Projet de Programmation : Cahier des besoins Spatialisation du son par le geste

---

Etudiants :
Johan David
Christophe Gaydier
Jean Lebrun
Louis Sicardon
Jon Stark

Chargé de TD :
M.Phillipe Narbel
Clients :
M.Myriam
Desainte-Catherine
M.Gaël Jaton
M.Jean-Michel Rivet

13 février 2020

Table des matières

1 Présentation du projet

Dans le cadre de l'unité d'enseignement *Projet de Programmation (PdP)* au sein de la première année de Master Informatique à l'Université de Bordeaux, le sujet qui nous a été attribué est le suivant : *Spatialisation par le geste*. Ce sujet a été proposé par des collaborateurs du *SCRIME*[1] : Mme Myriam DESAINTE-CATHERINE (directrice du SCRIME, chercheuse en Informatique et professeur des Universités) et M. Gaël JATON (régisseur technique et développeur électronique embarquée).

Dans le but de travailler en collaboration avec le compositeur M. Jean-Michel RIVET (compositeur de musique électroacoustique et collaborateur au *SCRIME*), l'objectif de ce projet est de répondre à un besoin proposé par ce dernier à savoir le contrôle de haut-parleurs disposés au sein d'un acousmonium[2] dans une salle de concert. L'idée proposée par Jean-Michel RIVET est de pouvoir contrôler des haut-parleurs à l'aide d'un capteur, situé dans sa main, afin de pouvoir y régler le volume durant une représentation lors d'un concert à la manière d'un chef d'orchestre. Le compositeur souhaiterait pouvoir désigner une enceinte en "pointant" son capteur vers le centre de celle-ci pour ensuite effectuer un geste vers le haut ou vers le bas afin d'augmenter ou diminuer le volume de la piste audio diffusée sur cette enceinte. Cela aura pour objectif de recréer une spatialisation de l'audio par les gestes dans l'espace et de substituer le travail réalisé à la main par le compositeur sur la table de mixage.

Nous allons devoir développer une interface utilisateur simple d'utilisation permettant une mise en service facilitée avant chaque concert et réaliser une interaction entre le capteur et l'enceinte en passant par l'interface mise à notre disposition et développée par le *SCRIME* pour le réglage des enceintes.

1. Studio de Création et de Recherche en Informatique et Musiques Expérimentales de l'Université de Bordeaux.

2. "Orchestre" de haut-parleurs destiné à l'interprétation en concert des musiques composées dans un studio électro-acoustique.

2 Analyse de l'existant

Deux projets ont été proposé en collaboration avec le SCRIME par le passé :

2.1 *Percussion aérienne* - 2004

En 2004, un premier projet *(cf. [2])* a été mis au point. Cela ne concerne pas totalement notre domaine car d'autres fonctionnalités ont été développé comme l'amélioration de la détection des coups dans le jeu d'un percussionniste. En effet, le projet avait pour but de commander des sons de batterie en fonction de gestes sur différentes enceintes situées dans la pièce. Ce projet datant de 2004 utilise d'anciennes technologies et sont peu adaptables avec les frameworks et les API développés par le SCRIME. La technologie utilisée n'intègre pas non plus de communication en Bluetooth, ni de PlayStation Move pour les mouvements. En effet, un capteur est présent sur l'extrémité d'une baguette de batterie permettant de contrôler les sons diffusés en fonction de l'orientation dans la pièce.

2.2 *Spatialisation de la musique par le geste* - 2011

En 2011, un second projet *(cf. [1])*, un PFA (Projet de Fin d'Année), a été développé dont le but est de reconnaître les mouvements du musicien pour les convertir en commandes. Une interface graphique permettait de représenter la scène physique avec l'utilisateur, l'émetteur et les enceintes afin de faire le calibrage.

2.3 *Cross-Platform Tracking of a 6DoF Motion Controller* - 2011

Une API pour gérer le PlayStation Move (PSMove) a été développé lors de cette thèse *(cf. [3])*. Cette librairie est *open-source* et *cross-plateform*. Elle permet de connaître les mouvements de plusieurs PSMove via Bluetooth. La manette contient un accéléromètre et différents capteurs permettant de localiser l'utilisateur et où il se situe. On peut ainsi contrôler les trois axes de position et de rotation. La librairie gère le problème de communication via le protocole Bluetooth HID.

2.4 *Extracting Commands From Gestures* - 2013

En 2013, cette thèse de fin d'année *(cf. [4])* explique comment reconnaître des gestes comme des commandes. Ce qui permet de donner à l'utilisateur un contrôle du flux sonore durant son écoute. Il y est expliqué l'utilisation de l'algorithme de `Dynamic Time Warping (DTW)` pour analyser et classifier des gestes. L'auteur cherche à répondre à deux problèmes. Le premier est de reconnaître des mouvements/commandes dans un flux de données. Le second est de chercher comment on divise ce flux en sous parties afin de trouver des échantillons pertinents pour nos données d'entraînement.

L'algorithme de `DTW` compare les données récupérées sous forme de signal par le capteur avec les différents modèles de gestes que l'on veut reconnaître. Pour cela, on calcule le taux d'erreur entre les modèles et le mouvement. On garde ceux qui ont un taux d'erreur inférieur à un seuil que nous aurons définis. On applique ensuite l'algorithme des plus proches voisins `Knn` pour choisir le signal du modèle qui correspond le mieux à notre geste.

2.5 *Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping* - 2012

La particularité de cet article *(cf. [?])* est qu'il a été écrit dans des conditions de recherche utilisant un échantillon de données plus important que tous les articles précédents. Il y est expliqué quatre optimisations de l'algorithme `DTW` permettant de résoudre des problèmes complexes de *Machine Learning* tel que la reconnaissance de symbole ou le partitionnement de données.

- *Early Abandoning Z-Normalization*
- *Reordering Early Abandoning*
- *Reversing the Query*
- *Cascading Lower Bounds*

L'intérêt de cet article est que leurs techniques nous permettent de gérer rapidement un flux de données important avec une puissance matérielle plus faible.

3 Besoins

3.1 Besoins fonctionnels

L'objectif principal de la solution est de pouvoir faire varier le son des enceintes à l'aide de mouvements de la main. Pour ce faire, on doit pouvoir sélectionner une ou plusieurs enceintes et y ajuster le son. Nous devons donc détecter le mouvement effectué par le "chef d'orchestre", en récupérant des échantillons de la position de la main et en déduire une enceinte et une valeur pour le volume. Ces valeurs sont ensuite envoyées au logiciel qui gère les enceintes afin d'ajuster le volume de celle sélectionnée. Il doit également être possible de visualiser quelle enceinte est sélectionnée.

Nous avons pensé à deux possibilités :

- placer des LED qui s'allument sur l'enceinte utilisée (solution visuellement peu esthétique) ;
- utiliser une interface graphique qui affiche l'état des enceintes (solution privilégiée par le compositeur).

Si un geste n'est pas reconnu, le programme ne doit pas planter et il doit être possible de reprendre la main sur la table de mixage à tout moment.

3.1.1 Diagramme d'activité

FIGURE 1 – Diagramme d'activité

3.1.2 Diagramme de séquence

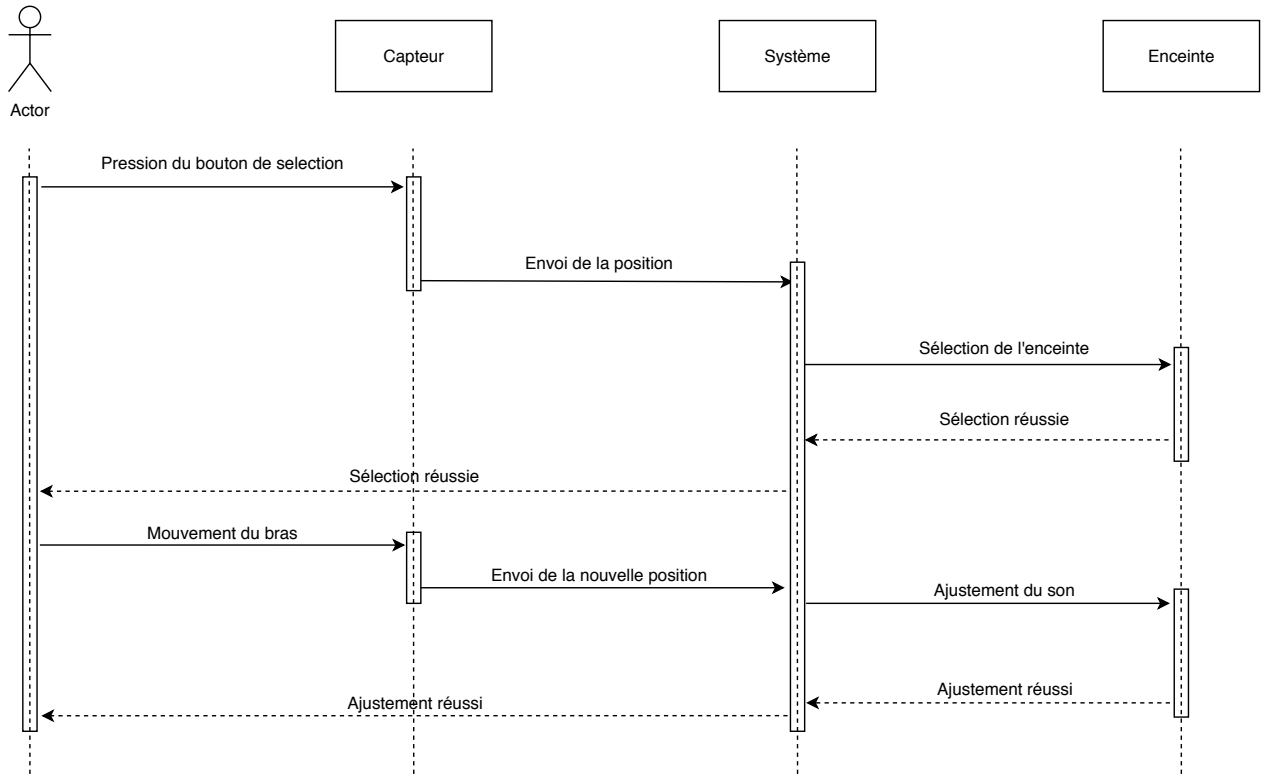


FIGURE 2 – Diagramme de séquence

3.1.3 Réalisation

Transfert et traitement des données

Les données du capteur seront exporté sous le format OSC (Open Sound Control) vers la plate-forme OSSIA. Ces deux éléments permette la récupération et le traitement de données en temps réel.

Calibrage

Tout d'abord, les positions des enceintes peuvent être différentes en fonction des salles dans lesquelles sont effectuées les représentations. Une première phase de **calibrage** est alors nécessaire afin d'interagir correctement avec les enceintes. Nous avons imaginé que l'utilisateur pourrait pointer les enceintes une par une et appuyer sur un bouton à chaque fois pour définir leur position initiale.

Sélection d'une enceinte

Nous utilisons une manette *PlayStation Move* contenant des capteurs qui nous permettent d'obtenir un **cap** (axe horizontal) et une **élévation** (axe vertical). Cette manette possède plusieurs boutons dont une gâchette. Nous avons donc imaginé que la sélection de l'enceinte se ferait à partir de ce bouton. Il faudra alors pointer avec la main l'enceinte que l'on souhaite sélectionner et appuyer sur la gâchette pour pouvoir modifier le son qu'elle émet.

Sélection d'enceinte d'un groupe d'enceinte

Ce système doit également permettre à l'utilisateur de sélectionner plusieurs enceintes. Un geste déclencheur ou un autre bouton pourrait permettre à l'utilisateur de garder active toutes les enceintes qu'il sélectionne et également de toutes les dé-sélectionner.

Reconnaissance de geste simple :

Lorsque l'utilisateur appuie sur la gâchette, on récupère le cap et l'élévation

courante du *PSMove* afin de pouvoir en déduire une enceinte ainsi qu'un point de départ. Tant que l'utilisateur maintient la gâchette, l'enceinte reste sélectionnée et on récupère les mouvements du *PSMove*. Il suffit ensuite de comparer la position courante de la manette avec celle de départ pour savoir si l'on doit augmenter ou diminuer le son et à quelle intensité.
Cette possibilité nous permet de modifier le son de manière continue à la vitesse du mouvement de l'utilisateur. On ne regardera alors que l'axe vertical du mouvement à savoir si la main monte ou descend. Nous pourrions également ajouter un effet selon l'axe horizontal (de la *reverb* par exemple).

Reconnaissance de geste complexe :

On remarque que les possibilités de mouvements sont limitées. Une autre solution serait de créer une banque données représentant des mouvements types (horizontaux, verticaux, circulaire, etc...). Il faudrait ensuite détecter le mouvement de l'utilisateur, le comparer aux données types, et trouver le mouvement type qui s'en rapproche le plus. Enfin il nous faut appliquer sur le son l'effet correspondant au mouvement.
Cette solution permettrait créer différents mouvements ou suites de mouvements qui offrirait une grande liberté au niveau des effets sonores. En revanche, on peut s'attendre à un temps de réponse supérieur dû à l'analyse du mouvement. De plus, il faut prendre en compte une marge d'erreur lors des mouvements et veiller à ne pas les confondre.

Spatialisation

L'utilisateur doit également pouvoir déplacer le son des enceintes afin de donner une impression de mouvement.

Interface Graphique

Une interface graphique pourrait être développée pour que l'utilisateur puisse visualiser l'endroit ou il pointe, les enceintes sélectionnées et le volume de chaque enceinte.

3.1.4 Priorités

Besoins de priorité majeurs
Ces besoins sont les plus importants et correspondent à la première phase du développement. C'est une solution minimale fonctionnelle.

- Transfert de données
- Calibrage
- Sélection d'une enceinte
- Modification du son (première solution)

Besoins de priorité intermédiaires
Ces besoins correspondent à la deuxième phase du développement.

- Modification du son (deuxième solution)
- Sélection d'un groupe d'enceinte

Besoins de priorité mineurs
Cette priorité correspond aux besoins qui seront implémentés durant la deuxième phase si le projet avance correctement.

- Spatialisation
- Interface graphique

3.2 Besoins non fonctionnels

Les besoins non fonctionnels peuvent se scinder en différentes catégories : les besoins non fonctionnels comportementaux, les besoins non fonctionnels externes et les besoins non fonctionnels organisationnels. Nous allons détailler ici des besoins relevant de ces différentes catégories.

3.2.1 Performance

1. Le taux d'échantillonnage [3] doit pouvoir être assez précis pour représenter le mouvement du chef d'orchestre. Il ne doit pas être trop élevé afin de ne pas avoir à gérer un trop grand nombre de données brutes provenant du capteur qui seraient difficiles à stocker du point de vue de la mémoire provoquant également une augmentation du "bruit" (parasitage des données) pouvant fausser la reconnaissance des gestes dans la banque de données. Mais également pas trop faible pour bien reconnaître un geste. Il faut donc trouver un juste milieu. Cela se fera uniquement à l'aide de tests.

2. La réduction de la latence (durée entre l'exécution du geste et la réponse du point de vue de l'audio) doit être aussi la plus faible possible. En effet, le temps de réaction de l'oreille humaine est environ de `20 ms`, ainsi lorsque le chef d'orchestre exécute un mouvement, il faut que la réponse se fasse après un délai raisonnable (maximum `1 s` après un geste).

3. Nous devons également connaître le nombre d'enceintes au préalable pour pouvoir les gérer de manière indépendante et/ou de manière groupée. Le nombre d'enceintes peut être variable ainsi que leurs positions dans la salle de concert. Une phase de calibrage doit être effectuée au préalable par l'utilisateur avant toute utilisation avec le capteur pour permettre d'identifier une enceinte et sa position. Il suffira ensuite de pointer avec le capteur vers l'enceinte émettant du son.

4. Il faut en outre définir une distance minimale à respecter entre les enceintes pour ne pas avoir d'interférences et ainsi sélectionner la bonne enceinte.

3. le nombre de données capturées par seconde

5. Nous devons réfléchir également au type de langage de programmation que nous allons utiliser. Devons-nous privilégier un langage orienté objet ou un langage purement fonctionnel ? (C++, Java, C, Python, ML, ...)

3.2.2 Compatibilité

1. Nous devons connaître le type de machine sur laquelle le logiciel va s'exécuter. Après consultation avec nos clients, nous avons décidé d'utiliser un Raspberry Pi 4 fonctionnant avec le système d'exploitation Linux. Il a la particularité de pouvoir s'exécuter sur n'importe quel ordinateur avec n'importe quel système d'exploitation et cela de manière indépendante et autonome. Ce matériel nous sera fourni par le SCRIME.

2. Notre système doit pouvoir s'intégrer avec les différents frameworks et plateformes utilisés par le SCRIME et ainsi s'adapter au système de gestion audio existant (*OSSIA-Score* - développé par le SCRIME).

3. Nous devons pouvoir utiliser tous types de capteurs inertiels mis à notre disposition. Nous utiliserons principalement comme capteur la manette *PlayStation Move (SONY)* comportant un accéléromètre, un gyroscope, une gâchette, divers boutons, et une communication en Bluetooth.

4. Nous devons être en mesure d'utiliser comme type de protocole les protocoles utilisés lors d'échanges de données musicales : Open Sound Control conçu pour le contrôle en temps réel.

3.2.3 Facilité d'utilisation

Le système doit être facile d'utilisation et facile d'installation lors de sa mise en route. Le chef d'orchestre doit pouvoir en quelques minutes mettre en route les liaisons entre les enceintes et définir leurs placements dans la pièce. La manière dont il commande les enceintes doit être également simple d'utilisation. Notre projet s'adresse à un client qui n'est pas dans le domaine de l'informatique, il doit être relativement simple d'utilisation et de mise en place pour lui.

3.2.4 Robustesse et Fiabilité

La fiabilité et la robustesse du système doit être un élément clé du projet. En effet, durant un concert, il faut pouvoir reprendre la main sur la table de mixage et pouvoir gérer le volume des enceintes manuellement. Nous allons devoir effectuer un bon nombre de tests dans différentes conditions réelles pour tenter de minimiser les erreurs et voir si des interférences peuvent se produire :

- présence de plusieurs capteurs Bluetooth présents dans la salle pouvant provoquer des interférences ;
- distance trop proche entre les différentes enceintes ;
- problème de liaison ;
- autres problèmes techniques.

Nous devons être capable de gérer tous ces cas là et de répondre le plus rapidement possible.

3.2.5 Organisation

L'organisation et la gestion du temps sont également à prendre en compte dans la liste des besoins non fonctionnels. En effet, nous devons respecter un certain planning et respecter le calendrier des livrables. Nous devons rendre une première version du code développé, rendre le mémoire (rapport final) et une version finale du code en fin de semestre.

Notre client aimerait également faire une première démonstration en public lors de la sortie de son nouvel album en mai.

4 Test

Afin de garantir le bon fonctionnement du projet après sa livraison nous allons effectuer une série de test sur les spécifications fonctionnelles et non fonctionnelles.

4.1 Test des spécifications fonctionnelles :

Transfert de données : Afin de nous assurer que la fonction de transfert des données s'exécute comme il faut, nous allons implémenter des tests unitaires, en comparant une valeur attendue et une valeur reçue.

Calibrage : Afin de tester la fonction de calibrage, nous allons essayer de calibrer plusieurs fois la même enceinte.

Reconnaissance de geste simple : Afin de tester la reconnaissance d'un geste simple (à savoir monter et descendre le bras) nous allons répéter plusieurs fois chaque geste en introduisant de plus en plus de bruit.
Nous couvrirons également la fonction de reconnaissance de geste simple de tests unitaires.

Sélection d'enceinte et de groupe d'enceinte : Nous effectuerons le geste de sélection d'enceinte dans un espace vide de toute enceinte pour vérifier qu'aucune enceinte est sélectionnée. Nous effectuerons aussi ce geste en sélectionnant une enceinte ou plusieurs enceintes.

Reconnaissance de geste complexe : Pour tester la reconnaissance de différents gestes, d'après cette thèse de fin d'année [4] nous devons définir des mouvements simples et caractéristiques. Comme par exemple, un mouvement horizontal, vertical, en diagonal ou bien de forme circulaire (arc de cercle ou cercle complet). On doit répéter plusieurs fois le même mouvement afin de créer un modèle pour ce geste. On teste plusieurs fois chaque modèle pour vérifier son efficacité. Cela signifie que l'on va comparer si le geste effectué et celui de notre modèle sont similaires.

Pour évaluer notre modèle, on peut créer plusieurs suites de mouvements

consécutives d'une durée de plus ou moins longue. Par exemple 10 à 20 secondes contenant 1 à 3 modèles différents.

On peut aussi définir des gestes à l'aide de données numériques et les tester sur les modèles. Il est aussi possible d'ajouter du bruit sur les données pour simuler un comportement humain.

4.2 Test des spécifications non-fonctionnelles :

Performance :

1. Taux d'échantillonnage : pour pouvoir définir un taux d'échantillonage nous allons faire un stress tests de l'algorithme de reconnaissance de geste complexe en définissant un taux d'échantillonage élevé (10 000 points par secondes). En fonction du délai d'exécution et de la précision de l'algorithme nous ferons varier le taux d'échantillonage afin d'améliorer la performance de notre programme.
2. Réduction de la latence : afin de tester ce critère nous mesurerons la durée entre l'exécution du geste et la réponse du point de vue de l'audio.

Compatibilité : Les tests relatifs à la compatibilité de notre application seront validés si notre application est compatible avec le matériel et les différents logiciels que nous utiliserons.

Facilité d'utilisation : Afin de valider le besoin de facilité d'utilisation nous effectuerons avec Jean-Michel Rivet une série de test où celui-ci utilisera l'application. S'il réussit à utiliser l'application sans difficultés avec le guide utilisateur alors le test sera validé.

Robustesse et Fiabilité : Pour tester la robustesse et la fiabilité du projet nous allons effectuer une série de test (introduire plusieurs capteurs Bluetooth dans la salle), empiler deux enceintes, générer des problèmes de liaison et d'autres problèmes techniques.

5 Diagramme de Gantt

Voici une première version du Diagramme de Gantt présentant les différentes échéances de mise en oeuvre des besoins tout au cours du semestre :

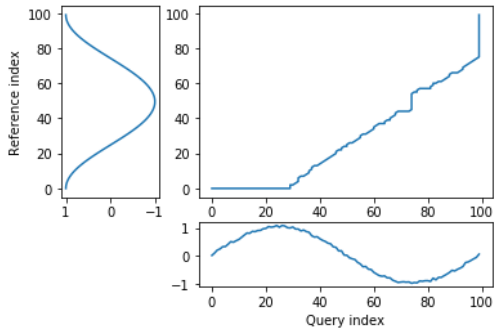
Tâche à accomplir	Semaine												
Suivi du projet	3	4	5	6	7	8	9	10	11	12	13	14	15
Rendez-vous Client													
Suivi du projet avec le chargé de TD													
Audit													
Soutenance													
Rédaction													
Rédaction du cahier d'analyse des besoins													
Livraison du cahier d'analyse des besoins													
Remaniement du cahier des besoins													
Rédaction de la documentation technique													
Rédaction du manuel utilisateur													
Rédaction du mémoire													
Livraison du mémoire, du manuel utilisateur et la documentation													
Développement													
Prise en main des API et des alogrithmes													
Première phase de développement													
Livraison de la première du code													
Intégration des retours clients													
Deuxième phase de développement													
Livraison finale du code													
Test													
Vérification des spécifications de la première phase													
Validation des spécifications de la première phase													
Vérification des spécifications de la deuxième phase													
Validation des spécifications de la deuxième phase													

FIGURE 3 – Diagramme de Gantt

6 Étude de faisabilité

6.1 Algorithme de Dynamic Time Warping

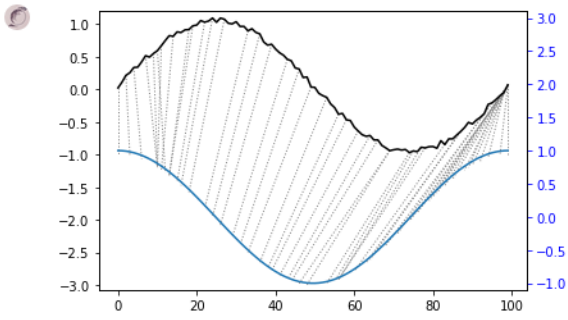
Nous avons testé un algorithme de Dynamic Time Warping [4] en ligne. Nous avons pu obtenir les résultats suivants :




```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc787403da0>
```

FIGURE 4 – Génération de signaux avec du bruit par rapport à un index de référence

```
[ ]  ## Align and plot with the Rabiner-Juang type VI-c unsmoothed recursion
     dtw(query, template, keep_internals=True,
         step_pattern=rabinerJuangStepPattern(6, "c"))\
         .plot(type="twoway",offset=-2)
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc78d3e8668>
```

FIGURE 5 – Comparaison d'un signal bruité par rapport à un signal de référence

4. https ://dynamictimewarping.github.io/python/

6.2 Connexion du PSMove et utilisation des capteurs

Afin de savoir s'il était possible d'établir une communication entre le PSMove et le Rasberry Pi4 nous avons installé la PSMoveAPI [5] sur ce dernier.

```
pi@raspberrypi:~/psmoveapi/build $ sudo ./psmove pair
Connected controllers: 1
[PSMOVE WARNING] Magnetometer in 00:06:f5:1e:47:f3 not yet calibrated.
PSMove #1 connected via USB.
[PAIRING LINUX] Found host adapter: dc:a6:32:44:4f:4a (name=hci0)
[PAIRING LINUX] File /var/lib/bluetooth/DC:A6:32:44:4F:4A/00:06:F5:1E:47:F3/info needs updating.
[PAIRING LINUX] Running: 'systemctl stop bluetooth.service'
[PAIRING LINUX] Writing file: /var/lib/bluetooth/DC:A6:32:44:4F:4A/00:06:F5:1E:47:F3/info
[PAIRING LINUX] File /var/lib/bluetooth/DC:A6:32:44:4F:4A/cache/00:06:F5:1E:47:F3 is already up to date.
[PAIRING LINUX] Running: 'systemctl start bluetooth.service'
Pairing of #1 succeeded!
Controller address: 00:06:f5:1e:47:f3
Calibration data available and saved.
```

FIGURE 6 – Connexion du PSMove au Rasberry Pi4

```
pi@raspberrypi:~/psmoveapi/build $ ./psmove dump-calibration
[PSMOVE WARNING] Magnetometer in 00:06:f5:1e:47:f3 not yet calibrated.
File: /home/pi/.psmoveapi/00_06_f5_1e_47_f3.calibration
System file: /etc/psmoveapi/00_06_f5_1e_47_f3.calibration
Flags: 1
Have USB calibration:
10 00 b1 07 d9 7f 64 7f 51 90 ec 6e 21 7f 3d 7f
12 80 59 7f 43 6e 08 91 a1 7f 65 7f f1 7f 59 90
92 7f 1e 80 63 6e 46 7f 3e 09 22 80 fb 7f 81 80
10 08 26 80 f8 7f 89 80 00 00 00 00 00 00 00 01
00 01 38 09 e0 01 d6 94 3d 80 bb 80 e0 01 d6 7f
0c 92 7a 80 e0 01 df 7f 0a 80 96 94 32 08 90 01
12 43 f0 0e c2 42 15 9b 6b 43 7c ff 70 3f 3a 3c
7c 3f 9f 95 72 3f 76 54 3b 3f 5a df ae 3d 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

# Temperature: 0x07B1 (27 °C)
# Orientation #0:      (  -39 |  -156 |  4177)
# Orientation #1:      (-4372 |  -223 |  -195)
# Orientation #2:      (   18 |  -167 | -4541)
# Orientation #3:      ( 4360 |   -95 |  -155)
# Orientation #4:      (  -15 |  4185 |  -110)
# Orientation #5:      (   30 | -4509 |  -186)

# Temperature: 0x0938 (36 °C)
# Gyro X, 80 rpm:      ( 5334 |    61 |   187)
# Gyro Y, 80 rpm:      (  -42 |  4620 |   122)
# Gyro Z, 80 rpm:      (  -33 |    10 |  5270)

# Temperature: 0x093E (36 °C)
# Gyro, 0 rpm (@0x2a): (   34 |    -5 |   129)

# Temperature: 0x0810 (29 °C)
# Gyro, 0 rpm (@0x32): (   38 |    -8 |   137)

# Temperature: 0x0832 (30 °C)
# Vector @0x5e: (146.006104 | 97.029175 | 235.605789)
# Vector @0x6a: (0.941398 | 0.985294 | 0.947596)

# byte @0x3f:  0x01
# float @0x76: 0.731758
# float @0x7a: 0.085387
```

FIGURE 7 – Calibration des capteurs du PSMove

5. https ://github.com/thp/psmoveapi

Références

[1] Brossard Emilien Catalano Jean-Baptiste Claudel Vincent Ramchurun Avinash Vigouroux Thibault Adam Thomas, Brassellet Bertrand. *Spacialisation de la musique par le geste*. PhD thesis, PFA Enseirb-MATMECA 2010 – 2011, 15 avril 2011.

[2] Vincent Goudard. *Percussion aérienne : amélioration de la détecion des coups dans le jeu d'un percusionniste*. PhD thesis, Université de la Méditerranée, Aix-Marseille II, 9 Septembre 2004.

[3] Thomas Perl. PhD thesis, PFA Enseirb-MATMECA 2010 – 2011, 15 avril 2011.

[4] Jiuqiang Tang. *Extracting Commands From Gestures : Gesture Spotting and Recognition for Real-Time Music Perfomance*. PhD thesis, Université Carnegie Mellon, Mai 2013.