Résumé

Ce mémoire présente le travail réalisé durant mon stage de seconde année de Master Informatique Multimédia[1] au SCRIME[2] sur le projet de percussion aérienne (également appelée 'percussion virtuelle'), qui vise à créer un nouvel instrument. Il s'agirait d'une sorte de batterie dépourvue de toms, constituée uniquement d'une paire de baguettes et de capteurs.

Ce stage professionnel a pour objectif l'amélioration de l'instrument existant. Cela passe par l'ajout d'accéléromètres, et donc la création d'une bibliothèque dédiée, ainsi que de tous nouveaux algorithmes prédictifs de détection des coups.

[1] réalisé à Bordeaux

[2] Studio de Création et de Recherche en Informatique et Musique Electroacoustique

Table des matières

Chapitre 1

Introduction

1.1 Cadre du projet

Le SCRIME est un laboratoire de recherche basé sur la cohabitation et la coopération d'artistes et de scientifiques afin de favoriser le transfert de connaissances scientifiques pour les uns et d'une expertise musicale pour les autres. Les membres du SCRIME sont donc des chercheurs en informatique musicale du LaBRI[1] et des compositeurs majoritairement issus de la section électroacoustique du Conservatoire National de Région de Bordeaux.

La recherche fondamentale menée au SCRIME concerne principalement la modélisation informatique du son (analyse, transformation et synthèse) ainsi que l'aide à la composition ; tandis que la recherche appliquée se tourne d'avantage vers des interfaces musicales (instruments virtuels, logiciels pédagogiques, méta-instruments).

Le projet Métamorphose a été créé en 2000 par Christophe Havel, compositeur membre du SCRIME. L'idée est de concevoir et présenter un spectacle où des instrumentistes traditionnels joueraient sur des méta-instruments électroacoustiques. Ces méta-instruments ressemblent généralement à des instruments classiques mais ils jouent des sons de synthèse.

L'originalité du projet vient du fait que l'on associe les capacités de musiciens confirmés à un instrument qui sort totalement de l'ordinaire. Chaque instrumentiste joue donc virtuellement, avec ses réflexes physiques et auditifs, à l'aide d'une série de capteurs adaptés à son instrument. Les informations ainsi recueillies sont envoyées à un ordinateur qui génère des sons. Le groupe se composait initialement

[1] Laboratoire Bordelais de Recherche Informatique

d'une guitare, d'un saxophone, d'un thérémin[2] et d'une percussion.

Pour les deux premiers instruments, la captation d'information fut confiée aux instruments MIDI correspondants déjà existants, quand au thérémin, il fournit déjà un son utilisable. Les informations de jeu collectées sont donc envoyées à un Mac (via MIDI) pilotant les sons générés par le logiciel Max/MSP[3].

Le cas du percussionniste est en revanche plus délicat car il s'agit de capter et de reconnaître une gestique en trois dimensions, sans qu'il y ait besoin de frapper des endroits précis. Les *pads* MIDI couramment employés par les batteurs voulant disposer de sons de synthèse, réduisent considérablement la finesse du jeu, la ramenant à un simple déclenchement par contact, le volume étant déjà pré réglé. De plus ils génèrent des bruits percussifs parasites dont on aimerait se passer.

Il est donc nécessaire de trouver une solution permettant de s'affranchir de ces problèmes, tout en rendant compte de la richesse gestuelle offerte par un espace en trois dimensions totalement libre. Le projet de percussion aérienne a été lancé la même année par un groupe de scientifiques au sein du SCRIME. Plusieurs étudiants se succèdent entre 2001 et 2003, où eut lieu la première représentation publique.

La percussion aérienne est un système de captation du geste intégré sur des baguettes de percussion traditionnelles. C'est aussi le nom de l'instrument dont il sera question dans ce mémoire. Il s'agit d'un ensemble de capteurs disposés de manière à recueillir les paramètres pertinents du geste de l'instrumentiste. Plus concrètement, c'est un système qui va interpréter le mouvement des baguettes afin de déterminer si celui-ci correspond à un coup. Le percussionniste va simuler des coups dans l'air sur des *toms* invisibles, et un son sera produit au moment de l'impact 'virtuel'.

Le stage s'est déroulé sous la direction de Myriam Desainte-Catherine, en collaboration avec le compositeur Christophe Havel. Il s'inscrit donc dans la continuité des recherches déjà menées, puisque son principal objectif est d'affiner la reconnaissance des coups. Bien que cela soit officiellement un stage professionnel, il s'apparente plus à un sujet de recherche, puisque totalement expérimental. En effet, Christophe Havel avait une idée théorique du résultat à atteindre, sans

[2]Un thérémin est un instrument générant deux signaux de fréquences élevées se combinant pour former un son. L'instrumentiste utilise son corps pour altérer directement la fréquence de ces signaux et ainsi modifier le son.

[3]Logiciel modulaire de traitement du signal en temps réel

FIG. 1.1 – Un exemple de jeu avec la percussion aérienne

toutefois savoir si cela serait techniquement possible.

Un point important à souligner est que je n'ai pas travaillé seul sur le sujet. J'ai été épaulé par Joseph Sanson, élève en première année de Master Mathématiques Appliquées à Caen, dont le rôle sera évoqué plus tard.

1.2 Etat de l'art

Très peu de travaux de recherche ont été réalisés sur ces instruments dits virtuels. Toutefois, si nous considérons le cas très particulier de la batterie, nous pouvons en lister quelques uns :

Le radio bâton est le premier système qui a été capable de suivre la position de deux baguettes dans un espace à trois dimensions (Figure 1.2). Max Matthews l'a réalisé pour contrôler, par le geste, le déclenchement d'évènements musicaux. Cependant, ce système n'a pas été initialement conçu pour capter le jeu réel d'un percussionniste, son but premier n'étant pas de réaliser une percussion virtuelle.

La percussion virtuelle du Gnem [2] introduit une nouvelle dimension dans la reconnaissance du geste : l'utilisation de caméras. Le champ de vision de la caméra est divisé en de multiples zones (Figure 1.3). Un son est joué lorsqu'un mouvement est détecté dans l'une de ces zones. Toutefois, compte tenu de la vitesse d'échantillonnage très faible du système optique (25 à 30Hz), cette application n'est pas destinée à capter le geste naturel d'un percussionniste.

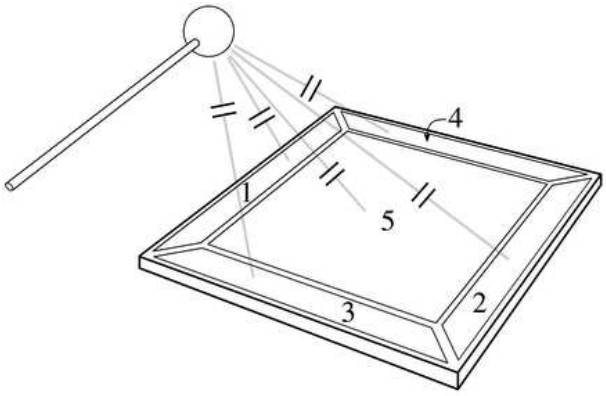

FIG. 1.2 – Schéma du radio bâton

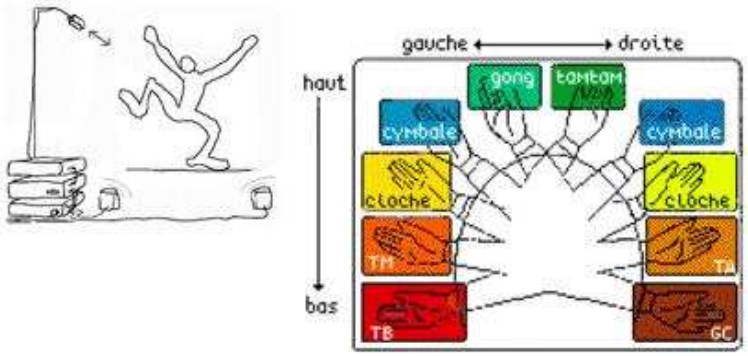

FIG. 1.3 – Système de percussion virtuelle du Gnem

Les batteries MIDI , bien qu'elles ne correspondent pas vraiment à un instrument virtuel, ne sont pas à oublier. En effet, malgré toutes leurs lacunes, elles restent un des moyens les plus fiables pour jouer un son synthétique.

La *'Nintendo Wii'* , console nouvelle génération, qui devrait arriver très prochainement propose une manette très particulière incluant capteurs de position et d'accélération. Un jeu de chef d'orchestre ainsi qu'une simulation d'instrument ont été annoncés. Il est encore un peu tôt pour juger de la qualité et du sérieux d'un tel produit, mais nous pouvons noter que ce genre d'interface commence à séduire les grands constructeurs.

1.3 Domaine d'application

Le domaine d'application est très restreint car cet instrument a été développé pour être utilisé exclusivement dans le cadre du projet Métamorphose. Il n'est donc utilisé qu'à l'occasion des rares concerts donnés. Il ne fait l'objet d'aucun prêt ou partage et n'est pas non plus développé dans un but commercial.

1.4 Notion de 'coups' ou 'd'impact'

Il est important de définir ici une notion que l'on retrouvera tout au long de ce rapport, qui est celle de 'coup' ou 'd'impact'. Pour comprendre cette notion il faut imaginer un batteur utilisant une batterie classique. L'impact arrive au moment où la baguette touche la peau du *tom*. Si on retire maintenant cette surface et que l'on analyse le mouvement de la baguette, l'impact arrive au moment où la baguette atteint sa position verticale minimale, ce qui se traduit par une vitesse et une accélération nulle. C'est ce moment là que nous appellerons 'impact' ou 'coup' et que nous allons chercher à détecter.

Chapitre 2

Existant

2.1 Présentation globale

L'instrument se présente donc ainsi (Figure 2.1). Un Pc sous Linux récupère les coordonnées renvoyées par les 'flock of birds' grâce à la Libflock. L'accélération est calculée d'après ces valeurs et est utilisée dans un algorithme de détection des impacts. Lorsqu'un impact est détecté, les coordonnées de la baguettes ,ainsi qu'un fichier préalablement généré par l'éditeur d'instruments, sont utilisés pour déterminer si un module à été frappé. Dans ce cas, un message est transmis au Macintosh qui est ,quand à lui, chargé de toute la partie synthèse sonore, grâce au logiciel Max/MSP.

2.2 Présentation détaillée

2.2.1 Les 'flock of birds'

Au niveau des capteurs, le projet a connu une première phase avec des radio bâtons. Toutefois, ceux-ci ont très rapidement montré leurs limites. En effet avec une vitesse d'échantillonnage très faible[1] ainsi qu'une surface de jeu réduite, ils ont vite été remplacés par les 'flock of birds' (Figure 2.2).

Ce système développé par Ascension Technology [1] propose un taux d'échantillonnage bien plus intéressant (100Hz) ainsi qu'une surface de jeu plus étendue (une sphère d'un mètre de rayon). Les capteurs sont montés au bout des baguettes. Une base émet un champ électromagnétique sphérique, et peut ainsi donner la position de un ou plusieurs capteurs dans l'espace. Ce nouveau matériel a permis

[1] 50Hz à l'époque, mais les travaux récents ont remis le radio bâton à jour en le dotant d'une carte son pour l'échantillonnage

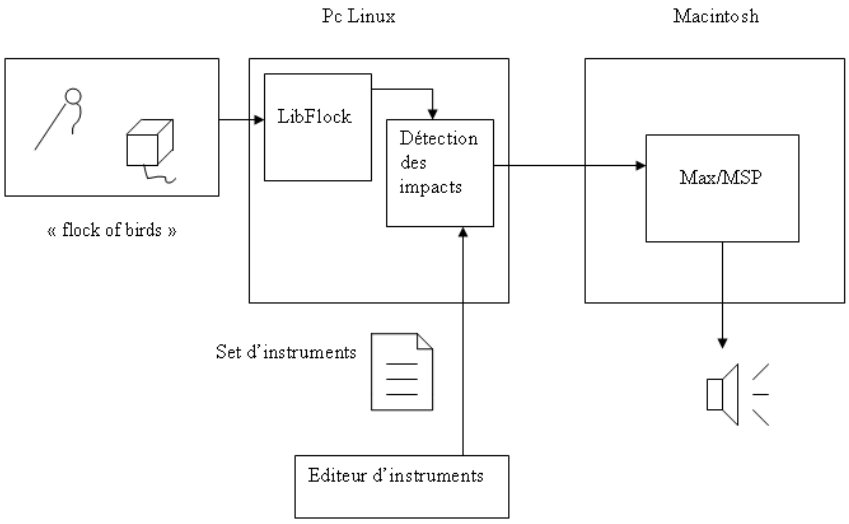

FIG. 2.1 – Schéma représentant l'état du système avant le début du stage

de considérablement améliorer le système de détection des coups par rapport au radio bâton.

Un détail qui aura son importance plus tard, ces capteurs utilisent une connectique série (RS232) pour communiquer avec l'ordinateur.

FIG. 2.2 – Flock of birds montés sur des baguettes classiques

2.2.2 La Libflock

Cette bibliothèque dédiée à l'utilisation des 'flock of birds' a été développée par Julien Vilain, qui était ingénieur au SCRIME. Ecrite en C, elle assure toutes les étapes nécessaires à leur utilisation, de l'initialisation du matériel à la récu-

pération des coordonnées sous forme de valeurs exploitables. Elle a toutefois été développée pour une utilisation exclusivement sous Linux.

2.2.3 La détection des impacts

L'algorithme de détection des impacts utilisés par l'instrument travaille sur l'accélération, calculée par dérivation successive des positions puis de la vitesse. Cette approche a été dictée par le besoin de prédire les impacts, conséquence directe de l'utilisation de Max/MSP pour la génération de son.

Max/MSP est un logiciel de traitement sonore très complet qui permet de partir d'une sinusoïde, d'y appliquer de nombreux traitements paramétrables en temps réel afin d'arriver à un son unique. La génération de son nécessitant de nombreux et coûteux calculs, elle n'est en fait pas instantanée. Le compositeur Christophe Havel a évalué à 10ms le temps nécessaire à la génération d'un son. Nous comprenons maintenant l'importance de détecter les coups juste avant qu'ils ne se produisent. Un retour auditif asynchrone se révèlerait trop inconfortable pour l'auditeur ainsi que pour le percussionniste.

Revenons à notre détection des impacts. Ici cela se fait par simple comparaison de l'accélération à différents seuils pour déterminer quand commence et se termine la phase d'accélération (au début du coup, quand la baguette part d'une position haute et accélère sous l'impulsion de l'instrumentiste), et quand débute la phase de décélération (peu avant l'impact, le percussionniste freine lui même la baguette pour effectuer un retour, puisqu'il n'y a pas de surface à frapper). La prédiction du coup intervient donc dès que l'on détecte une décélération juste après une accélération.

2.2.4 L'éditeur d'instruments

La possibilité d'utiliser un espace en trois dimensions pour jouer a apporté une incroyable liberté, mais aussi un manque de repères dans la définition et l'utilisation de cet espace. La première chose à faire est de découper cet espace en 'modules' distincts. Nous pouvons donc associer un son différent à chaque module et, par exemple, recréer un xylophone virtuel.

L'utilisation de cet espace en trois dimensions ne se limite pas à recréer des instruments 'invisibles'. Il n'y a pas de *toms* et donc pas de repère visuel ou physique. De plus il n'y a pas de *peau* pour arrêter la baguette. Chaque volume peut être 'frappé', mais aussi traversé. Nous pouvons imaginer déplacer une baguette à l'intérieur d'un volume et ainsi générer un son 'continu' modulé par le déplacement. Une image plus parlante serait celle d'une caisse virtuelle remplie de bou-

lons produisant des sons métalliques en fonction du déplacement de la baguette. De ces possibilités naît toute la puissance artistique de la percussion aérienne.

Il a donc fallu codifier toutes ces possibilités. Cet ainsi qu'un groupe d'étudiants a défini trois types de volumes, qui sont en quelques sorte des instruments à part entière (un peu comme la caisse claire, la cymbale et la grosse caisse d'une batterie classique) :

- Instruments 'frappés' (percussifs) : les sons sont générés par des impacts. Lorsqu'un coup est détecté grâce à l'algorithme dédié, on récupère la position de la baguette. Si celle-ci se trouve dans le volume d'un instrument 'frappé', alors un son correspondant est joué, modulé par la vitesse au moment de l'impact.
- Instruments 'secoués' (type maracas) : les changements de vitesse et/ou de direction à l'intérieur du module produisent des sons. L'amplitude du changement de vitesse détermine l'amplitude du son produit.
- Instruments 'toggles' (continus) : les sons sont générés en fonction de la position dans l'espace de l'instrument. L'entrée d'une baguette dans le volume de l'instrument déclenche un unique son continu dont le volume est fonction de la vitesse d'entrée. La sortie de la baguette stoppe le son.

Ces instruments sont caractérisés par :

- Le type de l'instrument (un instrument peut être de plusieurs types).
- La forme de l'instrument (cylindre, cube, sphère).
- La position de son centre et ses dimensions.

En 2003, Sébastien Lebreton développe un éditeur d'instruments virtuels avec une interface en trois dimensions (Figure 2.3). Lorsque le 'clavier' est terminé, l'éditeur génère un fichier que nous appellerons *set d'instruments*, qui sera utilisé par l'instrument pour déterminer, lors d'un impact, quel module a été frappé et quel son devra être joué.

2.3 Communication avec le Macintosh

La communication avec le Macintosh utilise le protocole OSC ; Max/MSP étant parfaitement habilité à traiter ce genre de données. Les paquets transitent par une liaison éthernet. Ils contiennent le numéro du module frappé, la position de l'impact, la distance entre le centre du module et l'impact (pour réaliser un effet de distorsion) ainsi que la vitesse maximale atteinte durant le coup qui déterminera le volume sonore.

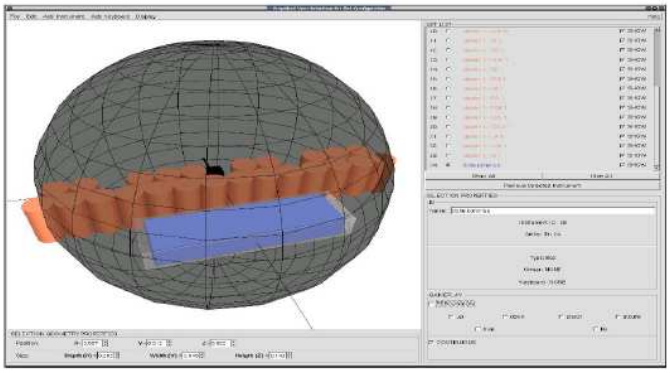

FIG. 2.3 – L'éditeur d'instruments permet de placer les volumes dans l'espace

2.4 Faiblesses de cette version

Depuis le remplacement du système radio bâton par les 'flock of birds' de nombreux rapports font état de problèmes, d'erreurs et qualifient ces capteurs de 'capricieux' ou d'approximatifs. Ces faiblesses ont réellement été identifiées en 2004 par Vincent Goudard [5] qui a effectué de nombreux tests.

Plusieurs phénomènes ont été identifiés :

- Erreurs dues à des perturbations électromagnétiques. La technologie des 'flock of birds' par champ électromagnétique pulsé les rend particulièrement sensibles à l'environnement électromagnétique dans lequel ils se trouvent. La percussion aérienne nécessite la proximité d'écrans, d'enceintes et de structures métalliques qui sont autant de sources potentielles de perturbation.
- Erreurs dues à la connectique. Le matériel vieillit et la liaison filaire des capteurs n'assure pas une parfaite communication, surtout lors des gestes violents de percussion.
- Erreurs dues à la saturation des capteurs. Ce sont les erreurs les plus handicapantes des 'flock of birds'. Lors d'une forte accélération (au moment d'un impact), on constate un blocage des données, ce qui entraîne des erreurs dans les calculs de l'accélération et donc des coups non détectés (Figure 2.4).

Ce problème de données perdues affecte sérieusement le fonctionnement de la percussion aérienne. Trop de coups sont ainsi ratés et le jeu devient pénible pour le percussionniste et trop incertain pour le compositeur. Pour pallier à ce genre

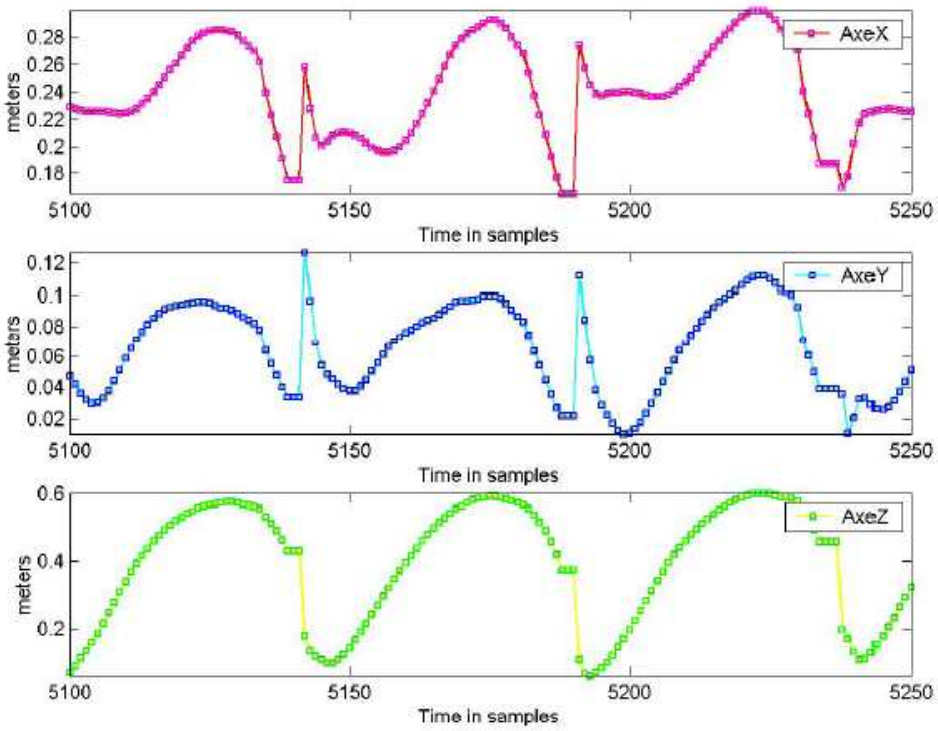

FIG. 2.4 – Courbes montrant la saturation des capteurs

de problème, Vincent Goudard a proposé d'utiliser un algorithme de correction des données. Il a pour ce faire réalisé une implémentation en Mathlab[2] d'un algorithme de prédiction linéaire(LPC) dont les coefficients sont calculés par l'algorithme de Burg[3]. Malheureusement nous nous sommes aperçus trop tard que le portage en C n'avait jamais été terminé et que nous tentions d'utiliser un fichier incomplet.

Une seconde faiblesse de ce système est qu'il n'offre pas toute la liberté espérée. Actuellement la détection se fait en calculant l'accélération en fonction de la coordonnée axiale verticale. Cette méthode ne prend donc en compte que ce type de coups 'vers le bas', or Christophe Havel souhaite pouvoir placer des volumes au dessus de la tête de l'instrumentiste.

[2]Logiciel de traitements mathématiques

[3]Algorithme réputé pour sa faculté à suivre des trajectoires harmoniques et produire des signaux synthétiques proches de signaux naturels

2.5 Améliorer l'instrument

La solution envisagée pour régler ces problèmes est d'enrichir le système en ajoutant des accéléromètres (cf 3.2.1) qui seront chargés de toute la partie détection d'impact. Le repère de l'espace dans lequel ils évoluent n'étant pas absolu (comme c'est le cas pour les 'flocks' avec la base émettrice), mais relatif, la détection va pouvoir être effective dans n'importe quelle position[4].

De plus, étant plus fiables et opérant à une fréquence plus élevée, ils devraient assurer une détection des impacts plus précises. Cela nécessite toutefois le développement de nouveaux algorithmes.

Les 'flock of birds' sont quand à eux relégués à leur fonction première, à savoir donner la position de l'impact dans l'espace. Le problème de la saturation des capteurs n'étant pas réglé, il se peut que certains impacts soient encore mal détectés, puisque détectés hors du module, et donc ignorés.

[4]Tant que le geste adéquat est effectué

Chapitre 3

Présentation du travail à effectuer

3.1 Présentation globale du projet

Voici donc l'instrument tel qu'il est prévu (Figure 3.1). Les nouveaux accéléromètres viennent se greffer au système, pilotés par leur propre librairie qu'il nous faut développer. Ils interviennent directement au niveau de la détection des impacts, qu'il faut aussi modifier.

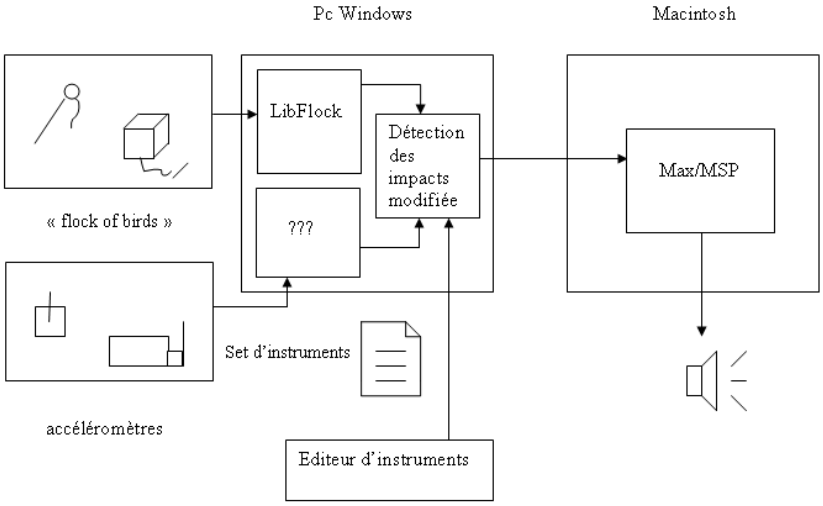

FIG. 3.1 – Schéma représentant l'état du système tel qu'il devrait devenir

3.2 Présentation détaillée

3.2.1 Les accéléromètres

La principale évolution de l'instrument consiste en l'ajout d'accéléromètres que nous allons présenter. Ce sont des accéléromètres Wi-fi G-Link de MicroStrain [3] pouvant mesurer une accélération jusqu'à 10G et ce sur trois directions. Leur fréquence d'échantillonnage est de 617Hz. Ils offrent ainsi une plus grande précision dans les calculs de détection d'impacts, puisque six fois plus rapides que les 'flock of birds'. Un accéléromètre est composé de deux parties (Figure 3.2), une base et un récepteur (que nous appellerons noeud) qui communiquent entre eux par Wi-fi.

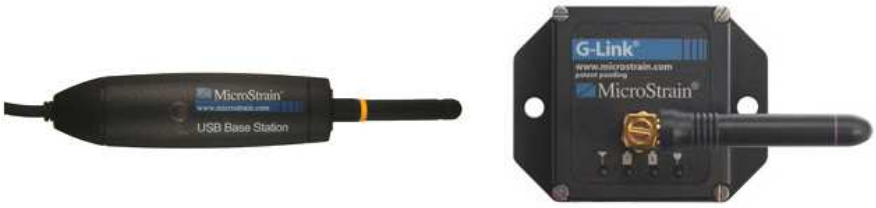

FIG. 3.2 – La base (à gauche) communique avec le noeud(à droite) par Wi-fi

Outre leur modernisme, ils possèdent deux atouts majeurs :

- La communication entre la base et le noeud est totalement transparente pour le développeur. En effet pour exécuter une commande spécifique à un noeud, il suffit de l'envoyer sur la base qui le redirige automatiquement. Il faut toutefois préciser l'adresse du noeud visé. De la même manière, les données émises par les noeuds sont transférées automatiquement par la base à la différence que nous ne pouvons pas savoir de quel noeud proviennent les données (que nous appellerons aussi échantillons).
- Comme nous l'avons dit plus haut, le repère spatial dans lequel opèrent les accéléromètres est relatif au noeud. Le percussionniste peut donc jouer dans n'importe quelle position. Cette nouveauté va permettre de considérablement étendre les possibilités de jeu.

Pour plus de confort de jeu les noeuds, en raison de leurs poids, vont être fixés sur le dos de la main (Figure 3.3).

FIG. 3.3 – Les noeuds seront fixés sur le dos de la main

3.2.2 La Libaccel

A l'image de la Libflock pour les 'flock of birds', il va falloir utiliser une librairie dédiée au pilotage des accéléromètres. Cette librairie n'existant pas, nous allons la développer, ce qui signifie rédiger les fonctions de plus bas niveaux que sont l'ouverture d'un port de communication, l'écriture et la lecture de données, jusqu'aux fonctions de plus haut niveau comme le ping d'un noeud où d'une base. Nous allons donc créer la Libaccel dont le but sera de proposer une interface permettant de configurer et de piloter les accéléromètres. Nous avons choisi de travailler sous Windows en utilisant les fonctions de la bibliothèque windows.h pour façonner la couche de plus bas niveau.

3.2.3 La détection des impacts

L'ajout de ce nouveau matériel a engendré la nécessité de réécrire l'algorithme de détection des coups. Toute la partie théorique de conception des algorithmes doit être majoritairement réalisée par le second stagiaire Joseph Sanson. Mon travail sera donc d'assister Joseph durant ces recherches et surtout d'implémenter et de tester les différents algorithmes créés. Jusqu'à présent nous avons vu que le but principal de l'instrument est de détecter des impacts. Il y a une autre détection que nous devions tenter de résoudre, c'était celle des *types de coups*, mais nous l'examinerons plus tard dans un chapitre spécifique (cf 10).

3.2.4 Cahier des charges

Après avoir pris connaissance des différents travaux à réaliser, nous allons présenter un cahier des charges dans sa version finale. En effet il a beaucoup évolué au fur et à mesure des problèmes rencontrés.

Besoins non fonctionnels

Pour des besoins matériels, le développement va être réalisé sous Windows. Il semble que les accéléromètres ne fonctionnent pas correctement sous Linux ou MacOS, mais nous y reviendrons plus tard (cf 4.6).

Nous avons vu que l'instrument tournait tout d'abord sous Linux, pour finalement être porté sur Windows. Cette dernière mouture est théoriquement la version définitive. Toutefois, afin d'assurer une portabilité future, il est nécessaire d'utiliser une structure par couche lors du développement de la Libaccel. Ainsi, si un nouveau portage devait être effectué il suffirait de modifier les fonctions de plus bas niveau, pour les adapter au nouveau système d'exploitation utilisé. Un langage objet semblait le mieux adapté à supporter ce type d'architecture. Le C++ a été préféré au Java pour des raisons de performance.

L'instrument doit pouvoir être utilisé pour une performance en direct et est donc soumis à une contrainte majeure : la robustesse. Lors d'un concert il ne doit surtout pas 'planter'. Le problème principal, qui a pu être identifié durant la phase de conception, est le risque de perte de données provenant des accéléromètres. La qualité du réseau Wi-fi dépend du niveau de réception qui a tendance à fluctuer au cours du temps, et surtout à se détériorer lorsque l'on s'éloigne de la base. L'acquisition des données doit s'accompagner d'un contrôle de celles-ci ainsi que d'un système de récupération du flux en cas d'erreur. Ainsi les incidences de telles erreurs sur le concert seraient minimes (tout au plus, une seconde où les coups ne seraient pas détectés).

Besoins fonctionnels

Nous pouvons énumérer trois besoins :

- Création de la Libaccel : Il faut développer une interface dédiée aux accéléromètres pour pouvoir les inclure à notre instrument.
- La détection prédictive des coups : Comme nous l'avons vu jusqu'ici, la fonction de l'instrument est de jouer des sons, donc de détecter des impacts. Il est cependant capital de le faire de manière anticipée. Il serait facile de détecter les coups exactement au moment où ils se produisent, mais le temps nécessaire à la génération de son avec Max/MSP nous impose cette prédiction.
- Le temps réel : L'instrument doit pouvoir être utilisé en concert. Il ne doit donc pas y avoir de temps de latence entre le moment réel d'un impact et le son correspondant. Chaque nouvel échantillon doit être traité immédiatement. Il faut déjà tenir compte des 10ms nécessaire à la génération

d’un son. De plus, il ne doit pas y avoir besoin de mémoriser les données afin qu’elles ne soient pas traitées en retard. Nous avons imaginé des tests simples, comme compter le nombre d’échantillons analysés par seconde et vérifier que cela correspond bien à notre fréquence (617Hz).

Chapitre 4

Développement de la bibliothèque de contrôle des accéléromètre : la Libaccel

4.1 Introduction

Le but de la bibliothèque est d'offrir une interface permettant d'utiliser et d'intégrer simplement les accéléromètres, sans se soucier des octets de commande à envoyer ou à attendre. La partie utilisable se doit d'être la plus simple et la plus complète possible afin d'être utilisable non seulement dans le cadre de la percussion aérienne, mais aussi dans tout autre projet faisant appel aux accéléromètres.

4.2 Schéma de présentation (Figure 4.1)

4.3 Le bas niveau

Nous allons voir ici les fonctions de plus bas niveau qui assurent toute la partie communication de la librairie. Un développeur volant intégrer les accéléromètres à son système ne devrait normalement pas les utiliser. En effet, celles-ci sont utilisées par des fonctions de plus haut niveau. Ces fonctions étant directement liées au système, ce sont celles-ci qui devront être modifiées en cas de portage.

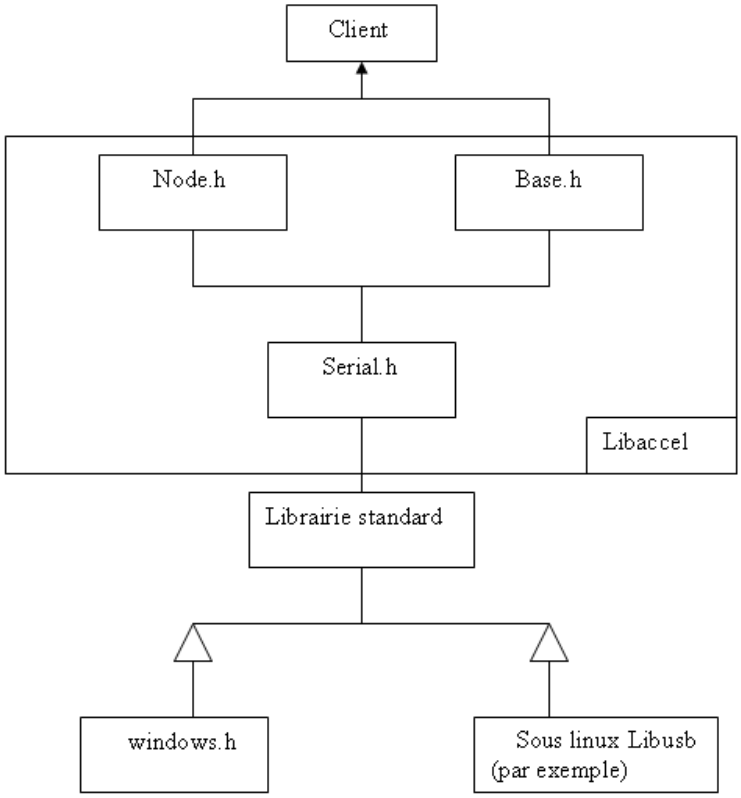

FIG. 4.1 – Schéma présentant la structure de la Libaccel

Serial.h
//Transforme un numéro de port en chaîne de caractères compréhensible par le système. char* make_port_string(unsigned int uiPort) ;
//Ouvre un port de communication avec un périphérique série en fonction du numéro de port ; retourne un identifiant système (HANDLE). HANDLE port_open(/* in */ char* lpszPort) ;
//Termine la communication avec un périphérique série en fonction de son identifiant système (HANDLE). int port_close(/* in */ HANDLE hPort) ;
//Ecrit une commande ucCmd de longueur dwLength sur le port hPort, et retourne le nombre de caractères écrits dwBytes_written. int port_write(/* in */ HANDLE hPort, /* in */ unsigned char *ucCmd, /* in */ DWORD dwLength, /* out */ DWORD & dwBytes_written) ;
//Tente de lire dwLenght caractères sur le port hPort, les place dans le tableau ucResult et retourne le nombre réel de caractères lus dwBytes_read. int port_read(/* in */ HANDLE hPort, /* in/out */ unsigned char *ucResult, /* in */ DWORD dwLength, /* out */ DWORD & dwBytes_read) ;

Nous allons nous servir des méthodes de la bibliothèque windows.h pour réaliser les fonctions de plus bas niveau.

- *port_open* : La première chose à faire est donc d'arriver à identifier notre périphérique sur le système. Sous Windows, cela se révèle assez simple puisqu'il suffit de connaître le numéro du port série sur lequel la base est branchée. Il faut ensuite utiliser la fonction d'ouverture *CreateFile* fournie par la bibliothèque windows.h, qui retourne l'identifiant système de notre périphérique, qui sera utilisé dans toutes les opérations d'écriture ou de lecture.
- *port_read* et *port_write* : Ces deux fonctions sont chargées de l'écriture de chaînes de caractères, correspondant aux commandes avec leurs arguments, et de la lecture des réponses des accéléromètres en utilisant l'identifiant obtenu plus tôt. Ces fonctions sont bloquantes. Elles bloquent le programme tant qu'elle n'ont pas été correctement effectuées. En d'autres termes, la fonction de lecture attend d'avoir lu le nombre de caractères passé en paramètre pour laisser se poursuivre l'exécution.
- *port_close* : Cette fonction est appelée en fin d'utilisation pour quitter proprement.

4.4 Le haut niveau

La seconde étape est d'implémenter toutes les commandes présentes dans le DCP [6]. Celui-ci, particulièrement bien rédigé, permet facilement de savoir quels octets sont à envoyer ainsi que les réponses attendues en cas de succès ou d'échec.

Nous avons préféré diviser les fonctions de haut niveau en deux fichiers, l'un dédié au contrôle de la base, l'autre au contrôle du noeud.

4.4.1 Contrôle de la base

Nous allons voir ici l'ensemble des fonctions dédiées à la base. Ces fonctions utilisent, pour communiquer, les fonctions définies dans les fichiers *serial.** vus plus haut.

Base.h
//Tente de pinguer la base sur le port hPort. Retourne 0 en cas de succès. int bs_ping(/* in */ HANDLE hPort) ;
//Tente d'écrire la valeur usValue à l'adresse mémoire ucLocation de la base identifiée par le port hPort. int bs_eeprom_write(/* in */ HANDLE hPort, /* in */ unsigned char ucLocation, /* in */ unsigned short usValue) ;
//Tente de lire une zone mémoire ucLocation de la base identifiée par le port hPort. En cas de succès, retourne 0 et inscrit la valuer lue dans usResponse. int bs_eeprom_read(/* in */ HANDLE hPort, /* in */ unsigned char ucLocation, /* out */ unsigned short & usResponse) ;

- *ping* : Cette fonction simple permet de savoir si la communication avec la base peut effectivement avoir lieu.
- *eeprom_read* et *eeprom_write* : Ces deux fonctions permettent de configurer la base, en modifiant directement la mémoire EEPROM. Une carte de la mémoire de la base a été fournie avec le DCP. Il est à noter que le seul moyen de configurer la base et les noeuds est de modifier cette mémoire limitée en nombre de réécriture.

4.4.2 Contrôle du noeud

Présentation d'une trame

Les données émises par les noeuds respectent une structure bien précise. Voyons ici un exemple de trame lors d'une acquisition d'accélération sur trois axes (Figure 4.2).

Il faut donc lire les octets huit par huit, vérifier la validité de la trame grâce au bit de 'checksum' et reconstituer les valeurs de chaque axe. Pour ce faire, il faut utiliser la formule très simple :

$$valeur = MSB * 256 + LSB/2$$

Il faut savoir aussi que les accéléromètres peuvent fonctionner en deux modes d'émission de données.

- *Mode discontinu* : Les noeuds ne vont envoyer qu'un certain nombre d'échantillons (donc de trames). Cette valeur doit alors être inscrite directement en mémoire eeprom. L'émission se poursuivra jusqu'à atteindre ce nombre, et se terminera par une succession d'octets 0xAA.

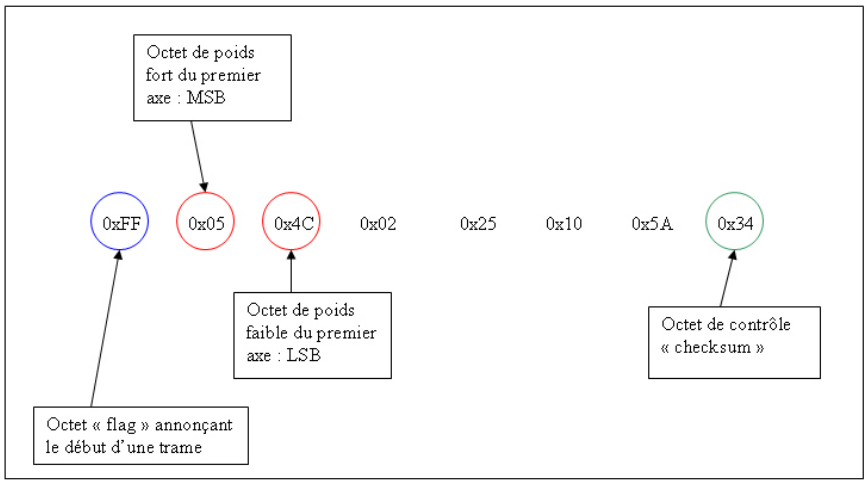

FIG. 4.2 – Exemple de trame

– *Mode continu* : Lorsque l'on fixe à zéro le nombre d'échantillons, les noeuds vont émettre en flux continu. Il n'y a plus de communication possible dans le sens base vers noeud. La seule chose autorisée est la récupération des données émises. Le seul moyen de mettre un terme à ce flux est d'éteindre le noeud. C'est ce mode de fonctionnement que nous allons utiliser, toujours en vu d'une utilisation en concert. Cela pose toutefois un problème en cas de perte d'un octet, mais nous y reviendrons tard (cf 4.5).

Fonctions relatives au noeud

Node.h
//Tente de pinguer le noeud usNode, à travers la base identifiée par le port hPort. Retourne 0 en cas de succès. int nd_ping(/* in */ HANDLE hPort, /* in */ unsigned short usNode) ;
//Tente d'écrire la valeur usValue à l'adresse mémoire ucLocation du noeud usNode à travers la base identifiée par le port hPort. int nd_eeprom_write(/* in */ HANDLE hPort, /* in */ unsigned short usNode, /* in */ unsigned char ucLocation, /* in*/ unsigned short usValue) ;
//Tente de lire une zone mémoire ucLocation du noeud usNode à travers la base identifiée par le port hPort. En cas de succès, retourne 0 et inscrit la valeur lue dans usResponse. int nd_eeprom_read(/* in */ HANDLE hPort, /* in */ unsigned short usNode, /* in */ unsigned short usLocation, /* out */ unsigned short & usResponse) ;
//Tente de placer le noeud usNode en repos, à travers la base identifiée par le port hPort. Retourne 0 en cas de succès. int nd_sleep(/* in */ HANDLE hPort, /* in */ unsigned short usNode) ;
//Initialise une acquisition continue du noeud usNode, à travers la base identifiée par le port hPort. int nd_stream_init(/* in */ HANDLE hPort, /* in */ unsigned short usNode) ;
//Tente de récupérer une trame du flux continue envoyé d'un noeud sur la base identifiée par hPort. Le nombre de canaux à lire ucNum_channels détermine le nombre de caractère à lire. La réponse est inscrite dans le tableau pusResponse. int nd_stream_sweep(/* in */ HANDLE hPort, /* in/out */ unsigned short *pusResponse, /* in */ unsigned char ucNum_channels) ;

- *nd_ping*, *nd_eeprom_read* et *nd_eeprom_write* : Ces fonctions opèrent exactement comme celles relatives aux bases.
- *nd_sleep* : Cette fonction permet de placer un noeud en sommeil afin d'économiser la batterie. Pour le réveiller il suffit de réaliser plusieurs 'ping' jusqu'à ce que le noeud réponde.
- *nd_stream_init* : Lorsque l'on a choisi le mode d'émission (continu ou discontinu), il ne reste plus qu'à envoyer la commande annonçant un début d'acquisition. A ce moment le noeud envoie une série d'octets (entre dix et vingt) sans aucun sens qu'il faut ignorer, jusqu'à l'arriver d'un octet 0xFF qui signale le début de la première trame correcte. Ce sont ces opérations d'initialisation de l'acquisition du flux de données qui sont réalisées ici.

– *nd_tream_sweep* : Cette fonction doit être appelée juste après la précédente. Elle va lire un nombre d'octet dépendant du nombre d'axes à analyser (huit dans le cas de trois axes). Elle va valider la trame grâce au 'checksum' et calcule les valeurs pour chaque axe. Elle doit donc être utilisée de manière redondante, dans une boucle d'acquisition par exemple. Cette fonction intègre aussi un système de récupération de flux en cas d'erreur détectée grâce au 'checksum' (cf 4.5).

4.5 Système de récupération du flux

Nous avons vu précédemment que l'émission des données se fait en flux continu, sans aucun moyen de contrôle (hromis le 'cheksum'). La connexion Wi-fi n'étant pas infaillible il arrive que l'on perde un octets. Or, comme nous l'avons vu plus tôt, les fonctions de lecture sont bloquantes, et attendent de lire huit octets, quels qu'ils soient. Si nous manquons un octet dédié à un axe (Figure 4.2), nous avons une trame incomplète de sept octets. Nous allons donc la compléter en prenant aussi le premier octet de la trame suivante, qui sera considéré comme le 'checksum'. Evidemment la trame sera considérée comme fausse, mais par effet boule de neige, toutes les suivantes le seront.

C'est pour palier à ce genre de problème qu'il a fallu mettre en place un système de récupération du flux. Dès qu'une trame apparaît comme fausse, nous nous re-synchronisons en lisant les octets un à un jusqu'à trouver un 0xFF, synonyme de début de trame. Cette solution a l'avantage d'éviter que l'instrument ne devienne totalement hors service, mais un second problème voit alors le jour.

La détection d'impact se joue à quelques échantillons près. Si nous en manquons deux, comme dans l'exemple précédent, nous pouvons manquer un coup. Il faut savoir aussi que des trames entière disparaissent lorsque l'on éloigne le noeud de la base. C'est ainsi qu'est née l'idée de remplacer ces échantillons manquant, par des données artificielles que nous aurons calculées.

La fréquence d'échantillonnage est de 617Hz. Nous avons donc une trame toute les 1/617 seconde. Nous allons calculer, en parallèle à l'acquisition de données réelles, des données artificielles dans un thread que l'on appellera 'l'usine'. Un 'serveur' est chargé de choisir quelle donnée sera utilisée dans l'algorithme. Nous pensions utiliser l'algorithme de prédiction linéaire (LPC) rencontré plus haut (cf 2.4). Mais lorsque nous sommes passés à la phase des tests, nous nous sommes rendus compte qu'il n'était pas totalement implémenté, et donc inutilisable. Le temps nous faisant défaut, nous avons donc laissé de coté cette aspect de la correction, bien que toute la structure soit en place et prête à accueillir un algorithme de prédiction (cf 6).

4.6 Tentative sous MacOS

Christophe Havel étant habitué à travailler sur Macintosh, nous avons par la suite tenté de porter la Libaccel sous MacOS 10.4.6 . Pour ce faire, j'ai demandé conseil à Patrice Kadionik [4], enseignant à l'ENSEIRB[1], qui m'a conseiller d'utiliser la Libusb !.

Chaque périphérique USB peut être identifié de manière unique grâce à un numéro constructeur ainsi que par le numéro du produit[2], un peu à la manière des cartes réseaux. Malheureusement ces identifiants n'étaient pas fournis et il a donc fallu les retrouver, en développant un petit utilitaire qui liste tous les périphériques USB détectés par le système.

Cette étape franchie avec succès et nos identifiants en main nous pouvons passer à la phase échange de données. Après les étapes d'ouverture et de configuration du port de communication, nous tentons d'envoyer l'octet 0x02, correspondant à la commande de ping. La base ainsi 'pinguée' aurait dû répondre en renvoyant l'octet 0x02, chose qu'elle ne fera jamais.

En effet, même après de nombreux essais d'après des solutions proposées sur des sites ou forums internets, la base reste désespérément muette. Le dernier recours a donc été de contacter la société fabricante américaine MicroStrain en leur demandant pourquoi la base ne répondait pas. La réponse a été plus qu'inattendue puisque qu'ils n'en savaient rien et qu'ils ne pensaient pas que les accéléromètres puissent fonctionner un jour sous MacOS. Dans le même temps nous avons contacté le distributeur français au sujet d'un noeud défectueux qu'il a fallu leur envoyer pour réparation. Le technicien se révélant vraiment compétent et collaborateur, nous en avons profité pour discuter de ce problème. La réponse a été presque la même à ceci près qu'il nous a conseillé de travailler sous Windows car il est certain que les accéléromètres ne fonctionneront jamais sous MacOS.

Après comparaison avec la version utilisée sous Windows, il semble que le problème vienne d'un petit chip utilisé dans les bases qui convertit le flux de donnée USB en flux série. Ainsi, malgré la connectique USB, sous Windows ils apparaissent comme des périphériques séries (Figure 4.3). Il semblerait donc que le chip chargé de cette conversion ne soit pas pris en charge par les systèmes MacOS et Linux.

[1] Ecole Nationale Supérieur d'Electronique et Informatique de Bordeaux

[2] Connus sous les noms de ID Vendor et ID Product

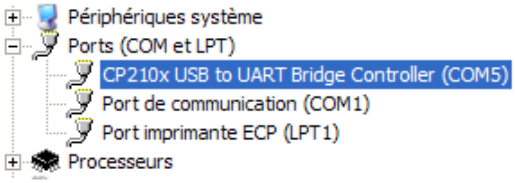

FIG. 4.3 – Les accéléromètres apparaissent comme des périphériques séries

Chapitre 5

Intégration des ‘flock of birds’

5.1 Introduction

Les ‘flock of birds’ fonctionnaient grâce à la Libflock qui n’existait que sous Linux. Nous devions donc porter cette librairie sous Windows. Le temps nous faisant défaut, nous avons préféré utiliser une ‘dll’ trouvée sur le site du constructeur [1]. Nous avons développé une interface permettant l’utilisation des ‘flock of birds’ sous Windows, et ce pour n’importe quel programme. Nous avons ensuite modelé cette bibliothèque pour l’adapter à notre problème.

5.2 Schéma de présentation (Figure 5.1)

5.3 Le haut niveau

Flock.h
//Tableaux contenant les coordonnées des capteurs. Protégés par un système de sémaphore, pour évter un accès concurrent en lecture et en écriture. Private : double coord_1[6] ; double coord_2[6] ;

//Tente d’initialiser les ‘flock of birds’. Retourne 0 en cas de succès. int flock_init() ;
//Lance une acquisition continue des positions des ‘flock of birds’.
void flock_stream() ;

//Permet de récupérer les coordonnées des capteurs. double * flock_get_coord_1() ; double * flock_get_coord_2() ;

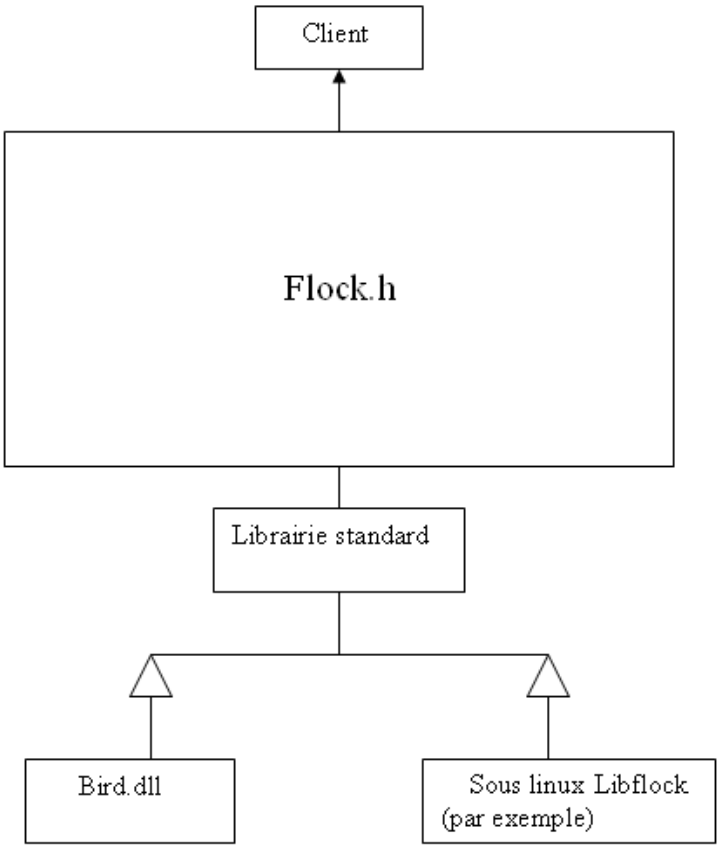

FIG. 5.1 – Schéma présentant la structure du fichier Flock.h

Nous avons choisi un tel mode de fonctionnement pour une raison simple. Les ‘flock of birds’ et les accéléromètres n’opèrent pas à la même fréquence d’échantillonnage (100Hz contre 617Hz). Ainsi, comme il est impossible de les synchroniser, lorsque nous recevons un échantillon provenant des accéléromètres, nous allons chercher la dernière position des ‘flock of birds’ connue. Nous pensions corriger, mais aussi extrapoler, le coordonnées spatiales grâce à l’algorithme de prédiction linéaire (LPC) déjà rencontré afin d’avoir la position exacte des baguettes pour chaque échantillon provenant des accéléromètres. Mais l’algorithme n’étant pas implémenté, nous avons repoussé cette étape.

Chapitre 6

Architecture logicielle

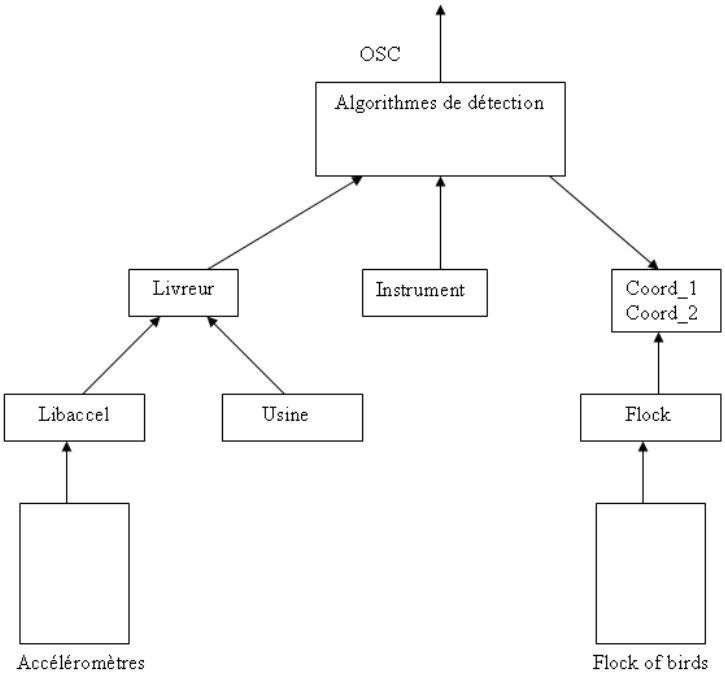

FIG. 6.1 – Vue d'ensemble de l'architecture logicielle de l'instrument

Nous allons résumer le fonctionnement de l'instrument. Les accéléromètres contrôlés grâce à la Libaccel émettent en continu. En parallèle, 'l'usine' calcule des données théoriques. Un livreur est chargé de choisir la donnée à exploiter. Lors qu'une donnée réelle est fausse, alors le livreur choisie celle calculée dans l'usine. La donnée est ensuite traitée par les algorithmes de détection.

Lorsqu'un impact est détecté, nous allons chercher s'il l'a été dans un module. Au lancement de l'instrument nous allons chargé un fichier préalablement généré par l'éditeur d'instrument. Les coordonnées des 'flock of birds' sont récupérées grâce à la librairie dédié. Elles sont placées dans des tableaux. Ceux-ci ne seront consultés que lorsqu'il y aura besoin de déterminer l'emplacement d'un impact.

Une fois un impact validé, un message est transmis au Macintosh grâce au protocole OSC.

Chapitre 7

Système de détection des impacts

7.1 Introduction

Le travail de recherche concernant la détection des impacts a été effectué en collaboration avec Joseph Sanson. Il a réalisé une étude statique des données issues des accéléromètres durant différentes séquences de jeu. Nous en avons déduit des algorithmes prédictifs de détection que nous avons ensuite modifiés en fonction d'impératifs matériels et de tests en conditions réelles, notamment avec le concours d'un percussionniste professionnel. Après quelques tests nous sommes très vite arrivés à la constatation qu'il fallait diviser la détection des impacts en plusieurs algorithmes. Un algorithme gère la détection de la majorité des coups, et un autre le cas particulier où le percussionniste désire jouer un *frisé* à l'intérieur d'un volume, en effectuant un enchaînement rapide de coups vers le bas et le haut.

Nous utilisons donc deux algorithmes que nous identifierons comme un **jeu normal** ou un **jeu frisé**.

7.2 Pourquoi deux algorithmes ?

Ce choix s'est imposé à cause d'une constante indispensable à notre algorithme, le temps minimum entre deux coups. Cette valeur sert à éliminer les retours de baguette qui sont détectés comme des coups. Elle est fixée à un maximum de huit coups vers le bas par seconde. Dans le cas d'un *frisé*, le percussionniste peut effectuer jusqu'à seize coups par seconde (huit vers le haut, huit vers le bas), il aurait donc fallu diviser cette constante par deux, et ainsi détecter des coups parasites en cas de jeu normal.

Il faut maintenant choisir quel algorithme utiliser, et quand en changer. En cas

de jeu normal, si lorsque le temps entre deux coups devient suffisamment faible alors, l'algorithme de détection des *frisés* intervient. Et inversement, en cas de jeu en *frisé*, si le temps entre deux coups devient suffisamment long, le système bascule en mode de jeu normal.

7.3 Choix de l'axe significatif

En première intention nous avons travaillé avec les valeurs d'accélération sur trois axes, en calculant le vecteur accélération. Les résultats ne se sont pas révélés concluants. En effet l'axe qui correspond à la profondeur (axe Y, voir Figure 7.1), n'est pas utilisé dans le jeu d'un batteur et parasite les données significatives lors du calcul du vecteur d'accélération.

Le choix entre les axes restants correspond plus au jeu du percussionniste. S'il joue en mode *timbales*, un peu à la manière des batteurs rock, le dos de la main perpendiculaire au sol, alors il faut travailler avec l'axe X (Figure 7.1). S'il joue en *percussion classique*, la dos de la main tourné vers le ciel, alors il est préférable d'utiliser l'axe Z (Figure 7.1). Le percussionniste attitré à la percussion aérienne utilisant plus ce dernier mode, c'est tout naturellement celui que nous avons choisi.

FIG. 7.1 – Exemple de trame

Il est nécessaire de voir avec le percussionniste quel type de jeu lui convient le plus. Ce choix est capital pour garantir un bon fonctionnement de l'algorithme (comme nous pouvons le voir ici Figure 7.2). Nous avons de plus constaté que la manière de jouer d'un percussionniste varie d'une main à l'autre. Il est donc nécessaire de trouver l'axe significatif pour chaque main de chaque instrumentiste. L'algorithme étant paramétrable, il faut donc réaliser une calibration de l'instrument afin de parfaitement l'adapter à son utilisateur.

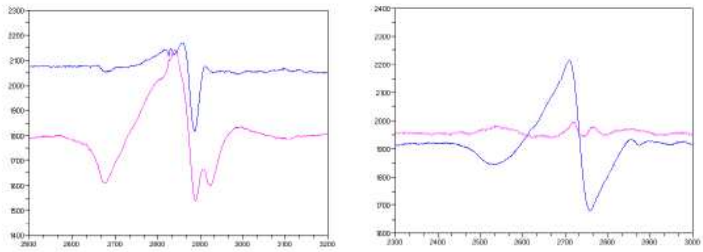

FIG. 7.2 – A gauche un jeu *timbales* et à droite un jeu *percussion classique*. (en bleu l'axe Z, en rose l'axe X)

7.4 Détection des coups 'vers le bas' (jeu normal)

Les algorithmes de détection antérieurs utilisaient des seuils d'accélération fixes afin de déterminer l'entrée en phase de préparation de coup et ainsi en déduire le moment de l'impact. Nous avons trouvé cette solution beaucoup trop rigide. Nous avons donc préféré travailler sur la dérivée de l'accélération, ce qui se révèle beaucoup plus souple.

La première chose à savoir est que nous ne travaillons pas directement sur les données brutes. Nous faisons une moyenne sur entre notre échantillon et les deux précédents, de manière à atténuer le bruit d'acquisition(Figure 7.3).

L'algorithme de détection fonctionne comme ceci. A chaque nouvel échantillon (moyenné) nous allons calculer la dérivée de l'accélération, par différence avec la précédente. Si elle dépasse un certain seuil que nous nommerons *accel_1*, nous entrons alors dans un premier traitement qui va déterminer si nous sommes bien en phase d'accélération, et nous incrémentons un compteur. Si ce compteur est incrémenté de manière **continue** [1] jusqu'à atteindre un seuil *accel_2* alors nous considérons que la phase d'accélération a bien eu lieu.

Nous allons ensuite attendre la phase de décélération. Ainsi si la dérivée passe sous un seuil *accel_3*, nous entrons en phase de décélération. A la manière de l'accélération, nous incrémentons un compteur de manière **continue**, et si celui-ci atteint un seuil *accel_4*, alors nous entrons dans le cas où un impact est probable. Il faut encore vérifier que celui-ci n'ait pas lieu sitôt après un autre (élimination des retours de baguette) et qu'il ait atteint une certaine intensité (élimination des

[1] Si ce n'est pas le cas, alors il n'y a pas de préparation de coup et le système de détection est réinitialisé

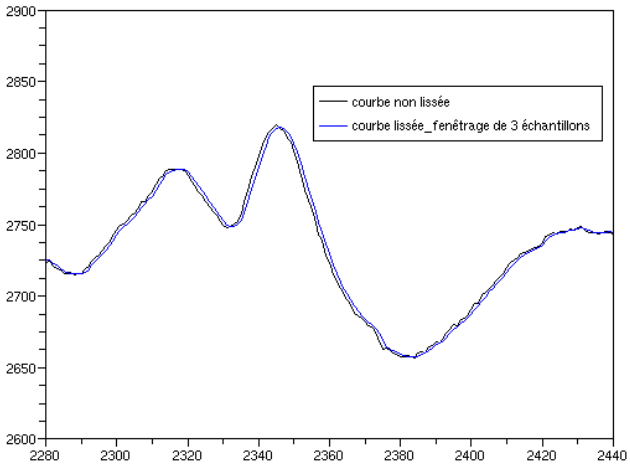

FIG. 7.3 – Une moyenne sur trois échantillons permet d'atténuer le bruit

coups parasites dus au bruit de déplacement).

Nous allons maintenant essayer de comprendre à quoi correspondent ces seuils :

- *accel_1* : Ce seuil détermine, en quelque sorte, l'accélération minimale à atteindre afin que soit possible la détection d'un coup. Autrement dit, il fixe la vitesse d'élan minimale à donner au mouvement. Ce seuil peut être vu comme la 'dureté' de l'instrument. S'il est trop bas, nous allons détecter trop de coups parasites et de retours de baguette, et s'il est trop élevé nous n'allons rien détecter du tout.
- *accel_2* : Ce seuil détermine la durée minimale de la phase d'accélération. Il ne devrait cependant pas être modifié, car il a été fixé durant les études de stagiaires précédents avec les 'flock of birds', pour être adapté aux accéléromètres.
- *accel_3* : Ce seuil fonctionne un peu à la manière de *accel_1*. Les modifications qu'éventuellement on lui apporte sont en revanche moins significatives.
- *accel_4* : Ce dernier seuil fixe la durée minimale de décélération. Lui aussi a été déduit d'études antérieures. C'est ce paramètre qui permet de fixer le niveau d'anticipation du coup.

7.5 Détection du jeu frisé

Dans le cas très spécifique d'un frisé, où l'on doit détecter les coups vers le bas et le haut, nous pouvons observer des courbes d'accélération comme celle ci-dessous (Figure 7.4).

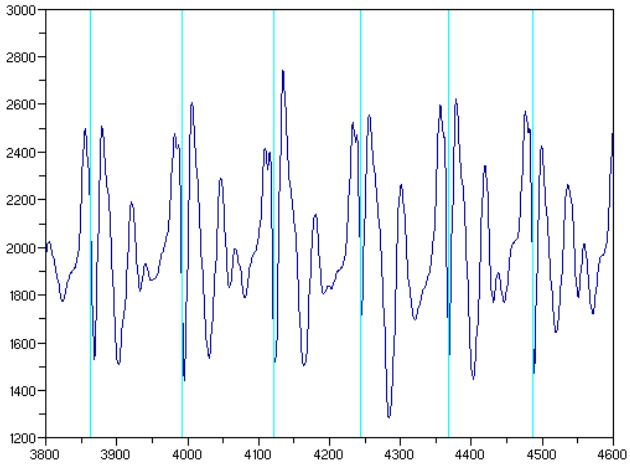

FIG. 7.4 – Détection des coups vers le bas dans une séquence *frisé*

Nous pouvons voir que dans ce genre de jeu rapide, les pic d'accélérations atteignent à peu près tous la même intensité. Nous avons commencé par mettre en place un système qui vérifie, dans un jeu rapide, qu'un pic détecté est bien du même ordre de grandeur que ceux détectés précédemment. Cela permet, entre autres, d'éliminer des pics parasites générés par ce mode de jeu.

Afin de réaliser la détection des coups vers le bas et le haut, nous nous sommes basés sur la constatation que l'espacement entre chaque coups était le même durant une séquence de jeu. La première détection qui tentait de repérer indépendamment les impacts vers le bas et vers le haut s'étant révélée un échec, nous avons mis en place un système bien moins scientifique, mais plus efficace. Dès qu'un tel type de jeu est détecté, nous calculons la distance entre deux coups vers le bas et nous rajoutons artificiellement les coups vers le haut à la moitié de la valeur obtenue (Figure 7.5. Bien entendu, le premier impact vers le haut ne sera pas détecté.

Bien que peu conventionnelle, c'est cette solution que nous avons soumise aux

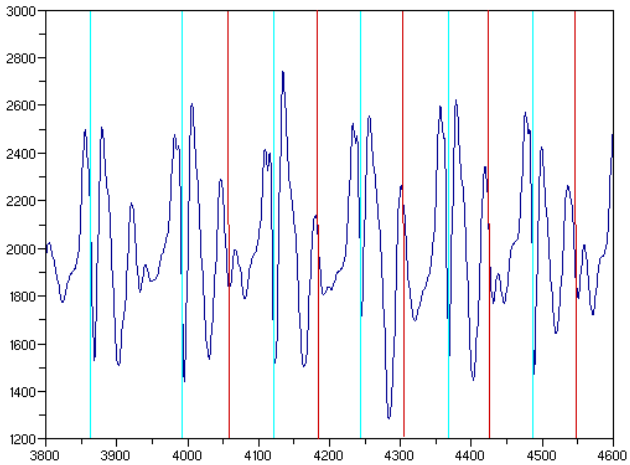

FIG. 7.5 – Rajout artificiel des coups vers le haut

différents testeurs, sans leur expliquer que la moitié des coups était artificielle. Nous verrons plus loin leurs commentaires (cf 9.2).

Chapitre 8

Rendre l'instrument fonctionnel

8.1 Schéma récapitulatif

En nous référant au schéma final (Figure 8.1) de l'instrument, nous voyons qu'il manque encore deux unités de traitement pour rendre le système utilisable :

- L'utilisation des sets d'instruments
- La liaison avec le Macintosh

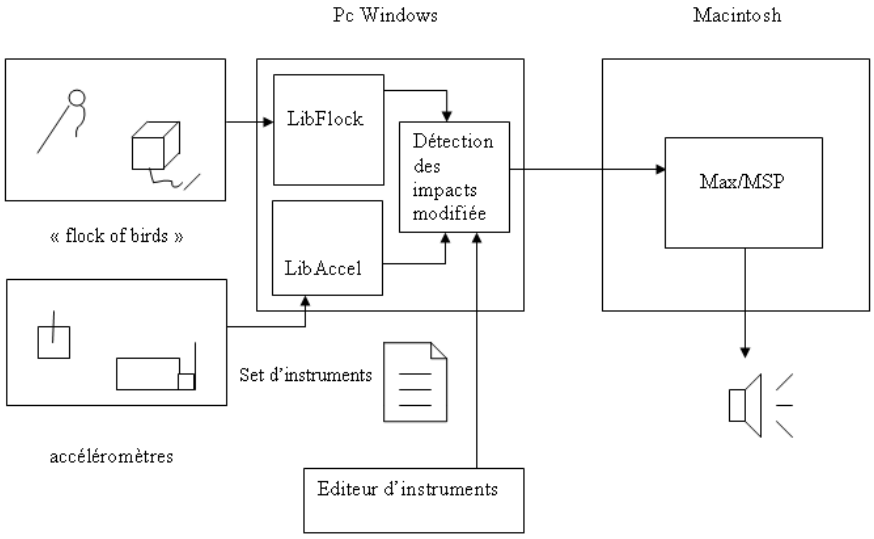

FIG. 8.1 – Schéma du système final

8.2 Intégration des sets d'instruments

La détection des coups ainsi que la récupération des coordonnées étant opérationnelles, il faut maintenant savoir si un module a été frappé. Nous allons donc mettre en place un système qui va récupérer et analyser les *sets d'instruments* créés par l'éditeur vu précédemment. Nous allons donc générer les fichiers *instrument.** :

- *load_set* : Cette fonction est chargée de d'analyser le fichier créé par l'éditeur et de charger en mémoire les coordonnées, tailles et types de chaque module.
- *retreive_instrument* : Cette fonction permet de déterminer l'instrument frappé, et si l'impact a eu lieu dans un module, en fonction des coordonnées de l'impact. Une fois l'instrument identifié, en fonction du type de l'instrument, nous sommes en mesure de remplir le message qui sera envoyé au Macintosh et de savoir quels messages suivront. En effet si un coup est porté dans un instrument 'toggle'(continu) il va falloir renvoyer continuellement la position de la baguette, qui sera utilisée pour moduler le son produit.

8.3 Communication avec le Macintosh

La dernière étape avant de rendre notre instrument parfaitement opérationnel est d'implémenter la communication avec le Macintosh. Tout comme dans la version précédente nous allons utiliser le protocole OSC. Cette partie est encore en phase de développement à l'heure où sont écrites ces lignes. Il reste principalement à décider avec Christophe Havel, quelles données il désire récupérer.

Chapitre 9

Résultats des tests

9.1 Résultats graphiques

Les premiers tests que nous avons réalisés sont purement statistiques. Nous avons enregistré dans un fichier les échantillons reçus pendant une courte séquence de jeu, dans laquelle nous enchaînons les *downs*. Nous avons ensuite appliqué notre algorithme de détection à ces données et marqué le moment où nous détectons un impact. Nous avons ensuite visualisé ces données sous forme de graphique (Figure 9.1).

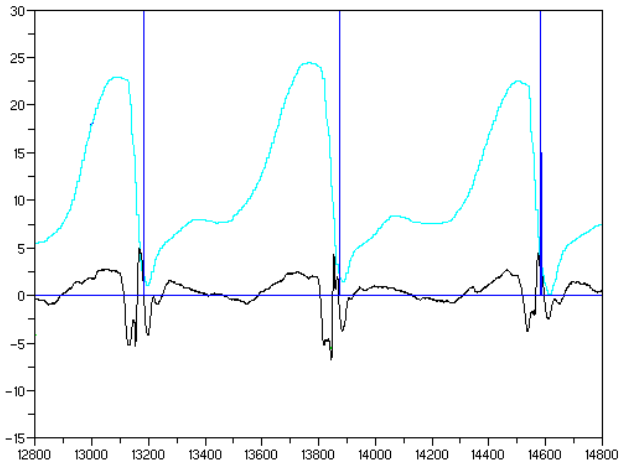

FIG. 9.1 – Courbes de position et d'accélération

Sur le graphique précédent nous pouvons voir en bleu clair l'évolution de la position suivant l'axe vertical, en noir la courbe d'accélération suivant le même axe et en bleu foncé le moment où nous détectons un impact. Nous pouvons donc bien constater que l'impact arrive juste avant la position verticale minimale, donc avant que le coup ne soit réellement effectué.

Cette période d'anticipation, demandée par le compositeur, sera donc utilisée pour générer un son correspondant aux paramètres extraits de l'accélération et des positions, à savoir l'accélération maximale obtenue (le sommet du pic), et le volume frappé.

9.2 Résultats en conditions réelles

L'avant dernière étape de test devait être réalisée par des instrumentistes professionnels. Nous avons mis en place un petit système qui joue un gingle au moment de la détection d'un impact. Le retour physique étant absent, et la connexion avec le Max/MSP n'étant alors pas encore opérationnelle, c'est le seul moyen que nous avions pour faire tester l'instrument. Nous avons donc fait appel à deux percussionnistes, dont Clément Fauconnet, instrumentiste attitré à la percussion aérienne.

La première constatation que nous avons pu faire est que le logiciel assure la robustesse demandée. Une fois passée la fastidieuse mais inévitable phase d'initialisation du matériel, le logiciel fonctionne sans problème. Il n'est sujet à aucune erreur quelles que soient les conditions de jeu. Le système de récupération de flux fonctionne donc correctement. Il faut toutefois noter que la liaison entre la base et le noeud stoppe lorsque l'on s'éloigne trop, et donc l'instrument plante, mais une fois encore ce problème n'est pas de notre fait.

La seconde remarque est venue des percussionnistes. Clément Fauconnet nous a donné son avis par rapport à la version précédente de l'instrument. Il nous a tout d'abord assuré que l'instrument est bien plus agréable à utiliser. Les impacts sont mieux détectés, ils arrivent au moment du coup alors qu'ils avaient tendance à arriver en retard. De plus il ne cherche plus à déclencher le coup en forçant anormalement le geste. Le nouveau système permet donc plus de finesse de jeu.

Il y a cependant un bémol, des coups parasites ont été détectés. Certains retours de baguettes, que nous avons tenter d'éliminer en travaillant sur les différents seuils, sont quand même apparus comme des impacts. Il faut relativiser en précisant que cela arrive moins de 10% du temps. Nous avons donc re-calibré

l'instrument au cours des tests et nous sommes arrivés à une conclusion surprenante.

Un percussionniste ne joue pas de la même manière avec chacune de ses mains. Il n'a pas le même geste. Il ne tient pas la baguette de la même manière, n'a pas les mains orientées exactement de la même façon. Il en résulte que l'algorithme doit être adapté à chacune des mains, de chaque percussionniste. Ainsi après une phase de calibrage, la détection des impacts c'est révélée très satisfaisante et les rares retours de baguette détectés sont ignorés car en dehors des modules.

Le second percussionniste a, quand lui, plus travaillé sur l'algorithme chargé de détecter les *frisés*. Il nous a confirmé que les sons émis correspondaient aux coups joués. Nous lui avons demandé de jouer le plus rapidement possible, et même là l'algorithme s'est révélé efficace.

Chapitre 10

Note sur la détection des types de coups

10.1 Introduction

La simple détection des impacts ne permet pas de rendre toute la finesse gestuelle propre à un percussionniste professionnel. Une fois cette détection mise en place, nous devions trouver un moyen d'enrichir les algorithmes afin d'identifier, toujours de manière prédictive, quel *type de coup* a été effectué. Le but est une fois de plus d'enrichir l'instrument en couplant les types de coups aux types d'instruments vus plus haut, pour offrir le plus large choix possible dans la composition d'une pièce. La première chose à faire est donc de définir ce que nous entendons par *type de coup* et *enchaînement*.

10.2 Définition des *types de coups* et des *enchaînements*

Les types de coups sont ceux définis par le percussionniste François Dupin, pour l'enseignement de la percussion classique, à savoir :

- *down* : le mouvement commence assez loin de l'instrument, le frappe et revient légèrement.
- *up* : le mouvement commence près de l'instrument, le frappe et remonte assez haut. Ce coup apparaît principalement dans les *flas*.
- *piston* : le mouvement est similaire à un va-et-vient.
- *étouffé* : le mouvement est arrêté brusquement après l'impact. Sur une batterie classique la baguette doit rester collée à la peau après l'impact afin d'atténuer la résonance.

Ces différents types de coups ont été modélisés lors des stages précédents, d'après leur 'allure théorique'(Figure 10.1). Il en a résulté toute une série de seuils (symbolisés par les lignes en pointillés) eux aussi totalement théoriques.

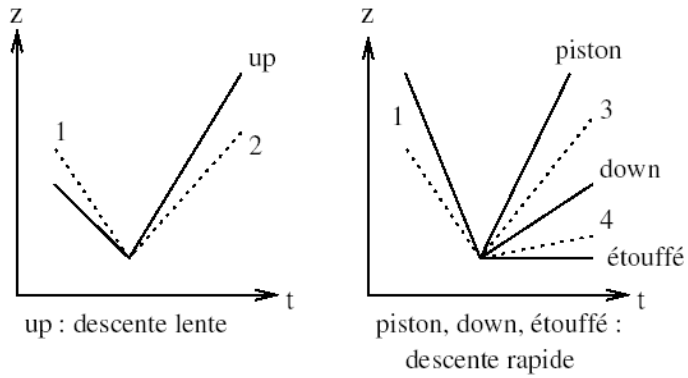

FIG. 10.1 – Les différents types de coups selon la coordonnée verticale au fil du temps

Une fois ces types de coups clairement identifiés il est possible d'aller un peu plus loin en détectant un *enchaînement* spécial, tel que :

- Le *fla* : qui consiste en une succession rapide et désynchronisée de *up* et de *down*.
- Le *frisé* ou *roulement* : qui consiste en une succession rapide de *pistons*. Pour visualiser cet enchaînement il faut imaginer un crayon que l'on positionnerait entre notre index et notre majeur et qui frapperait successivement et rapidement chaque doigt.

10.3 Problème de détection en intention

Nous n'avons pas eu beaucoup de temps pour mener ces recherches, mais il nous est très rapidement apparu que cette détection prédictive semblait très difficile, voir impossible compte tenu des données que nous pouvons exploiter.

Si nous nous référons aux courbes théoriques des coordonnées pour chaque *type de coup* (Figure 10.1), nous nous apercevons qu'il est impossible de différencier chaque *type de coups* avant d'avoir pu l'analyser dans son intégralité. En d'autres termes, la phase avant impact (courbe descendante) est la même pour chaque type.

Nous sommes donc passés à une étude des courbes d'accélération. Là encore nous nous sommes rendus compte qu'il était impossible de différencier certains *types de coups*. Si nous prenons l'exemple d'un down et d'un piston (Figure 10.2), nous pouvons voir que la partie avant le coup est sensiblement la même.

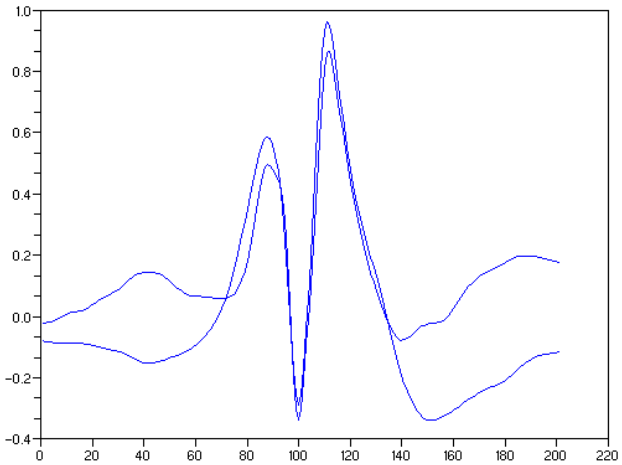

FIG. 10.2 – Courbes d'accélérations idéales d'un *down* et d'un *piston*

Nous en sommes donc arrivés à la conclusion qu'il était impossible d'arriver à identifier les *types de coups*, et par conséquent la plupart des *enchaînements* avec le peu d'information à notre disposition.

10.4 Solutions envisageables

Conserver la prédiction

Nous pensons qu'il serait possible de conserver un système prédictif de détection des coups, en augmentant tout simplement le nombre et le type de capteurs. En effet bon nombre de paramètres ne sont pas pris en compte tels que l'angle entre le poignet et le bras durant la préparation d'un coup ou encore la tension musculaire de l'avant bras, qui nous semblent très significatifs. Cette solution apparaît comme la plus intéressante artistiquement mais demanderait tout d'abord des dépenses supplémentaires, mais aussi un gros travail de recherche.

Mouvement déclencheur

Le percussionniste pourrait annoncer un *type de coup* ou un *enchaînement* en effectuant au préalable un mouvement codifié. Il pourrait par exemple dessiner un cercle en face de lui, qui serait alors interprétée comme le déclenchement d'un *fla*. C'est idée ne semble pas réellement envisageable puisqu'elle casse le jeu 'naturel' du percussionniste. Elle pourrait cependant être tournée à l'avantage de la

représentation en associant visuel et sonorité.

Module déclencheur

Cette solution s'apparente à la précédente puisqu'au lieu de réaliser un geste précis, l'instrumentiste devrait frapper un module de contrôle (par exemple situé au dessus de lui) qui servirait à annoncer un coup particulier. Les inconvénients sont sensiblement les mêmes que précédemment.

Analyse 'après coup'

L'idée serait ici d'analyser les coups, quitte à détecter en retard quel type vient d'être joué. Ainsi ce résultat pourrait être exploité par un automate probabiliste, issu d'une étude statistique, afin d'anticiper les prochains types de coups. En l'état actuel cette solution semble être la plus envisageable.

Chapitre 11

Extensions

11.1 Finaliser

L'instrument n'est pas encore fini. Même s'il est opérationnel, il reste encore à l'état de prototype. Plusieurs travaux sont encore nécessaires :

- Interface graphique : Il faut doter le système d'une interface graphique afin de pouvoir régler les différents paramètres. Dans notre version les numéros de port ainsi que les valeurs des seuils doivent être fixés avant chaque compilation. Il n'est pas envisageable de laisser cette manipulation à la charge du compositeur. Des travaux devraient être en cours au moment ou vous lisez ces lignes.
- Sélection d'algorithme : Le système permettant de changer d'algorithme n'est pas encore mis en place. En l'état, il faut choisir quel algorithme sera utilisé, sachant qu'il le sera pour le reste du jeu. Nous avons déjà conçu le système permettant de basculer (comparaison de la proximité des coups) mais nous n'avons pas encore pu l'implémenter.
- Bien définir OSC : Pour l'instant nous ne connaissons pas encore exactement les données qui seront envoyées. Il faudra sûrement adapter le module de communication.

11.2 Aller plus loin

Même si l'instrument est quasiment opérationnel, il pourrait encore être amélioré.

- Portage sous Macintosh : En effet, comme nous l'avons déjà mentionné dans ce rapport, Christophe Havel travaille sous Macintosh. Il serait donc préférable pour lui que l'instrument fonctionne sur une plateforme qu'il maîtrise, d'autant plus que le constructeur du chipset posant problème vient

de mettre en ligne, ce mois-ci, un driver MacOS et Linux. L'adaptation MacOS semble donc envisageable.

– La détection des types de coups : Il n'y a pas grand chose à ajouter sur ce qui a été dit plus haut, si ce n'est que ces travaux dépendent essentiellement de la décision prise par Myriam Desainte-Catherine et Christophe Havel.

Chapitre 12

Bilan

Au final cette version de la percussion aérienne remplit ses promesses. La détection est plus précise, et le jeu se révèle plus agréable pour l'instrumentiste. L'idée d'ajouter des accéléromètres s'est avérée particulièrement judicieuse, d'autant plus que toutes les possibilités qu'ils offrent n'ont pas encore été exploitées. Nous pensons, bien entendu, à la détection des types de coups.

Nous avons vu que les impacts peuvent désormais être détectés dans n'importe quelle direction, ce qui offre de nouvelles possibilités de création artistique pour le compositeur, qui devrait bientôt être explorée. En effet l'instrument devrait même être utilisé dans le cadre d'un concert Métamorphose en décembre prochain.

Bibliographie

[1] Ascension technology. www.ascension-tech.com.

[2] Groupe de musique electronique de marseille. www.gmem.org.

[3] Microstrain inc. www.microstrain.com.

[4] Page de patrice kadionik. www.enseirb.fr/kadionik/.

[5] Vincent Goudart. *Percussion aérienne*. SCRIME, 2004.

[6] Microstrain Inc. *Agile-Link 900MHzData Communications Protocol*. Microstrain Inc., 2005.