour trianguler la source du signal dans un espace à deux dimensions. La figure 6 montre la configuration du projet : trois microphones (en bleu) sont installés aux coins d'un triangle, dont le centre est l'origine (en rouge). Un son est produit à un endroit aléatoire src (en vert). La distance entre le microphone i et le lieu du son est donnée par par la vitesse du son dans l'air ($v \approx 343$ m/s) multipliée par le temps d'arrivée (t_i). Cette distance est également l'hypoténuse d'un triangle rectangle comme représenté sur la figure. Bien que nous connaissions la position de chaque microphone, la distance entre deux (micros i et j) peut également être caractérisée par leur différence de temps pour détecter le son ($v \times (t_i - t_j) = v \times \tau_{i,j}$).

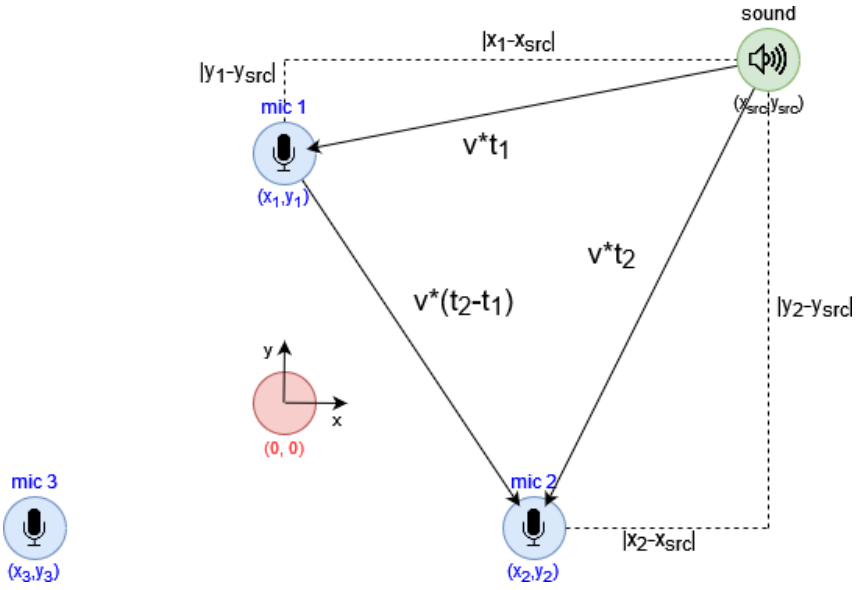


Figure 6 : Schéma de la position des microphones.

Sur la base de cette géométrie, la triangulation peut être écrite comme suit :

$$v \times \tau_{2,1} = \sqrt{(x_2 - x_{src})^2 + (y_2 - y_{src})^2} + \sqrt{(x_1 - x_{src})^2 + (y_1 - y_{src})^2}$$
$$v \times \tau_{3,1} = \sqrt{(x_3 - x_{src})^2 + (y_3 - y_{src})^2} + \sqrt{(x_1 - x_{src})^2 + (y_1 - y_{src})^2} \quad (3)$$

avec $\tau_{i,j}$ le décalage temporel TDOA entre les capteurs i et j, x_k , y_k les coordonnées x et y du capteur k et v la vitesse du son dans l'air.

Étant donné la figure 6 et les coordonnées du microphone (0, 0.0487), (0.0425, -0.025)
et (-0.0425, -0.025) pour les microphones #1, #2 et #3 respectivement, vous devriez être capable de trianguler la source d'un signal. Dans ce projet, nous nous concentrerons principalement sur la prédiction de l'angle entre l'axe des x et la position de la source, sur la base des coordonnées prédites. Nous vous fournissons la fonction localise sound qui fournit les coordonnées de la source lorsque deux TDOA sont donnés. **En utilisant ces coordonnées, implémentez la fonction angle source de façon à ce qu'elle fournisse l'angle pour des coordonnées données.**

1.6 Précision et rapidité des systèmes

Les systèmes en ligne comptent sur le faible coût de calcul de leurs opérations internes pour fournir les résultats les plus rapides et les plus précis. Lors de la mise en œuvre de votre système, il vous a été demandé de vérifier les sorties de vos fonctions. Cette section vise à mesurer la précision et la vitesse de votre système.

La façon dont nous mesurons la précision est de comparer la différence entre la valeur attendue (la **vérité terrain**, qui fait référence à une valeur que l'on sait être vraie) et la valeur prédite à un seuil.

Pour l'évaluation de la vitesse, Python n'est pas connu comme le langage le plus performant en terme de vitesse de traitement. Nous baserons notre mesure sur la bibliothèque time, qui fournit la fonction time ns . Cette fonction renvoie le moment où elle a été appelée en ns, ce qui peut ne pas être assez précis (et renvoie 0 ns) si la fonction est bien optimisée.

1. **Implémentez la fonction précision de manière à ce qu'elle compare la différence entre l'angle prédit et l'angle de vérité au sol avec un seuil pour détecter les prédictions erronées.[9].** La fonction doit renvoyer *True* si la différence entre les deux angles est inférieure au seuil ; sinon, elle doit renvoyer *False*.

2. **Utilisez la précision sur les 12 angles disponibles dans l'ensemble de données LocateClaps et rapportez les résultats (pour chaque angle et donnez également la précision moyenne).**

[9]Rappel : l'angle de vérité du sol est écrit à la fin du nom du fichier, avant l'extension : M1 30.wav signifie 30°.

3. La fonction time_delay imprime la vitesse de traitement d'une fonction donnée. Il s'agit d'une fonction wrapper, ce qui signifie que si vous lui fournissez une fonction et ses entrées, les sorties du wrapper seront les mêmes que celles de la fonction que vous avez fournie. Par exemple, lorsque nous appelons le code suivant :

```
def ma fonction ( a , b ) :
    return a*b

def wrapper ( func , args ) :
    pri nt (" Wrapper ici ") out
    = func (*args )
    Retourner

pri nt ( ma_fonction ( 10 , 2 ))
pri nt ( wrapper ( ma_fonction , [ 1 0 , 2 ] ) )
```

le résultat est le suivant :

```
20

Enveloppeur ici
20
```

Utilisez la temporisation du wrapper sur les fonctions suivantes (avec un fichier de LocateClaps comme signal d'entrée par défaut) :

- normaliser,
- le sous-échantillonnage,
- fftxcorr (et comparer avec fftconvolve),
- TDOA,
- localiser le son,
- angle de la source

Indiquez la vitesse lorsque le signal est sous-échantillonné et lorsqu'il ne l'est pas (n'évaluez pas la fonction de sous-échantillonnage dans ce cas). Expliquez la différence que vous observez et quelle situation est la plus rapide.

2 Localisation en temps réel

Pour effectuer la détection en temps réel, il faut un matériel spécifique qui embarque plusieurs microphones. En plus de cela, nous devons également gérer les données sous forme de flux continu. Cette section décrit les différents processus mis en œuvre pour obtenir une application en temps réel.

2.1 Equipement matériel

Le matériel sur lequel votre implémentation sera téléchargée consiste en un Rapsberry Pi 3 modèle B et un Respeaker 6-Mic Circular Array (cf. Figure 7 et Figure 8) :

Raspberry Pi 3B Le Raspberry est un ordinateur monocarte de la taille d'une carte de crédit. En plus des ports que l'on trouve sur la plupart des ordinateurs (HDMI, USB, jack 3,5 mm), il comprend également 40 broches GPIO qui permettent de contrôler des modules matériels (tels que des LED). La Raspberry exécute un système d'exploitation Linux spécifique appelé Raspberry Pi OS et contient un IDE Python.

Respeaker Mic Array Le réseau circulaire de 6 microphones conçu par Seeed Stu- dio pour étendre le Raspberry Pi. Il contient 6 microphones conçus pour des applications au- dio ainsi que 12 LEDs de forme circulaire uniformément séparées. Ces caractéristiques peuvent être contrôlées avec Python 3 puisqu'elles sont reconnues comme une entrée audio standard. Ses spécifications d'enregistrement sont les suivantes :

- max fs = 48kHz,
- Sensibilité omnidirectionnelle,
- SNR = 59dB

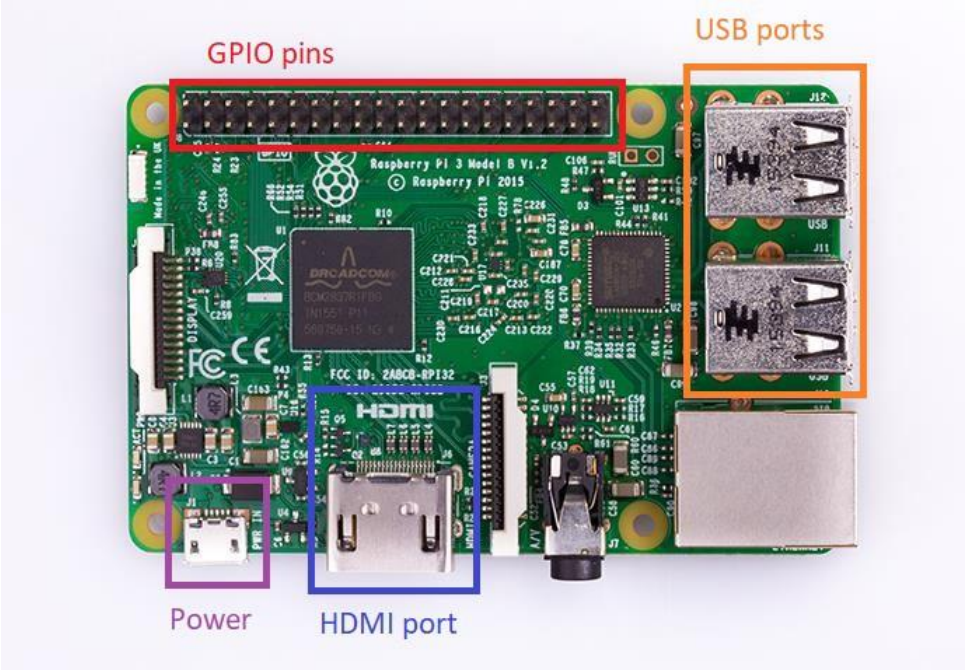


Figure 7 : Raspberry PI 3B, avec les zones mises en évidence [modifié de Mouser].

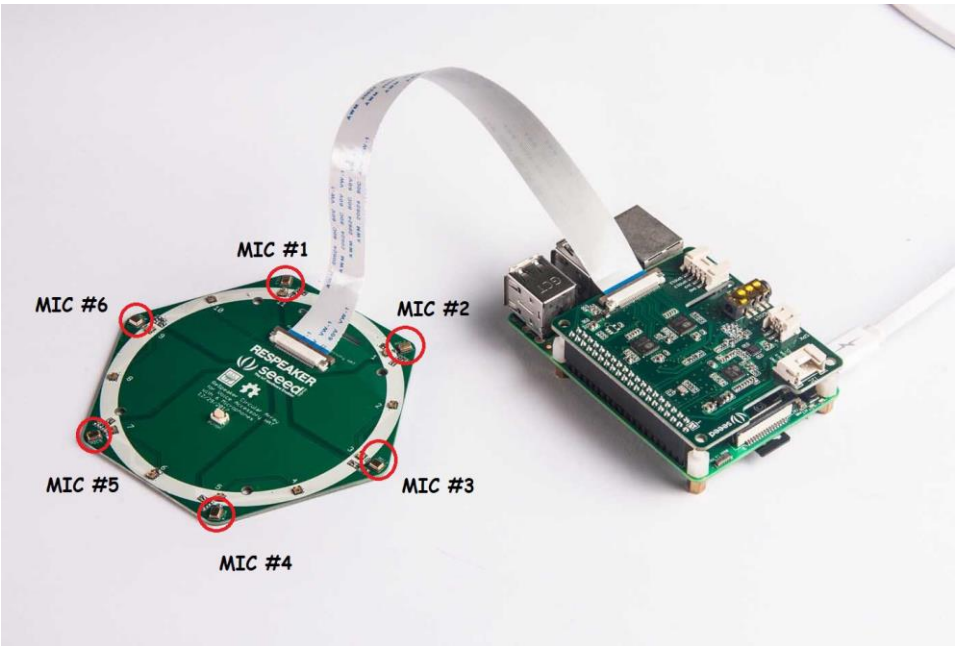


Figure 8 : Kit Respeaker 6-Mic Circular Array [de Seeed Studio].

Dans cette section, il vous est demandé de rechercher une applica- tion pratique avec Raspberry Pi. Expliquez brièvement dans votre rapport le fonctionnement de l'applica- tion choisie et son utilisation générale.

2.2 Acquisition et traitement des données

Callback Le matériel utilisé pour le projet envoie dans un flux la concaténation des données de plusieurs microphones, chacun étant échantillonné à 44,1 kHz. Dans ce projet, trois microphones seulement sont nécessaires alors que six sont disponibles sur la carte audio (cf. Figure 8), ce qui signifie que nous ne devons sauvegarder que la moitié du flux. Comme nous avons l'intention de conserver la même forme triangulaire formée par les microphones (cf. Figure 6), nous sélectionnerons les données des microphones #1, #3 et #5.

Pour sauvegarder les données sans arrêter le processus, nous avons implémenté une fonction de rappel. Une fonction de rappel peut être vue comme une fonction appelée chaque fois que le matériel est capable d'enregistrer un nouvel ensemble d'échantillons. La fonction de rappel que nous avons écrite sauvegarde les données dans des tampons séparés, chacun lié à un microphone.

Gestion des flux Pour collecter les données de la carte audio, nous devons créer un objet stream lié à ce matériel. Les fonctions init stream et close_stream gèrent respectivement la création et la fermeture d'un stream pour vous. Traiter les signaux en temps réel signifie utiliser les données du flux pendant qu'il est actif. Dans ce projet, la fonction detection appelle

les fonctions que vous avez implémentées ci-dessus pour détecter un angle en temps réel.

Retour visuel Afin de vérifier votre système, nous avons utilisé des fonctions qui contrôlent les lumières LED de la carte pour fournir un retour visuel sur l'angle prédit. Comme il y a 12 LEDs, nous avons adapté l'intensité de chaque LED pour fournir plus d'informations sur la proximité de l'angle prédit par rapport à la position d'une LED : si une couleur est brillante sur une LED particulière, la source est dans la même direction ; si deux ou trois LEDs adjacentes sont allumées, la source est dans une direction entre celles-ci.

Pour cette section, il **vous est demandé de faire une évaluation subjective de deux aspects de la mise en œuvre en ligne** :

1. la précision de la direction fournie par le retour visuel ;
2. l'impression de temps réel du système (mentionnez comment le délai entre la création du son et sa détection par le système)

Comparez les résultats avec différentes tailles de tampon (1s, 2s et 3s) ou fréquences d'échantillonnage (16 kHz ou 24 kHz).

Références

[1] C. K¨oppl, "Evolution of sound localisation in land vertebrates", *Current biology*, vol. 19, no. 15, pp. R635-R639, 2009.