

Rapport du Projet PSAR :  
Dispositif Autonome de Synthèse Sonore

Encadrant : Hugues Genevois

Pierre Mahé

4 mai 2015

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Contexte Global . . . . .	2
1.2	Objectifs du PSAR . . . . .	2
1.3	La Carte Udoo . . . . .	3
1.4	Pure Data . . . . .	3
1.4.1	Présentation générale . . . . .	3
1.4.2	Externals . . . . .	5
1.5	Structure du projet . . . . .	5
<b>2</b>	<b>Récupération de l'environnement</b>	<b>6</b>
2.1	Dispositif et capteurs . . . . .	7
2.2	Communication inter plate-forme . . . . .	8
2.3	External pour la communication . . . . .	8
<b>3</b>	<b>Traitement audio</b>	<b>9</b>
3.1	Acquisition audio . . . . .	9
3.2	Traitement du son bas niveau . . . . .	9
3.2.1	Filtrage . . . . .	9
3.2.2	Détecteur de notes . . . . .	11
3.2.3	Bandes de fréquences . . . . .	11
3.3	Traitement du son haut niveau - Extraction des métadonnées . . . . .	11
3.3.1	Mélodie et rythme . . . . .	11
3.3.2	Détection de Motif minimal . . . . .	13
<b>4</b>	<b>Synthèse musical</b>	<b>14</b>
4.1	Synthèse du dispositif à long terme - Modèle physique . . . . .	14
4.2	Synthèse implémenté . . . . .	14
<b>5</b>	<b>Interface Utilisateur</b>	<b>14</b>
5.1	Première idée d'implémentation . . . . .	14
5.2	Envoie des données des capteurs . . . . .	16
5.3	Envoie d'objet Pure data . . . . .	16
<b>6</b>	<b>Tutoriel et Documentation</b>	<b>17</b>
6.1	Écriture documentation Pure data . . . . .	17
6.2	Écriture du Tutoriel d'installation . . . . .	18
<b>7</b>	<b>Pour aller plus loin</b>	<b>18</b>
7.1	Test en environnement réel . . . . .	18
7.2	Test énergétique . . . . .	18
7.3	Serveur distant . . . . .	19
<b>8</b>	<b>Bibliographie</b>	<b>20</b>

# 1 Introduction

## 1.1 Contexte Global

Le projet DASS, Dispositif Autonome de Synthèse Sonore, est une collaboration de Michel RISSE, compositeur de "Décor Sonore" ayant à son actif de nombreuses créations et dispositifs sonores, et de Hugues GENEVOIS, chercheur au LAM, l'Institut Jean le Rond d'Alembert.

Le projet consiste à mettre en place un dispositif générant du son en prenant en compte son environnement dans le but de souligner les sons près-existants dans cet environnement. Le dispositif recueille les informations à l'aide de capteurs diverses permettant de récupérer les bruits ambients, la luminosité, la température...

Dans un première temps, le dispositif devrait être la base pour une installation artistique, se déroulant en juillet prochain, dans le cadre d'un festival de musique de Noirlac. Dans un second temps, Hugues GENEVOIS et Michel RISSE voudraient réaliser une installation plus importante avec de nombreux dispositifs, avec pour chacun d'eux un comportement et des caractéristiques propre. Dans le but de voir l'interaction qu'ils pourraient avoir ensemble et les comportements qui pourraient en émergeraient.

## 1.2 Objectifs du PSAR

Dans le cadre de mon projet PSAR, mon but était de créer un premier prototype du dispositif pour s'en servir comme base par la suite.

J'avais donc pour mission de créer un dispositif portable interagissant avec son environnement. Ce dispositif doit capter l'environnement à l'aide de capteurs : micro, luminosité, température, et humidité. Il devra traiter ces données pour en extraire des informations pertinentes sur les changements se produisant autour de lui (chants d'oiseaux, passage d'une personne au alentour...). Grâce à ces informations, il devra synthétiser du son.

La synthèse devra être paramétrable, pour permettre au compositeur de modifier l'influence des différents capteurs sur la synthèse sonore.

De plus une interface utilisateur devra être présente permettant au compositeur de simuler des données captées. Cela lui permettra de faire des expérimentations de sa composition.

Dans l'optique de la reproduction de ce dispositif pour des installations plus important. Une procédure devra permettre à un utilisateur de pouvoir reproduire l'ensemble du système, aussi bien niveau matériel que logiciel.

Le dispositif devant être portable, le programme doit fonctionner sur une nano-ordinateur de type Udo Quad. De plus le langage du programme principal doit être le Pure Data, pour assurer une maintenabilité et une extensionnalité plus facile.



FIGURE 1 – Carte Udooh

### 1.3 La Carte Udooh

La carte Udooh est un nano-ordinateur embarquant un processeur ARM et la connectique d'un ordinateur classique, avec une entrée et sortie micro, des sorties USB, HDMI, carte WIFI... L'entrée et la sortie audio étant d'une qualité suffisante pour notre projet, il ne sera pas nécessaire ajouter à la Carte Udooh une carte son externe.

La caractéristique de cette carte est d'être compatible Arduino et d'offrir les mêmes connectiques (GPIO, SPI ...) qu'une Arduino Mega ce qui permet de simplifier la réception des capteurs de plus il permet d'utiliser les bibliothèques déjà implémentées par la vaste communauté Arduino.

Pour stocker les données, la carte utilise une carte micro SD, ce qui est également un avantage pour le projet car cela permet de dupliquer la partie software du dispositif avec une simple copie de carte mémoire.

Le système d'exploitation installé est une version légère d'Ubuntu, permettant d'utiliser l'environnement Linux.

### 1.4 Pure Data

#### 1.4.1 Présentation générale

Pure Data (Pd) est un logiciel, open source, de programmation graphique pour la création et l'interaction musicale et multimédia en temps réel. Il permet également de gérer des flux entrants dans l'ordinateur (données de périphériques ou événements réseau par exemple) et de gérer des flux sortants (par des protocoles réseaux ou protocoles pour le pilotage de matériels divers).

Pure Data est un système modulaire où il est possible de définir des sous-modules qu'il est possible de ré-utiliser à sa guise dans d'autre programme appelé *patch*.

En Pure Data tous les objets (modules ou données) sont des *boîtes*. Ce langage est un langage très faiblement typé. Il existe quatre types de données :

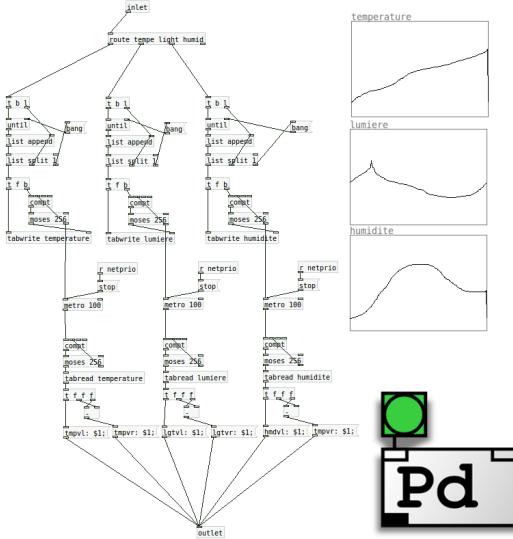


FIGURE 2 – Patch Pure Data

- **Les Nombres**, qui peuvent représenter des nombres entiers comme des nombres à virgules.
- **Les Message**, qui sont des messages strictement textuels.
- **Les Symboles**, qui correspondent aux messages mixtes, caractères, chiffres...
- **Les Tableaux**, qui sont uniquement des tableaux de nombres.

Il existe d'autres types de données comme les listes qui même si leurs représentation interne différant des messages ou des symboles sont graphiquement identique à un message (avec comme premier mot `list`). De plus pour changer une liste en message, il suffit de supprimer le mot clé `list`.

Une boîte Pure Data à une ou plusieurs entrées et une ou plusieurs sorties, toutes les entrées et sorties peuvent recevoir et envoyer n'importe quel message. On peut connecter deux modules en reliant la sortie de l'une avec l'entrée de l'autre.

Le Pure Data étant un logiciel orienté interaction, il est possible à tout moment de modifier son patch en ajoutant des boîtes au cours de l'exécution du programme, il n'est pas nécessaire de relancer le programme. Cela permet de modifier le patch en direct sans avoir l'interruption dans le flux audio sortant.

Pure Data a aussi l'avantage d'être facilement documentable car il est possible d'associer à une boîte, une boîte d'aide avec une explication et des exemples minimaux.

Cela s'avère très utile pour la compréhension de certaines boîtes complexes et permet de bien dissocier code et documentation.

Un autre atout de Pd est le nombre de d'outils déjà implémentés dans le logiciel et les boîtes créées par la communauté. Certaines procédures sont destinées à des traitements complexes, comme la détection en temps réel de fréquence ou le filtrage performant, lecture de fichiers musicaux...

#### 1.4.2 Externals

En Pure Data, le stockage de données est assez complexes c'est pour cela que pour procéder à des algorithmes complexes, il est possible d'avoir recours à des **Externals**. Les Externals sont des boîtes programmées en C++.

Il existe plusieurs API pour programmer des Externals. Dans ce projet nous avons choisi d'utiliser **Flext** pour sa simplicité d'utilisation. De plus cette API est développée par Thomas GRILL, un des principaux contributeurs de Pure Data ce qui garantit une parfaite compatibilité avec Pd et une certaine assurance que l'outil reste longtemps maintenu. La communauté des développeurs Pure Data étant limitée (une dizaine de personnes au total), cela a une importance car malheureusement beaucoup d'outils ne sont pas suffisamment maintenus ou ne sont plus complètement compatibles avec Pure Data.

### 1.5 Structure du projet

Nous avons décidé de diviser le programme en cinq modules :

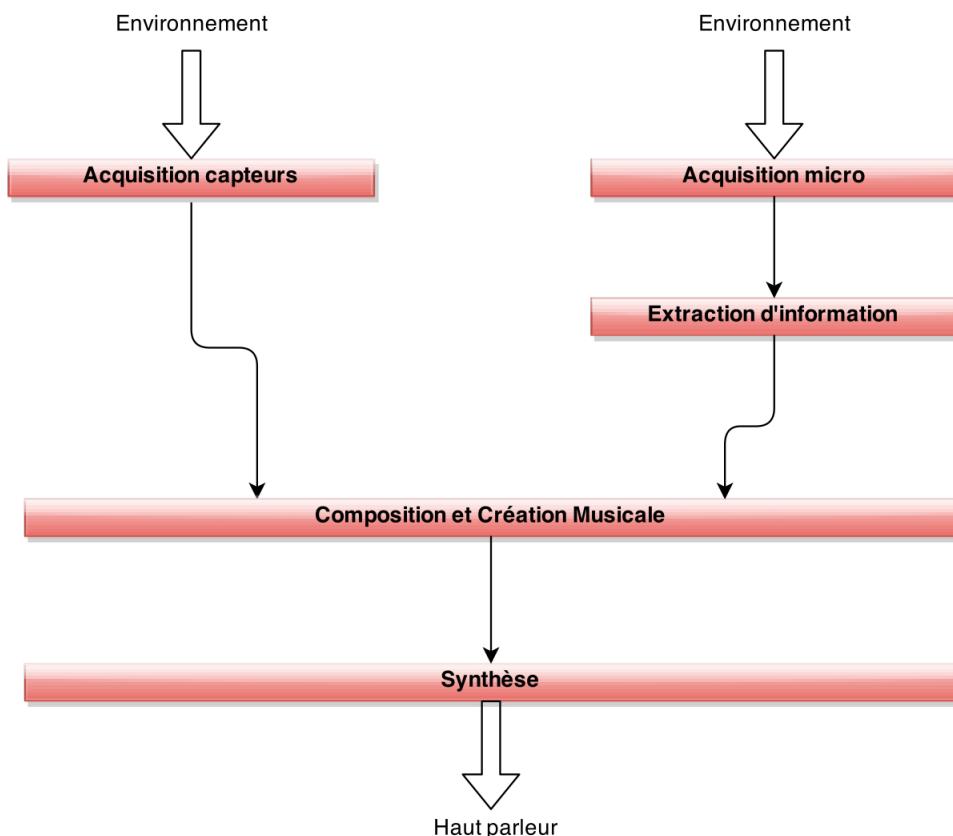


FIGURE 3 – Structure du projet

La partie acquisition des données des capteurs se fera par le biais de la communication entre la partie Arduino et Pure Data.

Nous avons écrit un programme Arduino qui va envoyer périodiquement les valeurs des différents capteurs au programme Pd.

Informations collectés par la partie Arduino :

- Luminosité
- Température
- Humidité

La seconde partie de l'acquisition est la réception du signal sonore, que l'on doit traiter pour pouvoir en extraire les informations pertinentes.

Le son capté par le micro est pollué par des sons ambients, typiquement le bruit d'une route proche ou d'une machine bruyante. Ces bruits sont généralement continus, grave (moins de 100 hertz) donc très énergétique. Ils peuvent donc fausser les résultats et alourdir les traitements en aval.

Les bruits naturels de cette fréquence sont très rares. De plus musicalement ces bruits sont assez pauvres.

Pour toute ces raisons il est préférable de les atténuer en utilisant des filtres.

Pour les mêmes raisons, nous avons filtré le signal pour éliminer les fréquences au dessus de 6000 Hertz.

Dans la partie extraction, avec le compositeur et le chercheur nous nous sommes questionné sur les informations pertinentes à extraire pour la composition musicale. En prenant en compte le fait que la carte a une capacité de calcul limité, cela ne permet pas de faire des traitements coûteux en temps de calcul.

La composition musicale est la partie propre au compositeur, et correspond réellement à la partie création artistique. Plus que la fabrication de ce module, la question a été : comment permettre au compositeur de la créer et de pouvoir modifier le module dans le dispositif à distance, sans avoir besoin de brancher un écran ou un clavier.

Il y plusieurs techniques de synthèses musicales. Certaines sont plus complexes et gourmands en ressources.

Nous avons choisi une synthèse simple. La synthèse étant très lié à la partie création musicale. Le module de création musicale étant rudimentaire, il en est donc disproportionné de faire un module de synthèse très complexe.

En plus de cette structure installé dans le dispositif, une interface graphique a été créée, elle communique à distante avec le dispositif.

L'utilisateur peut envoyer des données pour remplacer les informations reçues des capteurs, il peut de également changer le module de création musicale.

## 2 Récupération de l'environnement

L'un des avantages de la carte Udoو est d'être compatible Arduino est donc de proposer les mêmes connectiques qu'une Arduino Mega. Cela permet d'utiliser le même langage,

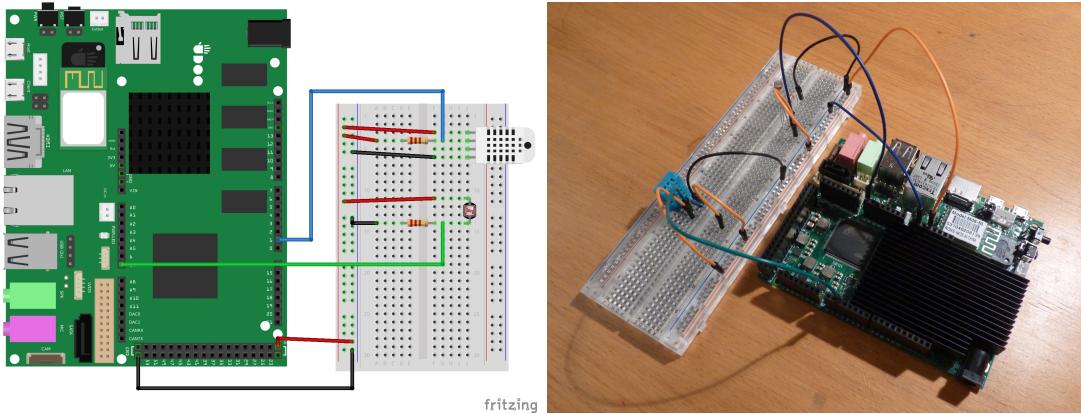


FIGURE 4 – Montage du dispositif

d'utiliser les bibliothèques déjà écrites ainsi que les mêmes capteurs. De plus la partie micro contrôleur étant connecté directement à la partie ordinateur via un bus spécial. La vitesse de communication est plus rapide et la latence est plus courte par rapport à une connexion USB classique.

Dans les informations que peuvent envoyer un capteur deux choses sont intéressantes, la valeur absolu et la variation de celle-ci (si elle augmente ou si elle diminue).

## 2.1 Dispositif et capteurs

Dans le cadre du projet, seul trois capteurs étaient prévu mais le programme permet un ajout de capteurs aisément.

Les capteurs utilisés sont des capteurs basique : un capteur de luminosité, de température et d'humidité.

Pour capteur la luminosité, nous avons utilisé une photo-résistance, son fonctionnement est assez simple, plus la luminosité va être forte plus la résistance interne de composant est faible.

Il suffit donc de récupérer la tension au borne de composant pour connaître l'indice de luminosité.

Malheureusement cette indice ne permet pas de connaître l'intensité lumineuse en Candela ou en Lux mais permet d'avoir une échelle relative de valeurs. Cela est suffisant dans le cadre du projet.

Pour la température et l'humidité la récupération la valeur est très similaire. Par contre, contrairement à la luminosité grâce à une formule (dépendant de chaque capteur) il est possible d'avoir la température ou l'hygrométrie précise (degrés Celsius pour la température et en pourcentage pour l'humidité).

## 2.2 Communication inter plate-forme

Pour la partie micro-contrôleur l'envoi de données est assez intuitive car il suffit d'écrire sur le port Série (**Serial port**), grâce à des fonctions implémentées de base en Arduino. Pour la partie ordinateur avec le logiciel Pure Data, il faut utiliser une boîte pour se connecter au périphérique puis il est possible de lire et écrire sur celui.

La connexion entre l'Arduino et Pure Data devant être unique, il a fallu trouver un moyen de pouvoir différencier les données des différents capteurs.

Pour communiquer entre les deux parties, un micro-protocole a donc été défini pour étiqueter les données envoyées.

À intervalle régulier (toutes les secondes) le programme Arduino lit les valeurs des capteurs et les envoie avec des messages de la forme :

```
1 NOM_CAPTEURvl: VALEUR;      //vl pour la valeur
2 MON_CAPTEURvr: VALEUR;      //vr pour la variation
```

Pour ajouter un capteur, il suffit d'écrire un nom de capteur -pas encore utiliser- et de modifier le programme Pure Data pour traiter la nouvelle étiquette.

Dans un second temps, il s'est avéré utile de pouvoir changer, au cours de l'exécution, le délai entre chaque envoi de valeurs.

Pour cela il a fallu permettre à l'Arduino de pouvoir recevoir des données. Les messages ayant la même forme que l'envoi de données avec l'étiquette `delay`.

Ce moment-là, nous avons découvert un défaut de Pure Data. Étant un langage très faiblement typé. Les données reçus de l'Arduino sont des octets que Pure Data caste en type Nombre et il n'existe pas d'objet permettant à partir de ce flux d'octets de récupérer des `integers` (valeur des capteurs envoyés par la partie Arduino).

Au première abord nous avons voulu utiliser l'objet `PackOSC` qui permet de convertir un message quelconque en commande OSC (**Open Sound Control**). Ce protocole étant un protocole utilisant le codage ascii, il serait possible de l'utiliser pour encoder et décoder les communications avec l'Arduino.

Mais cela aurait été une manipulation de l'objet de base de plus cela pourrait porter à confusion, le dispositif ne gérant en rien le protocole.

Nous avons discuté avec Hugues GENEVOIS et la programmation d'un externals apparaît comme plus adapté.

## 2.3 External pour la communication

En s'inspirant du fonctionnement des objets `PackOSC` et `UnPackOSC`, nous avons programmé deux externals : Un qui reçoit un message et le transforme en tableau d'octets (en code ascii plus exactement) en ajoutant le code ascii d'un point virgule à la fin pour délimiter le message.

Le second reçoit un flux d'octets et le stocke dans un tableau jusqu'à recevoir un point virgule, à ce moment-là il envoie toute la chaîne reçue.

Le fait d'utiliser le caractère ';' n'est pas un problème car en Pure Data ce caractère est

généralement le délimiteur interne pour indiquer la fin de chaîne, de liste et de tableau, d'une lecture fichier ainsi que la fin des messages TCP.

## 3 Traitement audio

### 3.1 Acquisition audio

Comme nous l'avons dit dans l'introduction du projet, Nous avons filtré le signal audio capté par le micro pour supprimer les bases fréquences et les hauts fréquences.

Nous avons chercher à savoir déterminer les fréquences de coupure à partir des quelles il n'y a plus d'informations pertinentes. Nous nous sommes rendu compte qu'au dessus de 6000 Hertz il n'y avait plus de son avec une fondamentale à cette haute, le signal capté ne sont plus que des harmoniques de sons plus graves.

Pour les sons très graves, il n'existe pas de sons produits par la nature à des fréquences inférieurs à 100 Hertz. Nous avons donc ajouté un filtre à notre module d'acquisition pour pouvoir supprimer les fréquence qui sont des bruits parasites.

### 3.2 Traitement du son bas niveau

#### 3.2.1 Filtrage

Une information qui paraissait essentiel à extraire était les notes et les mélodies qui sont joué dans l'environnement.

Il est assez facile de déterminer la fréquence principal d'un signal, il suffit de prendre la forme générale du signal. Il existe plusieurs boites en Pd pour le faire tel que **Fiddle~** ou **Signum~**.

Contrairement au suivis simple, la détection de notes jouées simultanément ou le suivi de plusieurs de mélodies simultanément est très difficile et fait encore partie l'objet de recherche.

Il n'est pas possible de déterminer toutes les notes jouées dans le monde fréquentiel, il est nécessaire de passe dans le monde spectral en utilisant la Transformé de Fourrier. Il existe une boite en Pure Data **FFT**. La boite manquant de possibilité de paramétrage (nombre de points...).

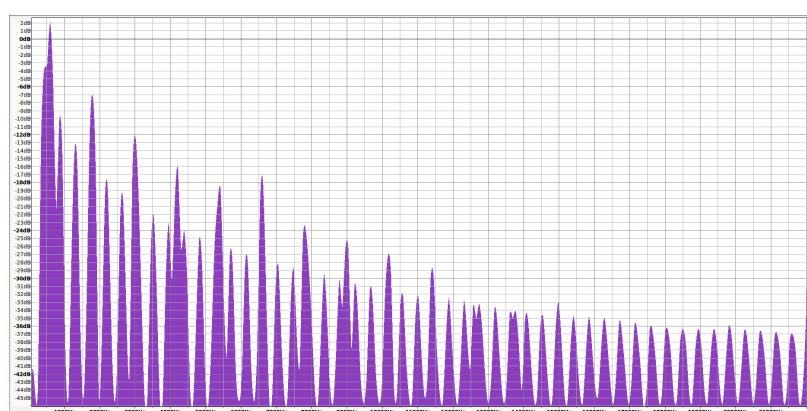


FIGURE 5 – Transformée de Fourrier

Nous avons implémenté un programme en C pour faire une transformée Fourrier Rapide (FFTW) dans l'optique d'en faire une boîte pour l'utiliser dans Pure Data. Ce qui permettrait de pouvoir mieux paramétrer la transformation.

Après des tests, notre programme c'est avéré être aussi trop gourmand pour la carte. Malgré le fait que la complexité de la FFT est de  $N^*\log(N)$ . Mais le N correspond au taux d'échantillonnage du signal (le nombre de points du signal). Notre signal est échantillonné à 44,1 kHz il y a donc 44100 points à traiter par seconde ce qui est lourd pour la carte (si nous voulons pouvoir faire d'autres traitements en parallèle).

Après ces tests insatisfaisante, nous nous sommes orientés dans une autre direction, Hugues GENEVOIS nous a conseillé de partager le spectre sonore en plusieurs bandes de fréquences à l'aide de filtres passe-bande et de suivre de déterminer la fréquence principale (la plus forte) de chaque bande, avec un **fiddle~**.

Après avoir implémenté ce mécanisme en Pure Data, un nouveau problème est apparu. Les filtres ne coupant pas parfaitement les fréquences en dehors de leurs bandes, les objets **fiddle~** détectent parfois des fréquences principales n'appartenant pas à leurs bandes de fréquence. Il y avait donc plusieurs filtres qui détectaient la même fréquence ce qui était assez problématique.

Ce phénomène arrive fréquemment quand le son enregistré contenait peu de fréquence (voir une seule). Les filtres qui n'étaient pas centrés sur la fréquence captée par le micro avaient peu de signal dans leur bande et par conséquent avaient tendance à récupérer les résidus, et donc la fréquence en dehors de sa bande de fréquence.

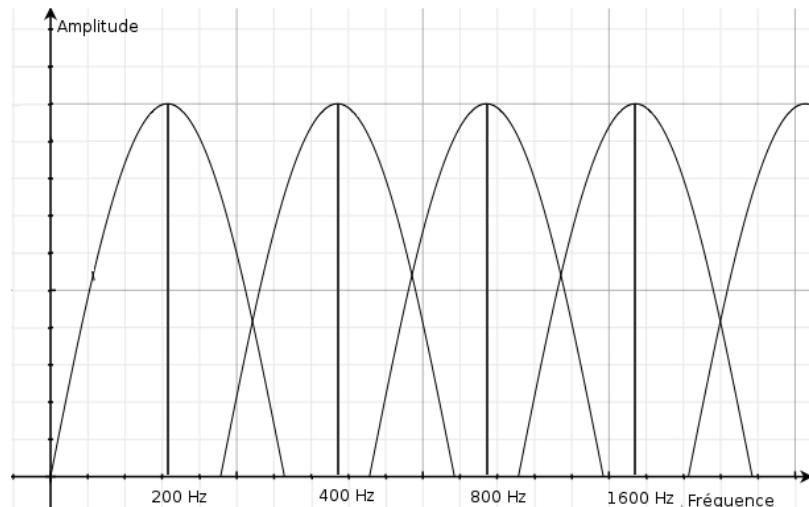


FIGURE 6 – Bandes de Fréquences

Pour résoudre ce second problème, nous avons utilisé des **Noise Gates**.

Une **Noise Gate** est un élément très utile en musique, il correspond à une boîte ne laissant passer que le signal sonore au-dessus d'une certaine intensité. Si le signal est trop faible, le signal en sortie de la boîte est égal à zéro. Cela permet quand il n'y a pas de fréquence centrée sur la bande de fréquence de supprimer les sons venant des autres bandes de fréquences. Étant atténué par les filtres, ces sons ne passent pas la **Noise Gate**, ce

qui limite le problème de la détection de fréquence erroné.

L'Élément clé pour limiter les erreurs est donc le seuil à partir duquel la Noise Gate laisse passer le signal.

Si il est trop faible, il restera beaucoup de fausses détections. Si le seuil est trop élevé, on va avoir tendance à ne pas détecter des fréquences pourtant très proche de la bande passante.

### 3.2.2 DéTECTEUR DE NOTES

Ce dispositif produit beaucoup moins d'erreurs que les précédents mais il n'était pas satisfaisant pour pouvoir faire un suivi de mélodie s'étalant sur plusieurs bandes de fréquences. Cela se traduise par la détection de morceaux de mélodie haché.

Nous avons donc décidé d'abandonner l'idée de suivre plusieurs mélodies simultanément et de ne garder que la mélodie principale du signal.

Et de n'utiliser les notes récupérer à la sortie des filtres uniquement pour connaître les notes présentes.

Cette information peut sembler peu pertinente mais il est très important pour le dispositif de savoir quelles notes sont jouées, si la mélodie qu'il capte utilise uniquement deux notes, il serait très dissonant que le dispositif joue une troisième note qui n'a rien à voir avec les deux premières.

Cela donnerait un rôle trop importante est donc contradictoire avec le projet initial qui était de faire un dispositif m'étant en valeur l'environnement.

Dans ce but, un compteur de notes a été implémenté. Il collecte les notes reconnues à la sortie des filtres et à intervalle régulier le compteur va nous donner la fréquence d'apparitions des notes.

Pour ne pas faire de dissonance il suffira au module de synthèse de piocher des notes tirés au hasard dans les notes les plus fréquemment jouées.

Il est également possible d'imaginer une synthèse qui regarderait les notes jouées pour jouer des notes pour compléter les accords.

### 3.2.3 Bandes de fréquences

Avec ce système de détection de note, il est possible de savoir la fréquence d'apparition des notes mais il n'est pas possible de savoir la hauteur des sons captés.

Nous avons donc ajouter un module pour mesurer l'amplitude du signal sur chaque bande de fréquence pour savoir comment sont repartis les notes sur les bandes de fréquences, si les bruits captés par le micro sont plutôt aiguë ou grave.

## 3.3 Traitement du son haut niveau - Extraction des méta-données

### 3.3.1 Mélodie et rythme

Comme expliqué dans la section précédente, nous avons implémenté un module pour détecter la mélodie principal du signal.

Il était naturel d'implémenter un module homologue à la détection de note, mais là pour déterminer les rythmes.

Pour synthétiser des sons avec des rythmes proches des sons environnent.

Concrètement si une personne marche près du dispositif, il pourrait répondre en jouer des sons avec le même rythme ou dans le cas où le dispositif capte un chant d'oiseau, il peut répondre en jouant des notes avec le même rythme.

Après en avoir discuté avec notre encadrant, il nous est apparu que la meilleure méthode pour détecter un début de note est d'essayer de détecter les impacts dans un signal audio. Généralement caractéristique d'un début de note. Pour cela nous avons calculé l'énergie du signal . Puis de regarder son évolution, si une augmentation brusque apparaît (un pente supérieur à un certain seuil), cela induit la présence d'un impact et donc du début de note.

Pour raffiner la détection d'impact, Le module a un seuil minimal et maximal à dépasser pour considérer que l'événement est bien un début de note.

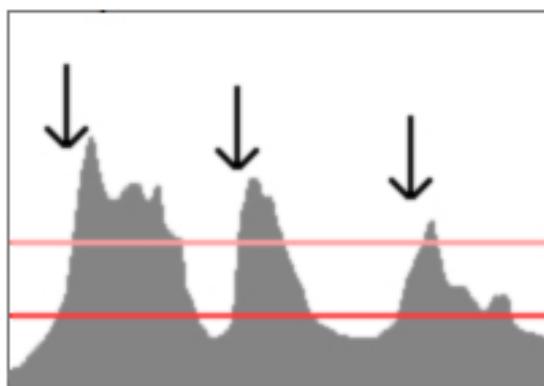


FIGURE 7 – Détection de début de note. En rouge les seuils min et max  
Les flèches représentent les notes détectés

Dans l'environnement les bruits sont intermittent, il est donc intéressant de grouper les événements rythmique en séquence.

Une séquence commence quand le dispositif détecte le début d'une note et termine après une laps de temps sans impacte (de quelques secondes).

Pour éviter de faire des séquences trop longue, dans le cas d'un environnement très bruyant, nous avons décidé arbitrairement de limiter un nombre d'événement maximum dans une séquence (de l'ordre de la centaine).

Pour permettre au dispositif de jouer la même structure rythmique mais à différente vitesse. Le module procède à une normalisation de la durée des rythmes.

Le premier intervalle entre deux notes sera la référence pour toute la séquence, elle aura une valeur '1'. Tous les autres intervalles seront des multiples de cette valeur.

Avec ce système, il est possible de mesurer des divisions de la valeur de référence jusqu'à la quadruple croche (1/16) ce qui apparaît comme une précision suffisante pour le dispositif. Une tolérance est appliquée au calcul des durées des rythmes pour permettre une certaine approximation des rythmes. Jouer rigoureusement plusieurs notes parfaitement égales étant quasiment impossible.

Les séquences détectées sont au final de la forme :



FIGURE 8 – Séquence Rythmique détecté



FIGURE 9 – Deux rythmes pouvant produire la séquence.

### 3.3.2 Détection de Motif minimal

Nous avons beaucoup discuté de la détection mélodie et rythmique avec Hugues GENEVOIS, et nous en avons tiré la conclusion qui serait intéressant d'extraire des séquences rythmiques, les motifs rythmiques récupérant dans ce sequence.

Cette opération s'avérant complexe et peu essayé en Pure Data à cause de la difficulté de stocker et parcourir des données. La détection de motif en utilisant des externals était bien plus facile. Deux externals ont été implémentés, un pour détecter les motifs mélodies et un pour les motifs rythmiques.

Le fonctionnement des externals étant très proches, nous allons ici nous limiter à la description du détecteur de motif rythmique, l'autre external fonctionnant de manière homologue.

L'external prend en paramètre le nombre minimal de note le motif doit être constitué au minimum.

La séquence rythmique est envoyé à la volé, valeur par valeur dès que le module de détection rythmique détecte un nouveau intervalle.

Il stock la valeur à chaque réception et va tester si il n'a pas déjà rencontré un motif contenant cette intervalle. Des qu'un motif est détecté, l'external va envoyer ce motif rythmique pour qu'il soit utilisé pour la synthèse.

A chaque fin de séquence, les valeurs stockées dans l'external sont supprimées, pour accueillir les valeurs de la nouvelle séquence et trouver un autre motif.

Ce module ne permet pas de faire reconnaître la plus long des motifs rythmiques présent dans la séquence mais par soucis de faire interagir rapidement le dispositif avec son environnement, il est préférable de récupérer le motif le plus vite possible même si celui-ci n'est pas le plus long. Pour être sûr d'avoir le plus grand motif obligera à attendre la fin

de sequence et donc à faire interagir le dispositif qu'après que l'émetteur ai fini de jouer.

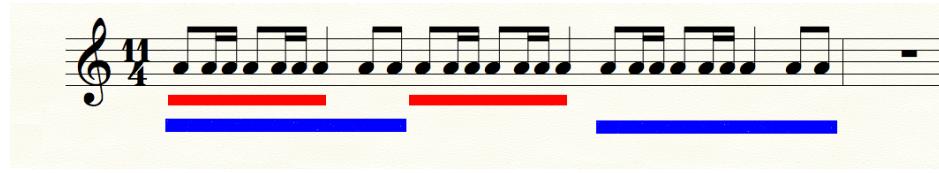


FIGURE 10 – En rouge le motif minimal. En bleu le motif le plus long de la séquence

## 4 Synthèse musical

### 4.1 Synthèse du dispositif à long terme - Modèle physique

La synthèse par modélisation physique consiste à produire des sons à partir d'un modèle informatique décrivant les propriétés physiques d'objets virtuels.

On donne les caractéristiques physiques (dimension, densité, élasticité...) de l'objet que l'on veut simuler, et les caractéristiques de l'objet qui sera l'excitateur.

Puis nous utilisons cette simulation d'instrument pour produire du son. Cela permet d'avoir une timbre de sons très riches et variés.

L'utilisateur de cette synthèse a été évoqué mais ça complexité plus élevée et son développement plus long, nous a fait préférer l'utilisation dans ce projet, d'une synthèse plus basique.

### 4.2 Synthèse implémenté

Pour ce projet, nous avons procédé à un implémentation d'un synthèse hybride utilisant la synthèse additive et soustractive pour générer du son.

La synthèse additive consiste à produire un signal audio complexe en additionnant plusieurs signaux élémentaires (typiquement des ondes sinusoïdal).

La synthèse soustractive quand à elle consiste à produire un signal audio en filtrant certaines fréquences de signaux riches en harmoniques comme le signal carré, triangle ou dent de scie...

## 5 Interface Utilisateur

### 5.1 Première idée d'implémentation

Le paramétrage du dispositif se faisant à distance, les données sont envoyées au dispositif par **wifi**, la carte Udoobea étant équipée.

Comme l'interface graphique et le dispositif communiquaient par **wifi**, il était intéressant de permettre à l'utilisateur de s'abstraire de la contrainte d'avoir Pure Data sur l'ordinateur configurant le dispositif.

Dans l'optique d'une interface portable et simple d'installation, nous avions pensé la faire en Java.

En Pure Data, les communications se fait à l'aide d'une boîte du nom de **netsend** qui permet d'envoyer des messages (Pd) commençant par **send**. En **Tcp** ou en **Udp** selon les paramètres de la boîte.

**netreceiver** la boîte complémentaire de **netsend**, stock les données reçus jusqu'à recevoir la fin du message, et l'envoie sur sa sortie.

Dans un premier temps il nous a donc fallu trouver comment Pure Data convertissait les messages avant de les envoyer si il s'agissait d'une conversion en ASCII ou un autre mécanisme plus complexe. De plus il a fallu trouver le caractère à ajouter au message par **netsend** pour que **netreceiver** puisse connaître la fin du message.

Après plusieurs expérimentations nous avons découvert que le caractère délimiteur était le ';' , ce qui nous a permis d'envoyer des messages extérieurs (d'une programme Java) à Pure Data.

Après quelques tests, nous nous sommes rendu compte que le paramétrage du dispositif ne pouvait pas se faire sans connaissances en Pure Data. Par conséquent, nous avons décidé de changer d'idée et de créer une interface graphique en Pure Data, l'utilisateur serait plus à l'aise avec une interface fait dans un langage qu'il connaît et qui à les mêmes principe et philosophie que le programme embarqué dans le dispositif. L'utilisateur pouvant également l'adapter selon ses besoins.

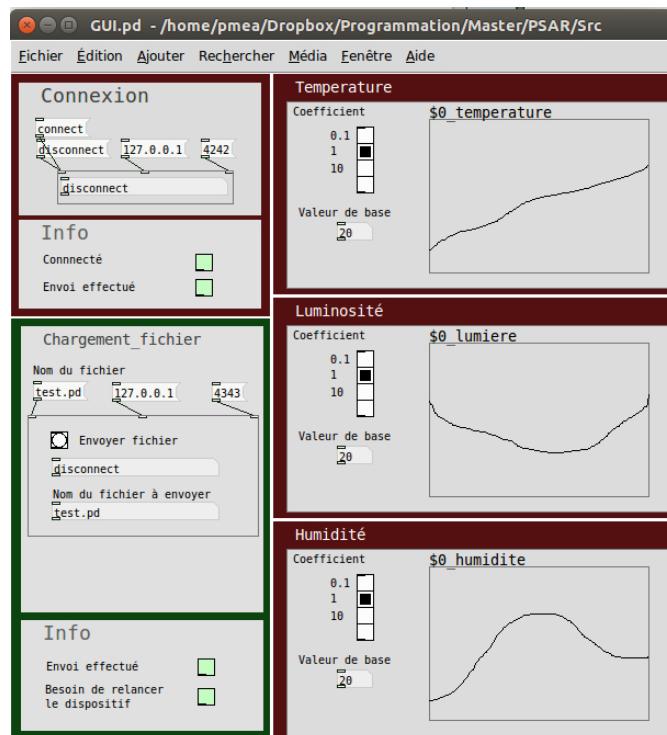


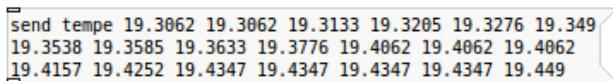
FIGURE 11 – Interface Graphique

## 5.2 Envoie des données des capteurs

Nous avons rencontré plusieurs fois le compositeur pour savoir ce qui lui serait le plus facile et intuitif pour l'envoi de données. Il lui a paru intéressant de dessiner des courbes pour chaque capteurs, pour visualiser exactement ce qui est envoyé au dispositif. Il a également demandé de pouvoir changer l'échelle des courbes pour pouvoir simuler de grandes variations comme des très petites.

**fonctionnement** L'utilisateur peut dessiner les courbes qui désir pour chaque capteurs, quand il a fini, il se connecte au dispositif à ce moment l'interface envoie les données au dispositif qui va les stocker et tant que la connexion ne sera pas rompu par l'utilisateur, (envoi du message de déconnexion). Le dispositif va lire ces données en boucle. Chaque lecture étant espacée d'un certain délai (fixé à 100 milliseconde). Et va les substituer aux données reçues par les capteurs.

Pour envoyer de nouvelles données, il suffira à l'utilisateur de tracer de nouvelles courbes, de se déconnecter et de se reconnecter pour que l'envoi des données est lieu de nouveau.



```
send tempe 19.3062 19.3062 19.3133 19.3205 19.3276 19.349  
19.3538 19.3585 19.3633 19.3776 19.4062 19.4062 19.4062  
19.4157 19.4252 19.4347 19.4347 19.4347 19.4347 19.449
```

FIGURE 12 – Message envoyé par l'interface au dispositif

Le nombre d'échantillon envoyé à chaque fois est fixe dans le patch, elle est fixé à 256 mais il est possible de modifier ça taille en changeant les propriétés tableaux contenant les courbes de l'interface utiliser ainsi que le programme embarqué dans le dispositif.

Pour savoir à quoi correspond chaque liste de données envoyé, le premier mot en début de liste correspond au capteur (tempe pour la température, humid pour l'humidité ...).

## 5.3 Envoie d'objet Pure data

En Pure Data, il est impossible de modifier un patch autrement que par l'interface du logiciel en ajoutant des boîtes ou des connexions. La création du module "Création Musicale" générique, que l'utilisateur pourrait modeler à sa guise serait impossible. La création d'une nouveau module est donc obligatoire pour chaque événement différent.

Nous nous sommes questionné sur comment pouvoir changer la partie "Création musicale" à distance.

La première idée envisagée était de faire un script **Bash** pour envoyer un patch programmé sur l'ordinateur de l'utilisateur vers le dispositif.

Le défaut majeur de cette technique est que cela force l'utilisateur à connaître les bases du **Bash** et les commandes utiles pour maintenir le script.

Ce qui peut paraître évident pour une personne travaillant dans le domaine de l'informatique mais qui est rare chez les utilisateurs de Pure Data, utilisant un langage graphique pour s'abstraire du code informatique. De plus cela oblige à utiliser un autre outil que Pd.

Nous avons vite laissé le problème de coté, par manque de solutions satisfaisantes. En effectuant des testes sur l'envoi des données pour les capteurs, l'idée nous est venu de ne pas envoyer une liste de données (venant des courbes) mais d'envoyer une liste qui serait le contenu d'un fichier, et plus particulièrement d'une fichier Pd. Les fichiers Pure Data étant écrit en `ascii`, il est donc possible de les stocker dans des messages Pd. L'idée est que l'utilisateur écrive un module de "Création musicale" qu'il envoi par le biais de l'interface graphique et de l'autre cote le dispositif le réceptionne et remplace le module de "Création Sonore" par le fichier reçu.

Comme le caractère de séparation dans les fichiers Pd est le point virgule, et qu'il est aussi utilisé en interne pour délimiter les messages, les fins d'envois... La lecture et le traitement de ce genre de fichier n'a pas été sans difficultés, Mais la transfert de boite via Pure Data est possible.

En commencent à implémenter cette méthode nous ne savions si cela allait vraiment fonctionner. Car aucun documentation sur le sujet n'a été trouvé dans les livres traitent du sujet, ni sur les sites de la communauté.

À notre grande surprise, il est tous à fait possible de réécrire des fichiers sans faire planter le patch actuelle chargé, même si le fichier modifié correspond à une boite du patch. Par contre Pure Data ne prendra en compte le changement de contenu de la boite qu'après fermeture et réouverture du patch.

Il n'existe aucune boite permettant d'agir sur les patchs chargés. Il aurait été possible d'écrire un external qui par une commande Bash recharger le patch. Dans la version actuel du projet, l'utilisateur est obligé de d'éteindre et de relancer le dispositif.

Un second problème de l'envoi d'un fichier et que si le patch envoyé n'est pas un fichier Pure Data ou que la syntagme du fichier n'est pas respecté. Quant Pure Data se ré-ouvre avec la nouvelle boite cela produit étrangement pas de message erreur, mais le chargement du patch s'arrête et aucune connexions entre le boite ne sont faite. À ce moment, la seule solution est de ré-installer toute la partie Pure Data du dispositif, ce qui est très contraignant, il faudra donc que l'utilisateur face très attention en utilisant l'envoi de fichiers.

Il aurai été intéressant de programme un external chargé de vérifier la syntaxe du fichier envoyé à l'aide de `regexp` mais faute de temps nous n'avons pas eu le temps de le faire.

## 6 Tutoriel et Documentation

### 6.1 Écriture documentation Pure data

Le projet n'étant qu'une première version, il est important de fournir un documentation la plus complète sur le fonctionnement de tous les patchs et les externals fabriqués. Pour

facilité la compréhension du programme par les personnes qui travailleront sur le projet.

## 6.2 Écriture du Tutoriel d'installation

Comme beaucoup d'utilisateur de Pure Data ont des connaissances limité en informatique. Une documentation a été écrite pour expliquer pas à pas se que l'utilisateur doit faire pour pouvoir reproduire le dispositif.

Elle décrit la partie Hardware avec les différents montages avec les capteurs mais aussi la partie Software pour l'installation de la distribution Linux, des paquets essentiels pour le faire fonctionner correctement, ainsi que la description de l'installation de Pure Data et de **Flext** à partir des sources.

Une image de la carte micro SD sera fourni pour pouvoir mettre en place le dispositif sur la même carte.

De plus un tutoriel est présent pour permettre à un utilisateur néophyte de faire une nouvelle image, si le dispositif venait à changer.

# 7 Pour aller plus loin

## 7.1 Test en environnement réel

Faire des tests sur le son n'est jamais quelques choses de faciles surtout quand il faut les bruits de l'environnement. ces sons étant souvent de faible intensités et de nature très diverse. De plus il est difficile de pouvoir prévoir tous les cas possibles au niveau des combinaisons de sons capté par le micro et les valeurs des capteurs.

Nous avons tout de même essayé de faire le plus de tests possibles que cela soit avec des sons de synthèses ou des enregistrements de environnement. Mais il aurait été pertinent de tester le dispositif plus amplement dans des conditions réels que nous n'avons pas eu le temps de faire faute de temps.

## 7.2 Test énergétique

La carte Udoor malgré tous ses avantages (ses connexions et le micro contrôleur intégré ...) la carte reste très consommatrice d'énergie.

Même si certaines documents non officiel, estime la consommation énergie moyenne de la carte à environs 10 Watts. Les spécifications et les divers documents officiel trouvé parlent seulement d'une consommation maximum de 24 Watts. Ce qui est considérable pour dispositif devant être le plus autonome possible énergétiquement.

Il aurait été intéressant de faire des tests avec d'autres cartes moins gourmandes en énergie.

Comme par exemple combiné une carte Arduino Nano, pour l'acquisition de données, etc une Raspberry Pi V2 pour faire les traitements. Ce dispositif serait plus économique car il consommerait moins de 5 Watts (environ 0.5 Watts pour l'Arduino et 4 Watts pour la Raspberry).

### 7.3 Serveur distant

Dans l'optique de diminuer la consommation d'énergie, il aurait peut être possible d'essayer de centraliser les traitements. La carte aurai pour rôle de collecter les données et les enverrait à un serveur qui lui ferrai les traitements et renverrait la flux audio ou simplement les paramètres pour synthétiser à la carte.

Cela permettre de ne plus se soucier du temps de calcul et permettre d'avoir une supervision des dispositifs, pour savoir ce que chaque dispositif capte et fait comme musique.

De plus il serait possible de partager des capteurs entre plusieurs dispositifs, comme la pression qui varie que très peu sur tout un site où les dispositifs seraient déployé.

La modification des modules de création musical serait également simplifié étant sur le serveur.

## 8 Bibliographie

### Références

- [1] Andéa-Novel Brigitte, Fabre Benoit, and Jouvelot Pierre. *Acoustique-Informatique-Musique*. Presses des Mines, 2012.
- [2] Leipp Émile. *Acoustique et Musique*. Presses des Mines, 2010.
- [3] Thomas Grill. Pure data patch repository, 2008(accessed mars, 2015).
- [4] Hans. pd, 2006 (accessed avril, 2015).
- [5] Adam Hyde. Pure data, (accessed mars, 2015).
- [6] Laurent Millot. *Traitemet du signal audiovisuel, Applications avec Pure Data*. Dunod, 2008.