Vocalyz : Transformée de fourier et musique

Introduction

Le but de ces recherches est de parvenir, en post-processing, a analyser les données contenu dans un fichier WAV préalablement enregistré afin d'en extraire les fréquences a partir d'un algorithme de transformée de fourier rapide. On procèdera donc avec deux aproches: Une approche fréquencielle : On analyse les resultat de la FFT, intensité et fréquence). Puis une approche temporelle, ou on traite ces données dans le temps.

Les données que nous utilisons seront donc dans un plan a trois dimensions (fréquences en hertz, temps en millisecondes et intensitée sonore en $W.m^{-2}$ (et/ou niveau d'intensité sonore, en décibels). Les deux semblent indispensable pour parvenir a des résultats concluants. Dans ce cas, on analysera des morceaux de piano, on s'interessera spécifiquement a cet instrument. La tâche semble hardu, et aucune technologies aujourd'hui, ne permet avec certitude de detecter parfaitement les pitchs (les hauteurs) d'un signal complexe (https://en.wikipedia.org/wiki/Pitch_detection_algorithm).

Défis notables :

-L'utilisateur joue plusieurs notes simultanément. En effet, certain pics de fréquences sont commun a plusieurs notes. Cela créer une ambiguïté lors de la détection.
-L'utilisateur joue deux notes, « a l'octave ». A4 et A3 par exemple.
-Si le piano est désaccordé et que chaque note et désaccordé différemment , il semble impossible d'arriver a détecter les note correctement (sauf chaque note est désaccordée de moins d'un demi-ton? => Trouver l'écart correspondant a un demi-ton en hertz)

Voici les différents outils utilisés dans ces travaux de recherches :

-C#, Visual Studio 2017 .NET Framework 4.5
-La librairie open-source Naudio pour C#. (https://github.com/naudio/NAudio).
-La librairie open-source MonoGame (https://github.com/MonoGame/MonoGame). (librairie de rendu graphique utilisée pour les tests).

La librairie Naudio nous fournie plusieurs fonctions utiles :

-Une fonction effectuant une transformée de fourier rapide, dont l'unique sortie est un tableau de « Complex » (type représentant un complexe mathématique. Je n'ai aucune idée de la signification physique des valeurs x et yi mais je sais que l'intensité liée a une fréquence est égale au module de ce complexe :

$$I = \sqrt{Im(z)^2 + Re(z)^2}$$

```
Math.Sqrt(c.X * c.X + c.Y * c.Y);
```

-Une fonction de fenêtrage (Hamming Window)

Principes

Mon idée pour arriver a détecter les notes est basé sur un principe autocorrection temporel et fréquentiel. Il s'agit de comparer les fréquences observée avec une base de donnée locale qui connaît les 'pattern' approximatifs pour les différentes notes d'un piano. On peut déjà soulever le fait que d'un piano a un autre, le timbre change (on pourra d'ailleurs mettre en place un système en algorithme génétique qui apprendrait les différents timbre de piano, le serveur et le multiclient étant une plate-forme idéale. Sans parler du C# impératif, structuré et orienté objet qui nous laisse une flexibilité totale quand a son implémentation).

On peut retrouver la fréquence des notes de piano 88 touches (standard) accordée avec un A4 de référence a 440hz a partir de cette formule :

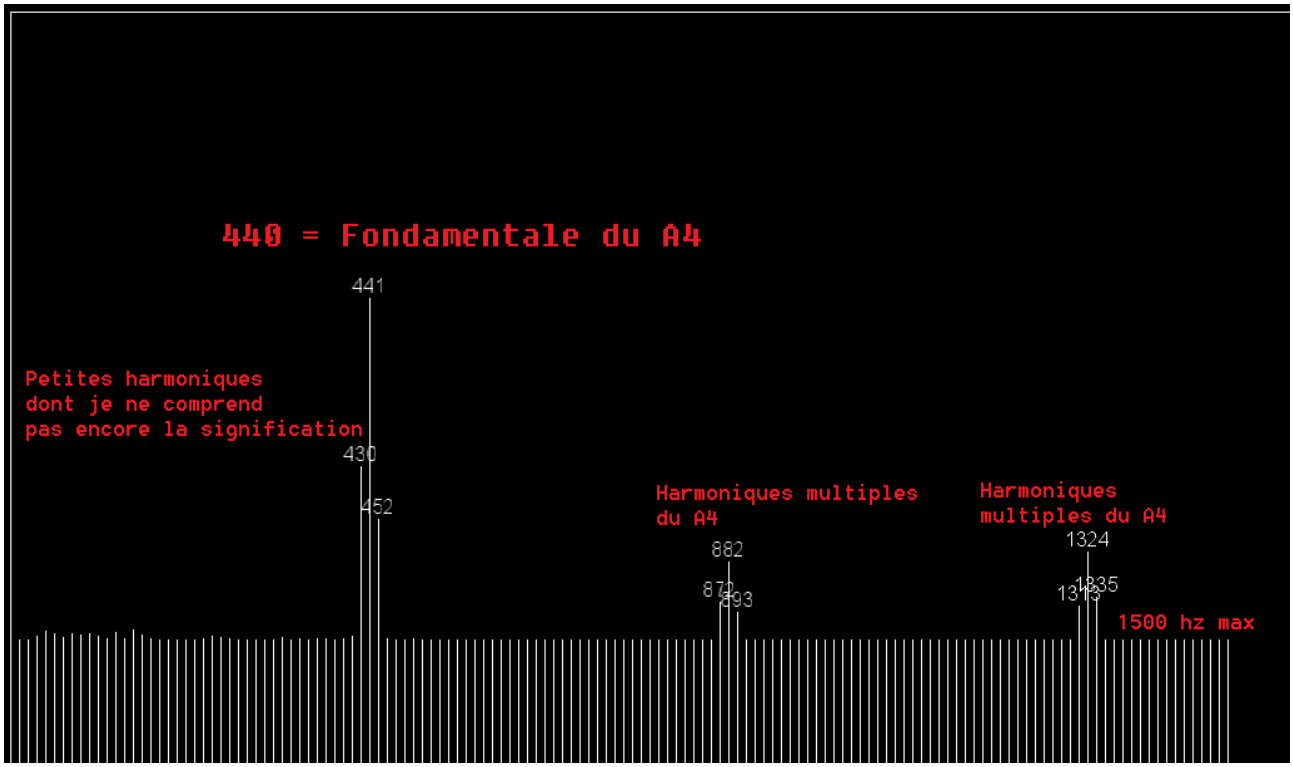
$$f(n) = \left(\sqrt[12]{2}\right)^{n-49} \times 440\,\mathrm{Hz}$$

Ma première approche fut de sélectionner la fréquence la plus élevé a un instant t, figé et rechercher une note dont la fréquence de référence étant proche, (on commence par une détection de note unique. L'approche temporelle dont je parlais précédemment n'est pas nécessaire dans ce cas). Mais il se trouve que cette détection est très naïve et même parfois loin des son perçu par l'oreille humaine. Il y a des situations, on si l'on joue un A4 sur un piano standard, une des harmoniques de ce A (880 hz par exemple) dépassera un cours instant en intensité la fréquence fondamentale a 440 hz (généralement lorsque l'intensité de la note baisse) pourtant l'oreille humaine entend bien un A4 et non un A5 ! La représentation graphique de la FFT m'a fait comprendre qu'une approche temporelle serait indispensable pour détecter plusieurs notes joué en même temps. (En effet, la baisse d'intensité en fonction du temps semble proportionnelles (du moins il semble y avoir une relation) en fonction du temps pour chaque

harmoniques d'une note).

Prémices et implémentation

La première étape du projet fut une représentation sous forme de 'plot'. Qui nous permet d'observer les pics de fréquence de la transformée de fourrier d'un morceau de piano au format .WAV Voici le rendu pour un son de A4 piano, enregistré en studio :



Note a moi même : je pense que les petites harmoniques donne un timbre au piano, je ne suis pas sur, sont elle multiples également avec celle proche du pics a 882 hz ?

Taille de la FFT : 4096
Fréquence d'échantillonnage : 44100 hz
Fréquence Minimum affichée : 0hz
Fréquence Maximum affichée : 1500 hz

La raison pour laquelle la valeur observé est 441 hz au lieu de 440 hz est due au fait que la taille de la FFT soit égale a 4096 et non a la fréquence d'échantillonnage du fichier source :

Pour avoir une précision au hertz prêt pour un fichier audio WAV de fréquence d'échantillonnage 44100. Il nous faut une FFT de taille égale a la fréquence d'échantillonnage, a savoir 44100 (ce qui est très très très gros, et lourd en terme de calcul, de plus on ne s'intéresse qu'au fréquence <= a 8000 hz, (note la plus aiguë d'un piano), car avec une FFT de taille 44100, on peut observer les pics de fréquence <= 44100 hertz ! D'où l'importance de modifier la fréquence d'échantillonnage après l'enregistrement afin d'optimiser le temps de calcul et surtout la fréquence de rafraîchissement de la FFT (il ne faut pas que le temps de calcul soit inferieur a celui pris par une petite note rapide jouée par le musicien!) + (n'oublions pas la nécessitée de l'approche temporelle!)

Questions

-La librairie NAudio est une librairie très haut niveau. Et mes connaissance en analyse du son ne sont que partielles, surtout a bas niveau. L'intensité de chaque fréquence de la FFT est générée a partir d'un nombre flotant (value). Dont je ne connais pas du tout la signification, il est pourtant très important et me semble indispensable a comprendre avant d'aborder la question du suréchantillonage.

```
fftBuffer[fftPos].X = (float)(value * FastFourierTransform.HammingWindow(fftPos,
fftLength));
```

Ce nombre flotant est lui même issu d'un buffer, float[] qui représente sans doute les données du fichier. Mais quelles sont ces données ? Des 'samples' d'après la documentation de Naudio. Probablement les échantillons obtenu a partir de la conversion analogique → numérique, mais quels sont leurs unitées ? Que représentent ils ? Les variations de la compression de l'air ? (#jesuisperdu).

-Afin de produire une transformée de fourrier correcte, la taille de la FFT doit être une puissance de 2. Sinon, les fréquence calculées sont fausse, pourquoi ?

Ex : Constante experimentale trouvée pour une taille de FFT 44100 (fréquence d'échantillonage du fichier) : `1.345565749235474`. Cette constante semble correcte. A quoi correspond elle ? Est elle valide ?

-Il semble que plus la fréquence soit haute, plus l'intensité liée soit faible pour une note qui nous parrait d'une intensitée équivalente a l'oreille, pourquoi ?

Faut-il créer un rapport pour augmenter l'intensité théorique d'une fondamentale a fréquence plus élévée ? Probablement lors de la conversion midi ? Peut être même avant ?

-A partir de quelle intensitée considère t-on une fréquence comme présente dans notre spectre de fourier ?

Actuellement, j'ai considéré que l'intensité est significative a partir de 25 décibels. Cette valeur est totalement arbitraire. Faut il normaliser l'intensité générale du morceau en fonction de des pics d'intensités du morceau ? Créer une sorte de médiane d'intensité qui sera notre seuil significatif ?

-Une telle fonction est telle viable ?

```
public static Note GetNoteByFrequency(double frequency, int frequencyGap = 6)
{
            return Notes.FirstOrDefault(x => x.Frequency > frequency - frequencyGap &&
            x.Frequency < frequency + frequencyGap);
}
```

-Quelle est la différence entre ces deux formules ?

```
this.DB = 10 * Math.Log10(intensity);
this.DB = 20 * Math.Log10(intensity);
```

-A quoi sert la fonction de fenetrage Hamming Window ?

Est-ce justement la normalisation des intensités dont je parlais précédemment ?

Implémentation Naudio.

Codes important : SampleAggregator.cs, issue de Naudio dont j'ai encore du mal a comprendre le fonctionnement de la fonction `private void Add(float value)`

```
using NAudio.Dsp;
using NAudio.Wave;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Vocalyz.Audio
{
    public class SampleAggregator : ISampleProvider
    {
        // volume
        private float maxValue;
        private float minValue;
        int count;

        // FFT
        private readonly Complex[] fftBuffer;
        private readonly FftEventArgs fftArgs;
        private int fftPos;
        private readonly int fftLength;
        private readonly int m;
        private readonly ISampleProvider source;
```

```
        private readonly int channels;

        public SampleAggregator(ISampleProvider source, int fftLength = 1024)
        {

            channels = source.WaveFormat.Channels;
            if (!IsPowerOfTwo(fftLength))
            {
                 //  throw new ArgumentException("FFT Length must be a power of two");
            }
            m = (int)Math.Log(fftLength, 2.0);
            this.fftLength = fftLength;
            fftBuffer = new Complex[fftLength];
            fftArgs = new FftEventArgs(fftBuffer);
            this.source = source;
        }

        static bool IsPowerOfTwo(int x)
        {
            return (x & (x - 1)) == 0;
        }

        public void Reset()
        {
            count = 0;
            maxValue = minValue = 0;
        }


        private void Add(float value)
        {
            if (PerformFFT && FftCalculated != null)
            {
                fftBuffer[fftPos].X = (float)(value *
FastFourierTransform.HammingWindow(fftPos, fftLength));   // what is that ?
                fftBuffer[fftPos].Y = 0;
                fftPos++;
                if (fftPos >= fftBuffer.Length)
                {
                    fftPos = 0;
                    // 1024 = 2^10
                    FastFourierTransform.FFT(true, m, fftBuffer);
                    FftCalculated(this, fftArgs);
                }
            }

            maxValue = Math.Max(maxValue, value);
            minValue = Math.Min(minValue, value);
            count++;
        }

        public WaveFormat WaveFormat => source.WaveFormat;   // informations descriptives du
fichier (Fréquence d'échantillonage, Bit per samples, etc)

        public int Read(float[] buffer, int offset, int count)   // Il me semble que ce
buffer de floats, est une conversion directe du buffer byte[] (données brutes du fichier
sources) en buffer... floats ^_^ tout simplement.
        {
            var samplesRead = source.Read(buffer, offset, count);

            for (int n = 0; n < samplesRead; n += channels)
            {
                Add(buffer[n + offset]);
            }
            return samplesRead;
```

```
    }
}

public class MaxSampleEventArgs : EventArgs
{
    [DebuggerStepThrough]
    public MaxSampleEventArgs(float minValue, float maxValue)
    {
        MaxSample = maxValue;
        MinSample = minValue;
    }
    public float MaxSample { get; private set; }
    public float MinSample { get; private set; }
}

public class FftEventArgs : EventArgs
{
    public FftEventArgs(Complex[] result)
    {
        Result = result;
    }
    public Complex[] Result { get; private set; }
}
```

Merci d'avoir pris le temps de lire ! N'hésitez pas a me répondre ou a me poser des questions !

Cordialement,

Marius Lumbroso

axiom1008@gmail.com

Bonne journée / soirée