

# Rapport du Projet PSAR : Dispositif Autonome de Synthèse Sonore

Encadrant : Hugues Genevois  
Cahier de Charges

Pierre Mahé

3 mai 2015

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Contexte Global . . . . .	3
1.2	Presentation de la demande . . . . .	3
1.3	La Carte Udoo . . . . .	4
1.4	Pure Data . . . . .	4
1.4.1	Presentation generale . . . . .	4
1.4.2	Externals . . . . .	5
1.5	Structure du projet . . . . .	5
<b>2</b>	<b>Récupération de l'environnement</b>	<b>8</b>
2.1	Dispositif et capteurs . . . . .	8
2.2	Communication inter plate-forme . . . . .	8
2.3	External pour la communication . . . . .	9
<b>3</b>	<b>Traitement audio</b>	<b>10</b>
3.1	Aquisition audio . . . . .	10
3.2	Traitement du son bas niveau . . . . .	10
3.2.1	Filtrage . . . . .	10
3.2.2	Détecteur de notes . . . . .	11
3.2.3	Bandes de fréquences . . . . .	11
3.3	Traitement du son haut niveau - Extraction des méta-données . . .	12
3.3.1	Mélodie et rythme . . . . .	12
3.3.2	Détection de Motif minimal . . . . .	13
<b>4</b>	<b>Synthèse musical</b>	<b>14</b>
4.1	Modele physique au longterme . . . . .	14
4.2	Synthèse implémenté . . . . .	14
<b>5</b>	<b>Interface Utilisateur</b>	<b>15</b>
5.1	Premiere idée d'implementation . . . . .	15
5.2	Envoie des données capteur . . . . .	15
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 . . . . .	17

<b>7</b>	<b>Pour aller plus loin</b>	<b>17</b>
7.1	Tests en environnement reel . . . . .	17
7.2	Tests énergétiques . . . . .	17
7.3	Serveur distant . . . . .	18
<b>8</b>	<b>Bibliographie</b>	<b>19</b>
<b>9</b>	<b>Annexe</b>	<b>20</b>
9.1	Rappel musical . . . . .	20

# 1 Introduction

## 1.1 Contexte Global

**Maîtrise d'œuvre :** Hugues GENEVOIS

**Maîtrise d'ouvrage :** Pierre MAHÉ

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 de 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 présents dans cet environnement. Le dispositif recueille les informations à l'aide de capteurs divers permettant de récupérer les bruits ambiants, la luminosité, la température.

Dans un premier temps devrait être la base pour une installation artistique, se déroulant en juillet prochain, dans le cadre d'un festival de musique à 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 des comportements et des caractéristiques propres, pour voir l'interaction qu'ils pourraient avoir ensemble et les comportements qui en émergeraient.

## 1.2 Présentation de la demande

Dans le cadre de mon projet PSAR, mon but était de créer un premier prototype du projet 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. Le but du projet est 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 alentours...). 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 `Udoo Quad`. De plus le langage du programme principal doit être le `PureData`, pour assurer une maintenabilité et une extensibilité plus facile.

### 1.3 La Carte Udoo

La carte Udoo est un nano ordinateur embarquant un processeur ARM et la connectique d'un ordinateur classique. Avec une entrée et sortie micro. La qualité étant suffisante pour notre projet.

La caractéristique de cette carte est d'être compatible arduino et d'offrir les mêmes connectique qu'une Arduino Mega ce qui permet de simplifier la réception des capteurs ainsi que le d'utiliser les bibliothèques déjà implémenter par la vaste communauté Arduino.

La carte utilise une carte micro sd pour stocker ses données, 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 simplifiée d'Ubuntu, permettant d'utiliser l'environnement linux.

### 1.4 Pure Data

#### 1.4.1 Présentation generale

Pure Data est un logiciel, open source, de programmation graphique pour la création musicale et multimedia en temps réel. Il permet également de gérer des signaux entrants dans l'ordinateur (signaux de capteurs ou événements réseau par exemple) et de gérer des signaux sortants (par des protocoles de réseau ou protocoles électroniques pour le pilotage de matériels divers).

Pure Data est un système module 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 de base :

- **Les Nombres**, qui peuvent représenter des nombres entiers comme à virgules
- **Les Messages**, qui sont des messages purement textuels

- **Les Symboles**, qui corresponde aux messages mixtes, caractère, chiffre...
- **Les Tableaux**, qui sont uniquement des tableaux de nombres.

Il existe d'autre type de donnée comme les listes qui même avec une représentation interne différente ou message ou symboles sont graphiquement identique a un message (avec comme premier mot `list`). De plus pour changer une liste en message, il suffit de supprimer ce mot clé.

Le Pure Data étant un logiciel orienter interaction, il est possible a tous moment de modifier son patch en ajoutant des boites au cours de l'exécution du programme, il n'est pas nessesaire de relancer le programme. Cela peut permet de modifier le patch en direct sans avoir l'interruption dans le flux audio sortant.

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

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

Un autre atout de Pd est le nombre de d'outils déjà implémenté dans le logiciel et les boites créées par la communauté. Certaines procèdent à des traitements complexes, comme la détection en temps réel de fréquence ou le filtrage performant.

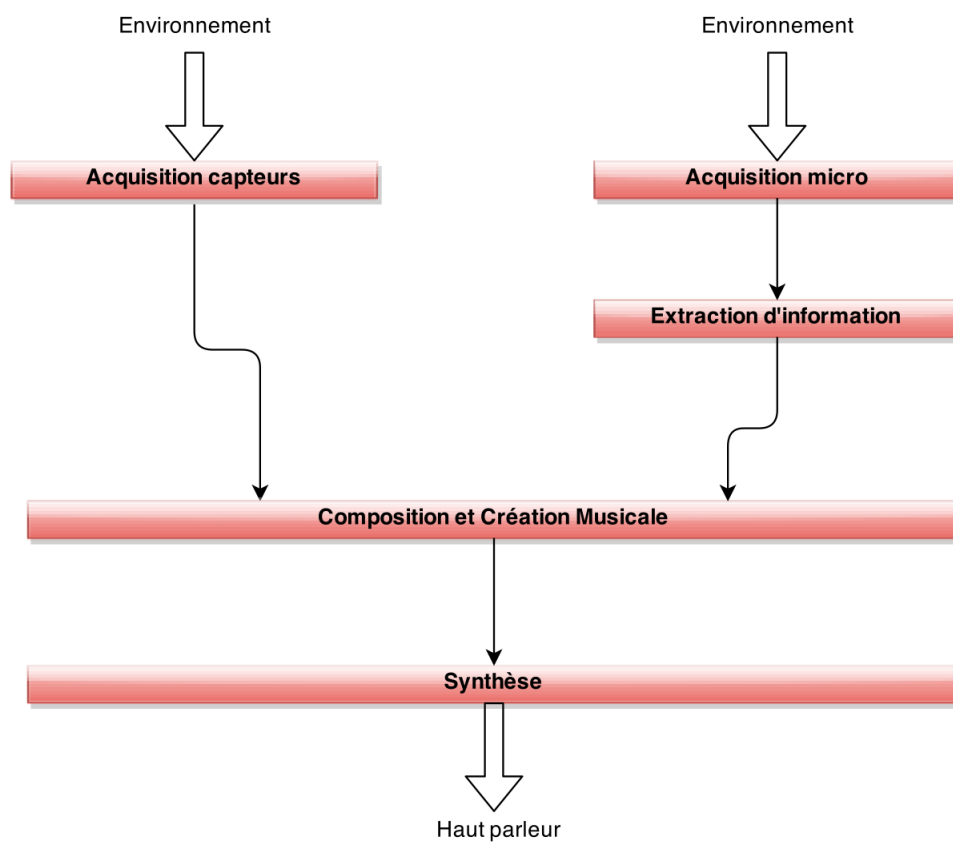
### 1.4.2 Externals

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

Il existe plusieurs API, 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 plus grande 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 à travers le monde), cela a une importance car malheureusement beaucoup d'outils ne sont pas maintenus suffisamment ou sont souvent obsolètes.

## 1.5 Structure du projet

Nous avons décidé de diviser le programme en cinq modules pour la cote synthèse.



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

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

Information collecté par la partie arduino :

- Luminosité
- Temperature
- Humidité

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

Le son capté par le micro est pollué par des sons ambiants, typiquement le bruit d'une route proche ou d'une machine bruyante. Ces bruits sont généralement continus, graves (moins de 100 hertz) donc très énergétiques. Ils peuvent donc fausser les résultats du traitement en aval.

Les bruits naturels à cette fréquence sont très rares dans la nature. De plus musicalement ces bruits sont assez pauvres.

Pour toutes ces raisons il est préférable de les filtrer. Pour les mêmes raisons, nous avons ajouté un filtre pour éliminer les fréquences au-dessus de 6000 Hertz.

Dans la partie extraction, avec le compositeur et le chercheur nous nous sommes questionnés sur les informations pertinentes à extraire pour la composition musicale. En prenant en compte le fait que la carte a une capacité de calcul limitée, cela ne permet pas d'avoir de faire tous les traitements complexes.

La composition musicale est plus la partie propre au compositeur, et correspond à la partie création artistique. Plus qu'une fabrication de ce module, la question était comment pouvoir permettre au compositeur de la créer et de pouvoir la modifier sur le dispositif à distance sans écran ni clavier.

Il y a plusieurs techniques de synthèse musicale certaines sont plus complexes et gourmandes en ressources.

Nous avons choisi une synthèse très simple car la synthèse est très liée à la partie création musicale.

En plus de cette structure implémentée dans le dispositif, une interface graphique a été créée, elle communique à distance avec le dispositif. L'utilisateur peut envoyer des données pour remplacer les informations reçues des capteurs, il peut également changer la partie création musicale.



## 2 Récupération de l'environnement

L'un des avantages de la carte Udoo est d'être compatible Arduino est de proposer les même connectique qu'une Arduino Mega. Cela permet d'utiliser le même langage, d'utiliser les bibliothèques déjà écrites ainsi que les mêmes capteurs. De plus la partie micro contrôleur étant connecté directement à la partie nano ordinateur via un bus special, la vite de communication est plus rapide et avec une plus courte latence par rapport à une connexion usb classique.

Dans les valeurs envoyés par un capteur deux choses sont interessante, la premiere la valeur et la variation de ceci (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 devrait permettre un ajout de capteurs aisé.

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

Pour capteur la luminosité, nous avons utilisé une photo-resistance, son fonctionnement est asses simple, plus la luminosité va etre forte plus la resistance interne de composant va etre faible. Il suffit de recupere la tension au borne de composant pour connaitre l'indice de luminosité.

Malheureusement cette indice ne permet pas de connaitre l'intancité lumineuse en Candela ou en Lux mais permet d'avoir une echelle d'avoir des valeur relavite. Ce qui est suffisant dans le cadre de ce projet.

Pour la Temperature et l'humidité le fonctionnement est tres similaire. Contrairement à la luminosité grace à une formule (dépendent de chaque capteur) il sera possible d'avoir la température ou l'hygrométrie précise.

### 2.2 Communication inter plate-forme

Pour la partie micro-contrôleur l'envoi de données est asses intruitive car il suffit d'écrire sur le port Série (Serial port), avec des primate d'écriture basique.

Pour la partie ordinateur avec le logiciel Pure Data, il faut utiliser une boîte pour se connecter à un périphérique puis on peut lire et écrire sur celui.

La connexion entre l'arduino et Pure data devant etre unique, il a fallu trouver un moyen de pouvoir differensier les données des différent capteur.

Pour communiquer entre les deux parties, j'ai donc défini un micro-protocole qui étiquette les données envoyés.

Tous les X secondes le programme arduino lit les valeurs des capteurs et envoie un message 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 sur le port série avec un nom de capteur pas encore utilisé et c'est dans la partie Pure data que nous traiterons la nouvelle étiquette.

Puis pour il nous est venu l'idée de permettre à Pure data de pouvoir changer le délai entre chaque envoi de valeur. Pour cela il a fallu permettre à l'arduino de pouvoir recevoir des données avec des messages ayant la même forme que l'envoi de donnée avec l'étiquette **delay**.

C'est à ce moment là que nous avons découvert un vice de Pure Data, étant un langage très faiblement typé. Les données reçues 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 entiers (valeur des capteurs envoyé par la parité Arduino).

Au premier abord nous avons voulu utiliser l'objet **PackOSC** qui permet à la base de convertir un message quelconque en commande OSC (**O**pen **S**ound **C**ontrol). Ce protocole étant un protocole avec un codage ascii nous pouvions l'utiliser pour encoder et decoder les communications avec l'arduino.

Mais cela aurait été une manipulation de l'objet de base de plus cela pourrait porter à confusion car nous ne gérons en rien le protocole.

Nous avons discuté avec Hugues GENEVOIS et nous avons utilisé de programmer nos propres externes.

## 2.3 External pour la communication

En s'inspirant du fonctionnement des objets **PackOSC** et **UnPackOSC**, nous avons programmé deux externes, un qui reçoit un message et qui le transforme en tableau d'octets (en code ascii plus particulièrement) en ajoutant le code ascii d'un point virgule à la fin.

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 PureData ce caractère est généralement le délimiteur de fin de chaîne, la fin de liste et de tableau ainsi qu'interne pour fin des messages tcp ainsi que la fin d'une lecture de fichier.

## 3 Traitement audio

### 3.1 Aquisition 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 frequency et les haut fréquence. Nous avons chercher a savoir determiner les frequences de coupure a partir du quelle il n'y as plus d'information pertinente. Nous nous somme rendu compte qu'au dessus de 6000 Hertz il n'y avait plus de son avec une fondamentale a cette haute, et qu'il n'y avait que des harmonique de son plus grave. (Voir l'annexe rappel musical pour les explications sur les fondamentales et les harmoniques). Pour les sons tres graves, il n'existe pa de sons produits par la nature à des fréquence inférieur à 100 Hertz. Nous avons donc ajouter un filtre a notre acquisition pour pouvoir supprimer les bruits qui pour nous sont des bruits parasites.

### 3.2 Traitement du son bas niveau

#### 3.2.1 Filtrage

Une inforation qui nous paraissé esentiel a extraire etait les notes et les melodies qui sont jouer dans l'environement.

Il est asses facile de determiner la frequency principal d'un signal, il suffit de prendre la forme generale du signal. Il existe plusieurs boites en Pd pour le faire tel que `Fiddle` ou `Signum`.

Pour detecter les notes jouer simultanement ou suivre plusieurs de melodies simultanement est tres quasiment impossible et fait encore partie l'objet de recherche. L'une des meilleurs compromis entre le temps de calcul et la fiabilité de la sortie est de faire En Pd il existe des boites pour déterminer la frequency pour cela nous avons pensé a utiliser la transformé de fourrier (voir Annexe) pour déterminer l'ensemble des notes jouer.

Nous avons implémenter un programme en C pour pour faire une transformée Fourier rapide (FFT), dans l'optique d'en faire une boite pour l'utiliser dans Pure Data mais nous nous sommes rendu compte que ce proceder etait beaucoup trop gourmand pour la carte. Meme si la complexité de la FFT est en  $N\log(N)$ , avec  $N$  le taux d'echantillonnage du signal (nombre de point du signal). Notre signal est echantilloné a 44k Hertz il y a donc 44100 points a traiter par seconde ce qui est trop lourd pour la carte (si nous voulons pouvoir faire d'autres traitements en parallèle).

Apres ces tests insatisfesant, nous nous sommes donc orienter dans une autre direction, le chercheur nous a conseiller de partager le spectre sonore en plusieurs

bandes de fréquences a l'aide de filtre passe-band et de suivre de deteter la fréquence la plus principal de chaque bande, avec un `fiddle`.

Après avoir implémenté en pure data ce mecanisme un nouveau probleme est apparu. Les filtres ne coupant pas parfaitement les frequences en dehors de sa bande, les objet `fiddle` detecter parfois des frequences n'appartiennent pas a leurs bande de fréquence. Il y avait donc plusieurs filtre qui detecter donc la même fréquence ce qui etait asses problematique.

Pour resoudre ce second probleme nous avons utilisé des `noiseGate`.

Une `noiseGate` est un element souvant utliser en musique, il correspond simplement a un boite qui ne laisse passer que les signals sonore d'une certaine intensité, si le signal est trop faible le signal en sortie de la boite est egale a 0. Cela permet de supprimer les sons venant des autres bandes de frequences, etant atenuer par les filtres, ces sons ne passe pas la `NoiseGates`, ce qui resolvé le problemee de la detection de fréquence erroné.

### 3.2.2 Détecteur de notes

Ce dispositif était bien mieux que les précédent mais ne permettait pas de suivre les melodies qui etait etaler sur plusieurs bandes de fréquences. Ce qui se traduisé par la detection de morceau de melodie souvant haché.

Nous avons donc decider d'abandonner l'idée de suivre plusieurs melodie simultanement et de ne garde que la melodie principal du signal.

Et de ne n'utilsier les notes récupérer par les filtres uniquement pour savoir quelle note sont jouer pendant un labse de temps.

Ce choix choix peut semblé un peu etonnant mais il est tres important pour le dispositif de savoir quel note son jouer, si la melodie qu'il capte utiliser uniquement deux notes, il serait tres disonnant que le disposirif en joue une troisieme note qui n'a rien a voir avec les deux premieres.

Donc implementé un compteur de notes qui collecte les notes reconnues a la sortie des filtres. A intervalle régulier le compteur va nous donner la fréquence des notes joué. Pour ne pas faire trop de dissonance il suffira au module de synthese de jouer des note tiré au hasard dans la liste de note en tenant compte de la fréquence d'apparition d'une note.

### 3.2.3 Bandes de fréquences

Avec ce systeme de detection de note, ils est possible de savoir la fréquence d'appartion des notes mais il n'est pas possible de savoir la hauteur des sons capté. Nous avons donc ajouter un module pour mesurer l'amplitude du signal sur chaque

bande de fréquence pour savoir où sont reparti les notes sur les bandes de fréquences, si les bruits capté par le micro sont plutôt aigus ou graves.

### **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.

Nous avons donc voulu implémenté un module homologue à la détection de note, mais pour déterminer les rythmes. Pour synthétiser du son avec des rythmes proche de l'environnement.

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

Après en avoir discuté avec notre encadrant, il nous est apparu que pour connaître pouvoir détecter un début de note, la manière de procéder est de détecter les impacts, généralement caractéristique d'un début de note. Pour cela nous avons calculé l'énergie du signal (voir annexe). Il suffit ensuite de regarder son évolution, si une augmentation brusque apparaît, cela induit la présence d'un impact et donc du début de note.

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

Dans l'environnement les bruits sont intermédiaires, il nous est apparu que il serait intéressant de grouper les événements rythmiques en séquence. Une séquence commence quand le dispositif détecte le début d'un note et termine après un laps de temps sans impact (de quelques secondes). Pour éviter de faire des séquences trop longues 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 différentes vitesses. Nous avons décidé de normaliser notre notation de rythme. 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 cette valeur jusqu'à la

quadruple croche (1/16) ce qui nous parrait etre une precision suffisante.  
 Nous avons rajouter une certaine tolérante au calcul de valeur pour permaitre une certaine approximation dans les rythmes. Il est quasiment impossible de jouer rigoureusement plusieurs notes parfaitement égales.  
 Les sequences dete ter son au final de la forme :

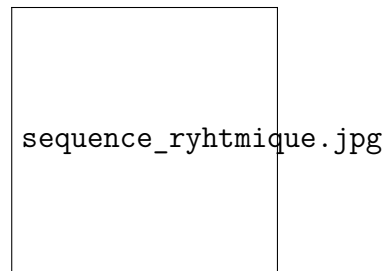


FIGURE 1 – Sequence Rythmique detecté

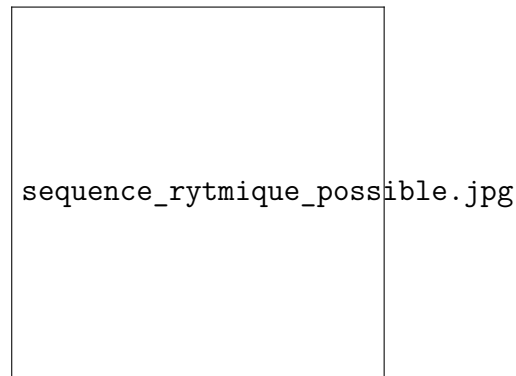


FIGURE 2 – Deux ryhtmes pouvant produire la sequence

### 3.3.2 Détection de Motif minimal

Nous avons beaucoup discuter de la detection melodie et rythmique avec Hugues GENEVOIS, et nous en avons tirer la conclusion qui serai interessant d'extraire des sequences rythmes les rythme les motifs melodie et rythmique qui revienne dans ce sequence.

Cette operation s'averant tres complexe en Pure Data a cause du fait qu'il est compliquer de stocker des données. Nous avons donc decider de faire deux externals un pour les motifs melodie et un pour le motif rythmique.

Nous allons vous decrire le fonctionnement detailler de l'external de detection de motif rythmique, il sauf savoir que le l'external pour de detection de motif melodie fonctionne de maniere homologue.

On donner en parametre a l'external le nombre minimal du motif. Et nous lui

envoyons la sequence rythmique à la volé, des que le module de tecton rythmique détecte un nouveau intervalle il est directement envoyer au module de detection de motif rythmique.

Il stock les valeurs est a chaque recption de nouvelle valeur va tester si il n'a pas deja rencontre un motif contenant ce rythme. Des qu'un motif sera detecter l'external va envoyer ce motif rythmique pour qu'il soit utiliser pour la synthese.

Ce module ne permet pas de faire reconnaitre la plus long des motifs rythmiques present dans la sequence mais pour pouvoir interagir rapidement avec son environnement, il est preferable d'avoir un motif court, le plus vite possible plutot qu'un motif plus grand mais qu'il faille que la sequence soit terminer et donc que l'interaction avec le dispositif soit moindre. Le dispositif n'ayant la plus longue sequence uniquement une fois que l'emetteur a fini de jouer.

## **4 Synthèse musical**

### **4.1 Modele physique au longterme**

### **4.2 Synthèse implémenté**

## 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 udoo étant équipée d'un module wifi.

Comme l'interface graphique et le dispositif communiquaient pas le wifi, nous sommes dit qu'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'avoir une interface portable et simple d'installation, nous avons pensé au Java.

En PureData, les communications se font à l'aide d'une boîte du nom de **netsend** qui permet d'envoyer des messages (Pd) commençant par **send** et envoie les données à la fin du message. En Tcp ou en Udp selon les paramètres de la boîte.

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

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

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

Après quelques tests nous sommes dit que le paramétrage du dispositif ne pouvait pas se faire sans connaissances en Pure Data. Par conséquent, nous décidons de changer d'idée et de fournir une interface graphique faite en Pure Data, l'utilisateur sera plus à l'aise avec une interface faite dans un langage qu'il connaît et qui a la philosophie que le programme embarqué dans le dispositif. L'utilisateur pourra également l'adapter selon ses besoins.

### 5.2 Envoi des données capteur

Nous avons rencontré plusieurs fois le compositeur pour savoir ce qui lui serait le plus facile pour l'envoi de données. Il lui a paru intéressant de dessiner des courbes pour chaque capteur. Il a pensé également que pouvoir choisir changer la précision des courbes pour pouvoir simuler de grandes variations ou au contraire des très petites variations.



**fonctionnement** L'utilisateur peut dessiner les courbes qui desir pour capteur, quand il a fini il se connecte au dispositif a ce moment l'interface envoie les doonnées au dispositif qui va les socker et tant que la connectio n'est rompu par l'utilisateur va les lire en boucle et s'en servir comme si ses informations etait celle des capteurs. Pour envoyer de nouvelle données, il suffira a l'utilisateur de tracer de nouvelle courbe, de se deconnecter et de se reconnecte pour que l'envoi de face a nouveau.

### 5.3 Envoie d'objet Pure data

En Pure Data, il est impossible de pouvoir modifier un patch autrement que par l'interface du logiciel en ajoutant des boites ou des connections. Et donc la creation du module "Creation Musicale" générique que l'utilisateur pourrai modelé a ca guise serait impossible a programmer. La Creation d'une nouveau module serait obligatoire pour chaque evenement.

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

La premiere chose envisage a etait de faire un simple script **Bash** pour envoyer un pathc programmé par sur l'ordinateur de l'utilisateur vers le dispositif. Le default majeur de cette technique est le cela force l'utilisateur a connaitre les commandes utile pour maintenir le script et les bases du ssh. Ce qui peut paraitre evident pour une personne travaillant dans le domaine de l'informatique mais qui est rare pour les utilisateurs de PureData. Utilisant un langage graphique pour s'abstraire du code informatique. De plus cela oblige à utiliser un autre outils que Pd.

Nous avons vite laisser le problème de coté, par manque de solutions satisfaisante. En effectuant des teste sur l'evoi de données pour les capteurs l'idée mes venu de ne pas envoyer une liste de données (venant des courbes) mais d'envoyer une liste qui serai le contenu d'un fichier, et plus particulièrement d'une fichier Pd. L'idée est que l'utiliateur ecrive un module de creation qu'il envoi le fichier par l'interface graphique et que de l'autre cote, le dispositif le recpetion et ecrive est remplace le module de "Creation Sonore" par le fichier reçu.

Comme le caractere de ceparation dans les fichiers Pd est le point virgule, et que ce point virgule est aussi utiliser en interne pour delimiter les messages, les fin d'envoi... Cela a lecture et le traitement de ce genre de fichier n'a etait sans difficultés.

De plus quand nous avons commencer a implementer cette methode nous ne savions pas cela aller vraiment marcher.

A notre grande surprise il est tout à fait possible de réécrire des fichiers sans avoir planté le patch actuellement chargé. L'inconvénient est que Pure Data ne recharge le contenu de la boîte qu'après fermeture et reouverture du patch le contenant. Il aurait été possible d'écrire un external qui par une commande **Bash** recharge le patch, dans la version actuelle du projet, on est obligé de l'éteindre et de relancer le dispositif.

Un second problème a été trouvé si le patch envoyé n'est pas un fichier Pure Data ou que la syntaxe du fichier n'est pas rigoureusement suivie. Quand Pure Data se reouvre avec la nouvelle boîte cela produit étrangement pas de message d'erreur, le chargement du patch s'arrête et aucune connexion entre la boîte ne se fait. À ce moment, la seule solution est de réinstaller toute la partie Pure Data du dispositif. Il aurait été intéressant de programmer un external chargé de vérifier la syntaxe du fichier envoyé à l'aide de **regex** 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 une documentation la plus complète sur le fonctionnement de tous les patches et les externals fabriqués. Pour faciliter la compréhension du programme par les personnes qui travailleront sur le projet.

### 6.2 Écriture du Tutoriel d'installation

Comme beaucoup d'utilisateurs de Pure Data ont des connaissances limitées en informatique, nous avons écrit une documentation qui explique pas à pas ce que l'utilisateur doit faire pour pouvoir reproduire le dispositif. Elle décrit la partie Hardware avec les différents montages des capteurs mais aussi la partie Software pour l'installation de la distribution Linux, des paquets essentiels pour le faire fonctionner correctement, ainsi qu'une description de l'installation de Pure Data et de FlexT à partir des sources.

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

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

## 7 Pour aller plus loin

### 7.1 Tests en environnement reel

Faire des testes sur le son n'est jamais quelques choses de faciles surtout quand il faut les buits envivorant. ces sons etant souvnat de faible intensiter et de nature tres diverse. Nous tous de même essayer de faire le plus de teste possible que cela soit de son synthetiser par ordinateur ou des enregistre d'environnement. Mais il aurait etait interse de teste les dispositif plus emplement dans des conditoon reeln, que nous n'avons pas eu le temps de faire faute de temps.

### 7.2 Tests énergétiques

La carte Udoo malgré tous c'est avantage et par toute ca connection et le micro controleur pour un nano-ordinateur. Même si sur certaine document non officiel, estime la consomation moyenne d'énergie de la carte a environs 10 Watts. Les specifications et les divers documents officiel trouvé parlent d'une consomation maximum de 24 Watts.

Ce qui est considerable pour dispositif devant etre le plus autonome energetique energetiquement.

Il aurait etait interessant de faire des testes avec d'autres cartes moins gourmandes en energie.

Comme par exemple combiné une carte Arduino Nano pour l'aquisition de données et un Raspberry Pi V2 pour les traitement, consommerait moins de 5 Watts (environ 0.5 Watts pour l'Arduino et 4 pour la Raspberry).

Des testes des programmes sur d'autre plateforme et voir si la traitement pourrai se faire en temps reel.

### 7.3 Serveur distant

Dans l'optique d'etre de diminuer la consomation d'énergie, il aurait peut etre possible d'essayer une implementation centraliser. La carte n'aurai pour role que de collecer les donné pour les envoyer a un serveur qui lui ferai les traitement et renveré la flux audio ou simplement les parametres a la carte pour synthetiser du son.

Cela permaittre de ne plus ce soucier du temps de calcul et permaittre d'avoir une supervision des dispositifs. De plus il serait possible de partagerdes capteurs entre plusieurs dispositifs, comme la pression qui varie que tres peu dans un endroit donné.

La modification des modules de creation musical serai egalement simplifié.

## 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. *Traitement du signal audiovisuel, Applications avec Pure Data*. Dunod, 2008.

## **9   Annexe**

### **9.1   Rappel musical**