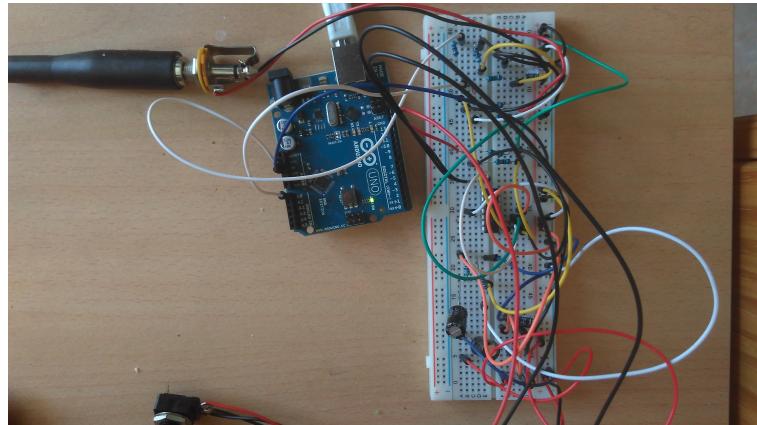


ENSIMAG

RAPPORT DE FABLAB 2A

Reconnaissance des notes jouées à la guitare



Matthieu BAUMANN

Responsable de projet :
M. MAISONNASSE

Janvier 2015 — Mai 2015

Table des matières

1 Présentation du projet	2
2 Etude du schéma d'amplification	2
2.1 Explication détaillée	3
2.2 Simulations LTSpice et résultats	4
3 Programme Arduino UNO	6
3.1 Echantillonnage et buffer de données	6
3.2 Méthode de reconnaissance de la fréquence d'un signal avec bruit	6
3.2.1 Principe d'autocorrélation du signal	7
3.2.2 Calcul de fréquence	7
4 Communication Serial avec un PC Linux	8
4.1 Interfaçage MIDI avec les logiciels	9
4.2 Ttymidi, QjackCtl, Qsynth et Rosegarden	9
5 Conclusion	10
5.1 Résultats	10
5.2 Difficultés et idées d'amélioration	11

1 Présentation du projet

Etant musicien et guitariste depuis quelques années, j'ai souhaité pouvoir convertir le signal audio créé par ma guitare en signaux MIDI compréhensible par un PC/Mac. Les logiciels de MAO (Musique Assistée par Ordinateur) tels que GarageBand, Ableton Live 9 ou encore Rosegarden sur Linux importent des banques de sons MIDI qui sont aujourd'hui de bonne facture (sons de percussions, cuivres, cordes, piano...). Reconnaître la fréquence des notes jouées à la guitare pourrait dès lors présenter un grand intérêt pour les musiciens dans le sens qu'il serait possible de superposer aux notes jouées ces sons MIDI. On se sert alors de sa guitare comme d'un contrôleur MIDI de la même façon que l'on se servirait d'un clavier maître MIDI.

2 Etude du schéma d'amplification

Le μC Arduino exige des tensions d'entrées comprises entre 0-5 V. Il m'a donc fallu concevoir un circuit permettant de :

- amplifier le signal recueilli par les micro de ma guitare. Le taux d'amplification étant inconnue car dépendant du type de micro ainsi que de la guitare utilisée, il faudra permettre à l'utilisateur de régler le gain d'amplification du signal de sa guitare.
- ajouter un offset au signal reçu égal à 2.5V de sorte que le signal évolue autour de cette tension.

Le schéma a été conçu et simulé à l'aide du logiciel gratuit LTSpice IV.

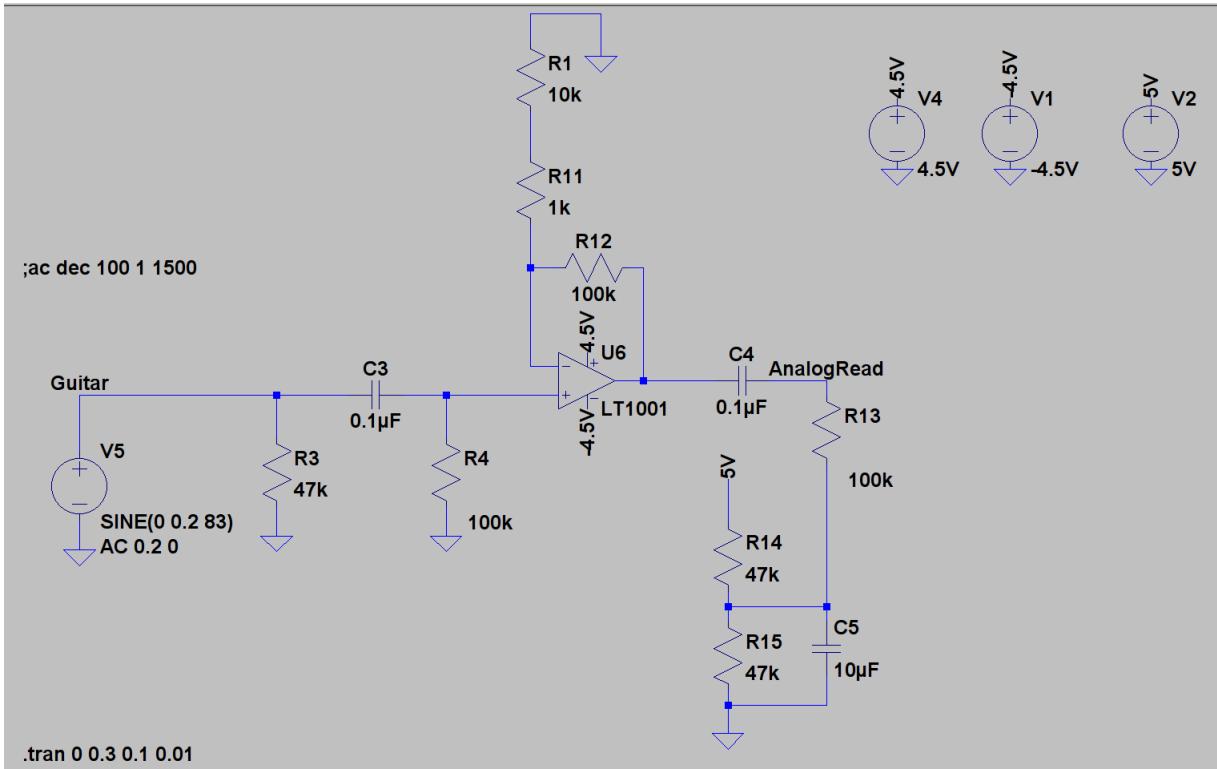


FIGURE 1: Schéma d'amplification de l'adaptateur MIDI

2.1 Explication détaillée

La guitare (représentée par un générateur de courant sinusoïdale de fréquence f) est branchée via une connectique jack 6.35mm mâle/femelle à gauche du schéma. La sortie du circuit à gauche est directement connectée à l'entrée A0 d'un μ C Arduino UNO.

Le signal passe par divers étages qui sont :

- Un filtre passe haut formée par l'association du condensateur C3 et de la résistance R4 et permettant de coupler le signal de la guitare c'est-à-dire de supprimer la composante DC du signal de sorte que ce dernier soit bien centré autour du potentiel 0V. Ce filtre passe-haut ne doit en aucun cas couper les fréquences qui nous intéressent (les fréquences correspondantes aux notes qu'une guitare peut produire en accordage traditionnel EADGBE) ! On estime à ce propos qu'une guitare produit des notes de fréquences comprises entre 50 et 1500Hz. On donne, pour s'en assurer, la correspondance notes-frequencies à l'aide du tableau suivant :

Corde	Note	Notation anglophone	Fréquence
1(fine)	Mi	E	329.6Hz
2	Si	B	246.9Hz
3	Sol	G	196.0Hz
4	Ré	D	146.8Hz
5	La	A	110.0Hz
6(grave)	Mi	E	82.4Hz

Nous devons donc choisir un filtre passe-haut dont la fréquence de coupure soit inférieure à 80Hz, voir en dessous si possible et ce, afin de ne pas atténuer le signal produit par la guitare tout en supprimant son offset DC. Ainsi on a :

$$f_c = \frac{1}{2\pi R_4 C_3}$$

Avec $C_3 = 0.1\mu F$, $R_4 = 100k\Omega$ on obtient

$$f_c = 15.9Hz \leq 82.4Hz = f_{Mi\grave{ }}$$

- Le deuxième étage est un étage d'amplification du signal. Le gain d'amplification de l'AOP branché est de :

$$Gain = 1 + \frac{R_{12}}{R_{11} + R_1}$$

Si on choisit R_{11} tel un potentiomètre/trimmer de $100k\Omega$ et $R_1 = 10k\Omega$, $R_{12} = 100k\Omega$, alors le gain peut varier de 1.9 à 11, ce qui donne une bonne plage d'amplification du signal. L'utilisateur pourra tourner ce potentiomètre (ou trimmer) selon les propriétés de ses micros afin d'amplifier ou d'atténuer le signal.

- Le dernier étage donne un offset de 2.5V au signal amplifié. Un pont diviseur formée par les résistances R_{14} et R_{15} donne une tension de 2.5V à résistance égale à partir d'une alimentation de 5V fournie par la Arduino. Le condensateur polarisé électrochimique C_5 a pour fonction celle d'une petite batterie qui stocke du courant et le restitue si l'alimentation de 5V commence à faiblir. Il renforce en ce sens la stabilité du circuit.
- Finalement, le signal est envoyé à l'entrée A0 analogique de l'arduino UNO.

Un dernier point concerne l'alimentation. L'AOP est alimenté en +/- 4.5V. J'utilise une pile de 9V que je transforme en alimentation symétrique +/- 4.5V DC. L'alimentation 5V de l'arduino est utilisée pour le pont diviseur de tension.

2.2 Simulations LTSpice et résultats

La simulation LTSpice IV nous a permis de bien vérifier les points suivants :

- le filtre est bien dimensionné et ne coupe pas les fréquences intéressantes.
- l'amplification est correctement effectuée

- le signal de sorti est relevé de 2.5V
- la réponse générale du circuit est stable

J'ai décidé de modéliser la guitare par un générateur fournissant un courant sinusoïdale de fréquence $f = 83\text{Hz}$ c'est-à-dire une fréquence égale à celle que produirait la note mi grave car je voulais mettre en évidence le fait que le signal ne soit pas coupé par le filtre passe-haut. On simule alors le diagramme de bode du filtre pour des fréquences entre 1Hz et 1000Hz.

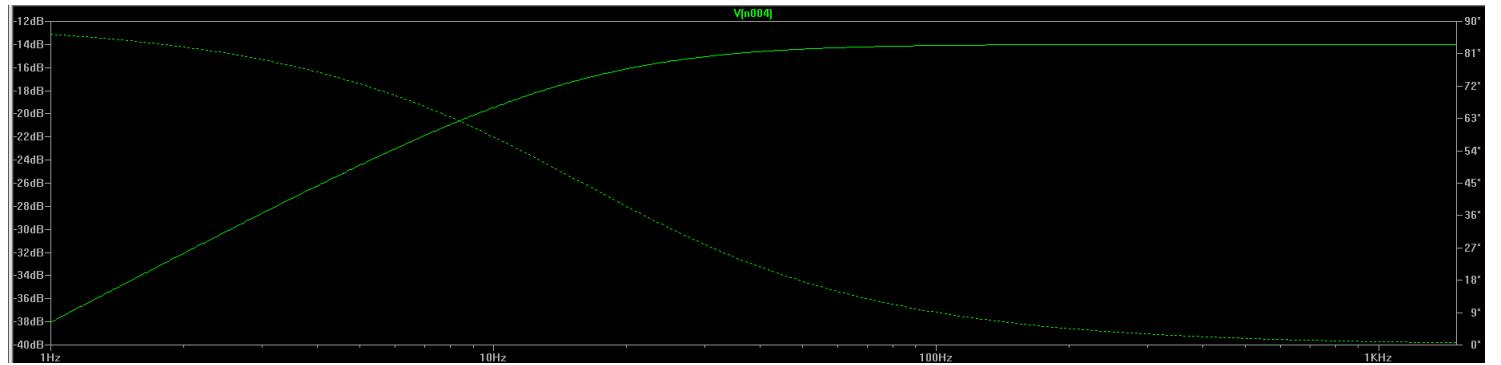


FIGURE 2: Diagramme de Bode du filtre passe haut de fréquence de coupure égal à 15Hz environ

Le diagramme de Bode nous indique une atténuation négligeable du signal aux alentours de 80Hz (échelle logarithmique). Au delà, le signal n'est pas du tout atténué.

On réalise une simulation du circuit avec en entré un signal sinusoïdal d'amplitude 400mV pic à pic, de fréquence 83Hz pendant 200ms. On obtient ce diagramme :

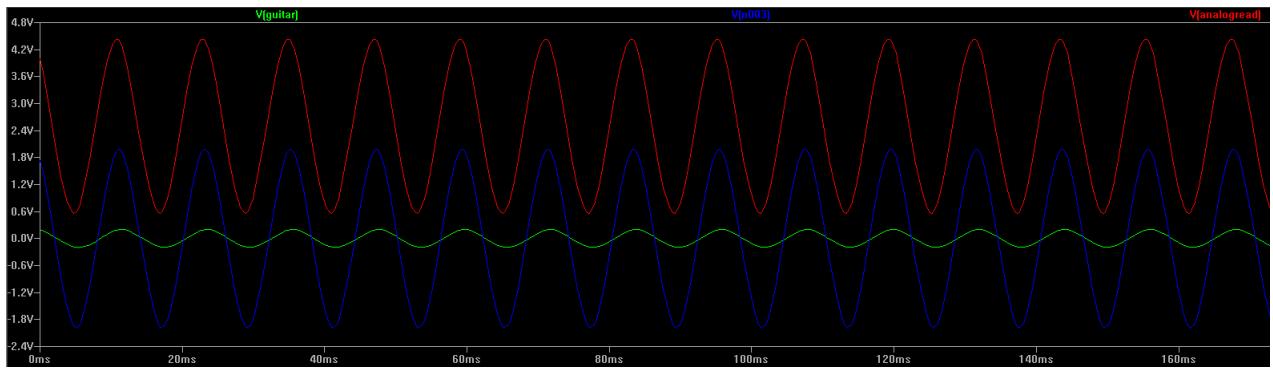


FIGURE 3: Simulation : signal vert en entrée, signal bleu après amplification, signal rouge après offset de 2.5V

Conclusion, le signal est bien amplifié selon un facteur proche de 10-11 conformément à la valeur de R_{11} de $1k\Omega$. Il n'est donc pas atténué par les filtres. Enfin, il est bien centré autour des 2.5V et s'étale dans toute la plage 0-5V.

Le signal est prêt à être échantillonné et analysé par le μC .

3 Programme Arduino UNO

3.1 Echantillonnage et buffer de données

La première étape est la récolte des données en temps réel ou échantillonnage. La fréquence d'échantillonnage de base de la carte Arduino UNO est de $9009Hz$ et correspond très bien à l'échantillonnage de signaux produits par une guitare électrique, c'est-à-dire compris entre 50 et 1500 Hz. Le principe de Shannon est bien respecté car $9009 > 2 \times 1500 \geq 2f$ où f est la fréquence réelle de la note jouée.

Il est important de signaler qu'une fréquence d'échantillonnage trop importante pourrait dramatiquement réduire les performances de l'analyse en augmentant la latence du traitement. Le principe est le suivant :

On doit recevoir un échantillon de taille suffisante pour reconnaître une période. Notre méthode analyse le signal (ou plutôt un écart quadratique de deux signaux : cf méthode d'autocorrélation d'un signal) afin de repérer 2 fois le minimum de ce dernier. Comme nous ne savons pas où se trouve potentiellement la ressemblance maximum du signal (correspondant à un minimum des écarts quadratiques des deux signaux) alors il nous faut recueillir un échantillon de taille

$$2 \times T_{MiGrave} = 2 \times \frac{1}{82.4} = 0.024s$$

car le Mi grave possède le signal de période la plus élevée. Ainsi, pour les notes plus aigues, on est sûr d'avoir au moins une période du signal dans l'échantillon.

Maintenant, avec notre fréquence d'échantillonnage, on est en mesure de calculer la taille du buffer de données : on obtient $2 \times T_{MiGrave} \times 9009 = 218.7$

Si on avait posé une fréquence d'échantillonnage plus élevée alors le buffer aurait donc été plus gros. Ainsi les temps de traitement aurait été bien plus grands !

3.2 Méthode de reconnaissance de la fréquence d'un signal avec bruit

La méthode de reconnaissance d'une note correspond à la détection de la fréquence d'un signal avec bruit. La méthode que j'ai adopté pour reconnaître une fréquence jouée en temps réel par ma guitare est celle de l'autocorrélation du signal.

3.2.1 Principe d'autocorrélation du signal

L'autocorrélation d'un signal est une convolution du signal avec lui-même mais décalé d'un temps τ . On décide de calculer l'erreur quadratique entre ces deux signaux (le signal réel avec le signal décalé d'un temps τ). Cette erreur quadratique s'exprime donc par la formule suivante :

$$EQM(\tau) = \sum_{j=1}^W (x_j - x_{j+\tau})^2$$

avec x_j le signal au temps j, W la taille de la fenêtre à considérer. Cette fonction est déclarée $\forall \tau \in [1; T_{buffer} - W]$. Nous avons calculé précédemment que la fonction EQM doit avoir un support de taille supérieur à 219. Nous choisissons donc les valeurs suivantes :

- $T_{buffer} = 300$ doit être le plus faible possible car l'échantillonnage du buffer à une fréquence de 9009Hz prend du temps. Ici en l'occurrence, $Temps_{ech} = 300 \times \frac{1}{9009} = 33ms$ ce qui reste plutôt correct en terme de latence (le cerveau humain ne détecte pas de latence en dessous d'approximativement 40ms)
- $W = 75$ plus cette valeur est grande, et plus on considère une portion de signal élevé pour le calcul de l'EQM au temps τ .
- Conclusion, on a donc bien : $T_{buffer} - W = 300 - 75 - 225 > 218.7$. Le support de la fonction EQM est de 225.

3.2.2 Calcul de fréquence

J'ai reporté les valeurs de la fonction EQM calculée par mon programme Arduino pour certaines notes jouées à la guitare (mi grave, la, ré, mi aigue).

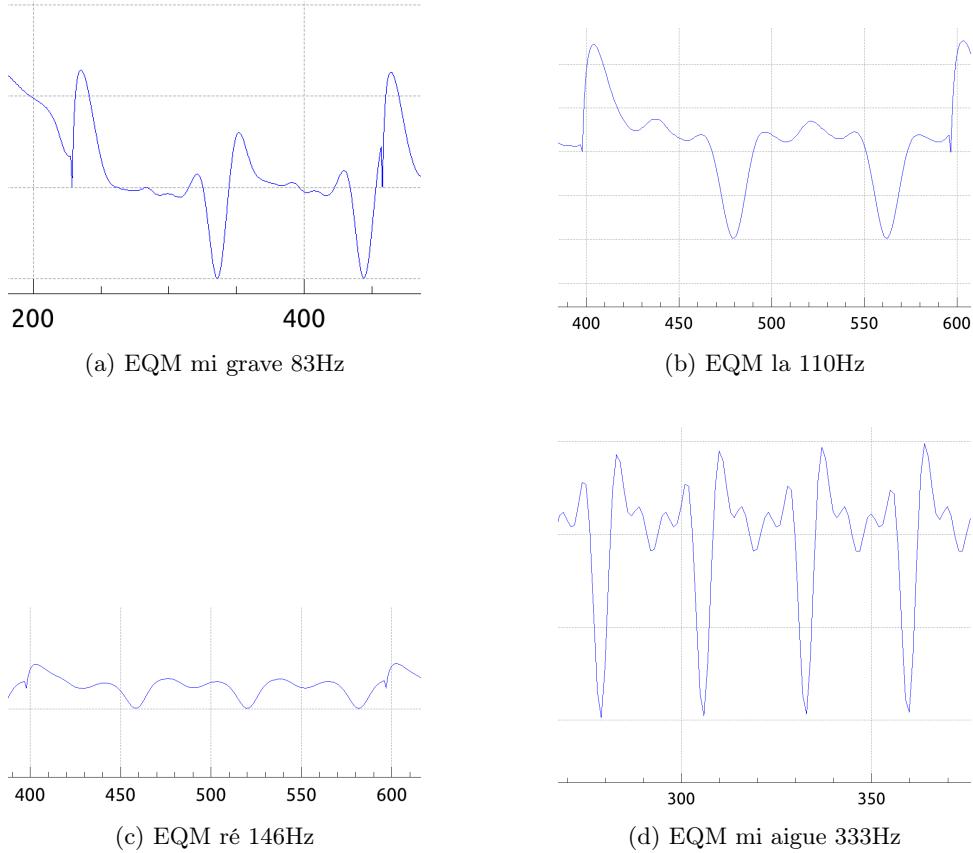


FIGURE 4: Tracé des EQM calculés par la Arduino de diverses notes

On constate que l'on retrouve à chaque fois au moins deux minimaux correspondants aux temps τ tels que l'écart entre les deux signaux soit le plus faible ! La période du signal est donc égal à la différence des deux τ tels que l'EQM soit minimal. Ainsi on retrouve f telle que :

$$f = 9009 \times \frac{1}{\tau_2 - \tau_1}$$

4 Communication Serial avec un PC Linux

La fréquence de la guitare étant reconnue, il nous faut réaliser une interface entre un logiciel de MAO et la carte Arduino UNO. En effet, le logiciel de MAO doit reconnaître en temps réel la note jouée et pouvoir superposer ses effets, banque de sons etc... J'ai décidé d'utiliser le format de communication MIDI.

4.1 Interfaçage MIDI avec les logiciels

La liaison entre le PC et la carte Arduino UNO se fait par liaison serial. L'échange MIDI exige que le débit soit de 115200 baud. A chaque fois qu'une fréquence est détectée et identifiée :

- j'envoie en liaison serial un octet pour prévenir que la note précédemment jouée est terminée. $note_{off} = 10010000$
- j'attends 30ms
- je calcule la valeur MIDI (sur un octet) de la note jouée à partir de la nouvelle fréquence reconnue en utilisant la formule :

$$note = 69.0 + 12.0 \times \log_2\left(\frac{f}{440.0}\right)$$

- j'envoie en liaison serial un octet pour prévenir qu'une nouvelle note de code $note$ doit être jouée. $note_{on} = 10000000$

4.2 Ttymidi, QjackCtl, Qsynth et Rosegarden

J'ai utilisé plusieurs softwares afin d'interfacer mon programme Arduino avec un logiciel de MAO. J'ai utilisé Rosegarden qui est disponible gratuitement sur Ubuntu. Dans l'ordre la procédure d'interfaçage s'effectue ainsi :

- on lance son programme arduino et on ouvre une communication Serial.
- on lance ttymidi à l'aide de la commande suivante :

ttymidi -s /dev/ttyACM0 -v

où ttyACM0 est le nom du port dans lequel la liaison Serial échange les données MIDI. ttymidi a pour but de faire la liaison entre ces données présentes dans le Serial, et les données MIDI qu'un logiciel de MAO peut récupérer en temps réel.

- on lance Rosegarden qui est le logiciel de MAO recevant les données MIDI et superposant des banques de sons, effets etc...
- Rosegarden envoie ensuite ces données audio modifiées à qsynth qui s'occupe dès lors de les envoyer sur la sortie audio (moniteur ou casque). Des effets de réverb et de chorus sont également disponibles à partir de l'interface de Qsynth.
- QjackCtl s'occupe de faire les liaisons entre les divers logiciels. Ainsi la chaîne d'échange de données s'organise ainsi :

ttymidi → rosearden → qsynth → sortie audio

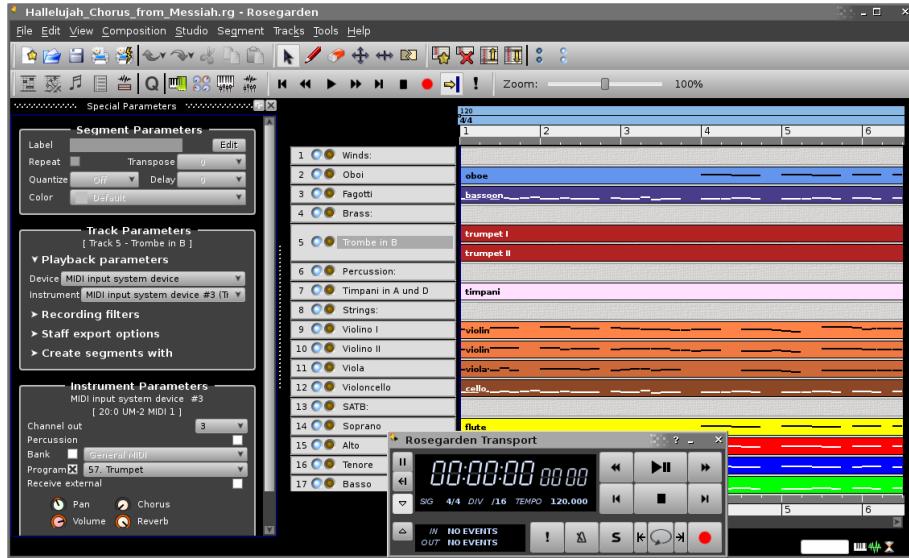


FIGURE 5: Rosegarden, logiciel de MAO gratuit avec sa banque de sons MIDI en bas à gauche

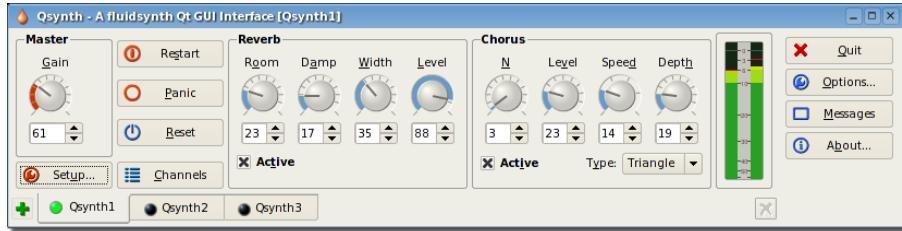


FIGURE 6: Qsynth, réverbération, chorus

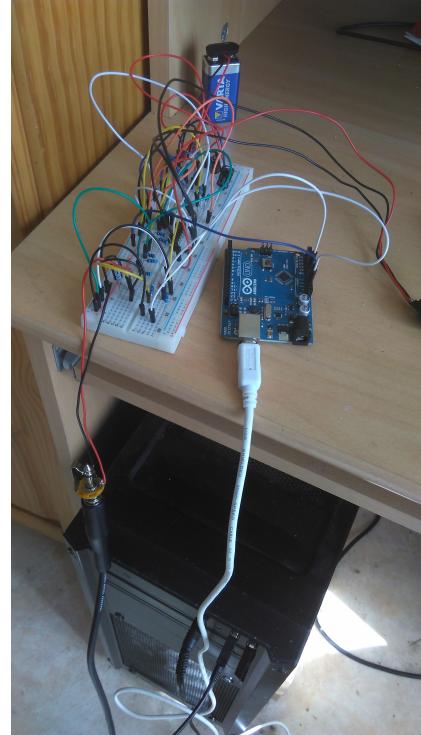
5 Conclusion

5.1 Résultats

Les résultats obtenu sont très encourageant puisque les temps d'analyse sont très faibles et les latences sont à peine perceptibles ! Il est ainsi possible de jouer une note et de la superposer à un son de violon, flute, synthétiseur, percussion car cette note est convertie en MIDI. On se sert alors de sa guitare comme d'un clavier maître ou d'un contrôleur MIDI ce qui est non seulement amusant, mais peut également être très pratique.



(a) Vue d'ensemble du projet



(b) Connexion jack à la guitare

FIGURE 7: Quelques photos du projet

5.2 Difficultées et idées d'amélioration

J'ai rencontré de nombreuses difficultés en réalisant ce projet. J'en parle afin de pouvoir donner des pistes aux groupes de l'an prochain qui souhaiterai améliorer le produit Guitare-MIDI.

- Certaines notes sont mal reconnues (notes aigues). J'ai parfois une erreur d'un demi-ton entre la note jouée et celle reconnue ce qui est audible (c'est un +/- 1 sur le code de la note MIDI identifiée). J'en déduis qu'il doit y avoir des problèmes d'approximation de calcul en nombre flottant, notamment dans mon code de calcul des fonctions EQM et EQM normalisée (cf l'article de l'algorithme de YIN).
- Certaines notes aigues (très peu) sont reconnues mais une ou deux octaves plus basses. Là encore, l'autocorrélation a peut-être rogné certains calculs ce qui a pu occasionner des ressemblances plus fortes à une ou deux octaves plus basses.
- Il est difficile de savoir lorsqu'une note a été jouée. Ainsi, l'analyse de

la valeur absolue de l'amplitude d'un signal dépassant un certain seuil n'est pas suffisant (une note a pu être jouée il y a longtemps et de façon très forte par le guitariste ce qui a pour effet de la maintenir encore au delà du seuil). Une idée serait d'analyser les variations du signal (sa dérivée) afin de constater lorsque qu'il y a une variation soudaine du signal (correspondant à une note jouée).

Un dernier point important à souligner est que cette méthode ne permet que d'analyser une seule note jouée à la fois. La détection fréquentielle n'est donc pas multiple mais mono. Une idée d'amélioration (très compliquée, basée sur des statistiques d'accord joués) pourrait être développée afin de détecter un accord joué ou plusieurs notes.

Références

- YIN Algorithm pour comprendre l'autocorrélation d'un signal échantillonné
- Référence pour apprendre à programmer son arduino
- Méthode similaire à la mienne, basée sur la convolution du signal
- Mine d'informations sur l'électronique audio, l'amplification de signaux...
- Interfaçage Serial/MIDI à l'aide de ttymidi