DENOYELLE Fabien, POULHE Lucas, VATTEVILLE-REVEILLON Eloïne

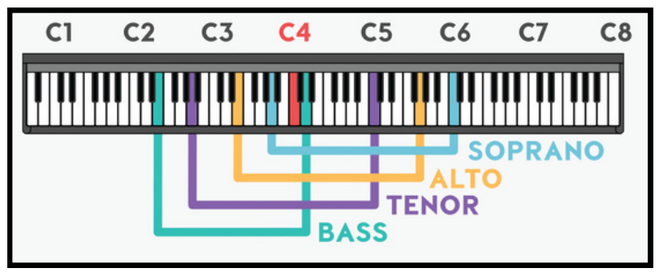
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

PROJET SON - 3E ANNÉE TC - 2023 2024

Création d'un Auto-Tune

Notre proposition

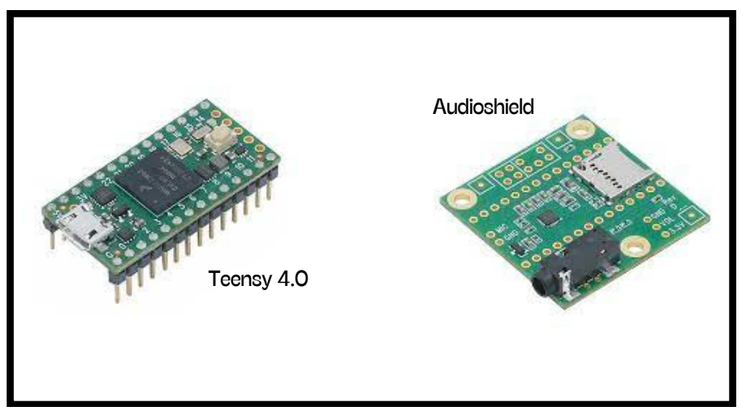
Nous avons tenté de réaliser un auto-tune conçu pour fonctionner sur la voix humaine à l'aide du Teensy. Il permet de corriger des fréquences allant de 65 à 1975 Hz ce qui correspond à 5 octaves allant du Do2 au Si6, la note basse d'une tessiture de basse étant estimée un Mi2 et la note aigüe d'une tessiture soprano un C6, voire plus. Utilisant des gammes chromatiques, notre auto-tune permet d'ajuster le signal reçu au demi-ton près.



Notes chantées par des voix de différentes tessitures [1]

Hardware

Le traitement du son sera réalisé grâce au **Teensy 4.0**, un microcontrôleur spécifiquement conçu pour le traitement du son en embarqué, en temps réel, grâce à un **processeur beaucoup plus puissant** que ses homologues de la très connue marque Arduino.
La réalisation de l'acquisition et de la restitution audio est effectuée au travers de la **carte son Audioshield** à laquelle on le connecte directement.



Teensy 4.0

Qu'est ce que l'auto-tune?

L'Auto-tune est un module de traitement audio qui permet de 'corriger' les notes d'une mélodie reçue afin qu'elles sonnent plus justes. En live, il est souvent utilisé sur la voix de chanteurs pour garantir qu'ils chantent juste, ou pour ajouter un effet sur de la voix parlée, par exemple dans le rap.
Le principe de base est le suivant: à l'aide d'une analyse fréquentielle, la fréquence fondamentale détectée dans le son reçu est associée à la note de la gamme chromatique dont la fréquence est la plus proche, puis le signal reçu est transposé pour être ramené à cette note.

Réalisation

Détection de fréquence

Pour trouver la note chantée par la personne utilisant l'objet, nous utilisons une fonction de la librairie teensyduino qui implémente l'algorithme de Yin conçu pour la **détection de la fréquence fondamentale** d'un signal audio.
A partir de la fréquence détectée par cette fonction, nous effectuons une **recherche dichotomique de la note la plus proche** dans une liste de notes prédéfinies et une fois qu'elle est déterminée on transpose le signal à cette fréquence.

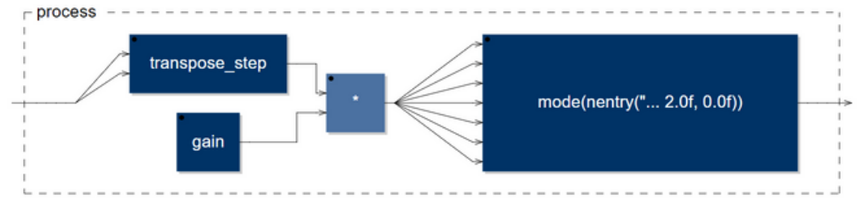


Diagramme de fonctionnement du programme Faust utilisé
transpose_step réalise la transposition et mode la sélection des accords

Approche Naïve (fonction hardware)

L'une des solutions que nous avions imaginé était d'utiliser une Transformée de Fourier Discrète pour modifier les fréquences.
Le décalage, déterminé grâce à l'algorithme de Yin, aurait été appliqué au spectre fréquentiel, puis une Transformée de Fourier Inverse aurait permis de repasser dans le domaine temporel.

Algorithmes

Deux algorithmes différents:

- Algorithme en **$O(n^2)$**.
  - Simple à implémenter
  - Mauvaise efficacité temporelle (ne peut pas être utilisé pour du temps réel)
- Algorithme en **$O(n.log(n))$** : **FFT**
  - Complexe à implémenter
  - Beaucoup plus efficace
  - Le processeur du Teensy admet une **fonction hardware,** c'est-à-dire une fonction physiquement intégrée dans le processeur pour réaliser la FFT. Cela permet d'avoir une **vitesse d'exécution grandement améliorée**.

Problèmes

Pour augmenter ou diminuer les fréquences afin que l'harmonique principale corresponde à la note jouée, il faut avoir **une résolution très fine** sur le spectre (ex. 3Hz). Cependant, ayant une fréquence d'échantillonnage très grande (44.1 kHz), il faudrait alors pouvoir réaliser une **FFT sur un très grand nombre d'échantillons** (14700), et cela est **chronophage**.
Une FFT suivie d'une IFFT sur 1024 échantillons prends autant de temps que l'acquisition des échantillons. Il n'y a donc pas de temps disponibles pour faire du traitement en temps réel.
Il est **impossible de réaliser un traitement correct via cette méthode**.

La **résolution en amplitude est grandement diminuée** par la fonction hardware.
Une FFT suivie d'une IFFT sur 1024 échantillons codées sur 16 bits, l'amplitude en sortie sera **codée sur 6 bits** (valeurs réparties entre -32 et 32).
Cela n'est **pas assez précis** pour avoir un son suffisamment qualitatif en sortie. Cette lacune de précision est due à l'implémentation hardware et **nous ne pouvons donc malheureusement pas y remédier**.

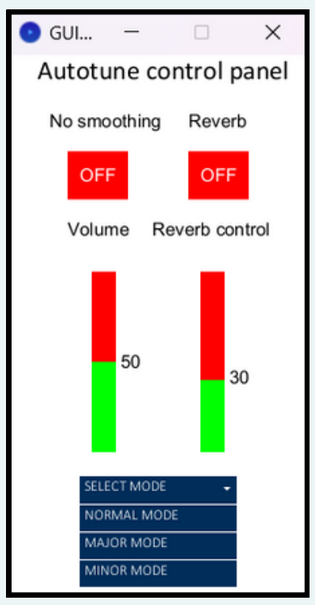
Approche Finale (Faust)

La version finale de notre programme fonctionne à l'aide de Faust dont la librairie d'effets contient une **fonction de transposition** qui prend en entrée l'écart en demi-tons entre les notes traitées. Les notes perçues par l'être humain étant reliées à une échelle logarithmique de base 2 et une gamme chromatique étant composée de douze notes,
l'écart est calculé ainsi :

$$\log_2\left(\frac{\text{fréquence la plus proche}}{\text{fréquence détectée}}\right) \times \frac{1}{12}$$

A l'aide de Faust nous avons choisi d'amplifier l'effet 'son auto-tune' prisé des rappeurs en ajoutant la possibilité de contrôler les effets suivants :

- désactivation du lissage lors des changements de facteur de transposition
- réverbération sur le son en sortie (à partir d'une fonction de la teensy audio library)
- génération d'un accord majeur ou mineur en ajoutant des copies du son d'entrée transposé



Interface graphique pour contrôler le système

Interface graphique

Logiciel **Processing** avec la bibliothèque **ControlP5**

- passage des données par le port série.
- envoi d'un octet → 255 valeurs
- action en fonction de la valeur reçue
- traitement en direct

[1] Flypaper, novembre 2016 https://flypaper.soundfly.com/play/how-to-find-your-vocal-range-and-write-it-on-a-resume/