Audio Analyzer

Author: Luciano Ciamarone

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 frammenti_prev Namespace Reference

Functions

- def rotate (average, new_value)

Variables

- string instrument = "piano"
- string note = "Beat_It_cut"
- float MOVING_AVARAGE_P = 0.1
- float ATTACK_FACTOR = 0.6
- list average = [0.] ∗ 20
- list moving_avg = [ ]
- list moving_avg_mia = [ ]
- list threshold = [ ]
- float value = 0.05
- list real_signal = [ ]
- float p = MOVING_AVARAGE_P;
- value_mio
- th = pow(real_signal[i],2) / value_mio
- color

4.1.1 Detailed Description

```
if note:
    if vectplot[note-20] == 0:
        _event_buffer[note - 20] = _event_buffer[note - 20] + 10



    if note + 20 in scoreout: # the note is preset in the score
        out_n.write("\t" + str(note + 20))
        _event_buffer[note] = 100
    elif note + 1 in scoreout: # this could be a second harmonic
        seconds.append(note + 20)
    elif note + 8 in scoreout and note + 27 in seconds: # this could be a first harmonic
        firsts.append(note + 20)
elif _event_buffer[note] < 100 and _event_buffer[note]/10 > wait:
    out_n.write("\t!" + str(note + 20))
    _event_buffer[note] = 0
```

4.1.2 Function Documentation

4.1.2.1 rotate()

```
def frammenti_prev.rotate (
              average,
              new_value )
```

4.1.3 Variable Documentation

4.1.3.1 ATTACK_FACTOR

```
float frammenti_prev.ATTACK_FACTOR = 0.6
```

4.1.3.2 average

```
frammenti_prev.average = [0.] * 20
```

4.1.3.3 color

```
frammenti_prev.color
```

4.1.3.4 instrument

```
string frammenti_prev.instrument = "piano"
```

4.1.3.5 MOVING_AVARAGE_P

```
float frammenti_prev.MOVING_AVARAGE_P = 0.1
```

4.1.3.6 moving_avg

```
frammenti_prev.moving_avg = []
```

4.1.3.7 moving_avg_mia

```
list frammenti_prev.moving_avg_mia = []
```

4.1.3.8 note

```
string frammenti_prev.note = "Beat_It_cut"
```

4.1.3.9 p

```
float frammenti_prev.p = MOVING_AVARAGE_P;
```

4.1.3.10 real_signal

```
frammenti_prev.real_signal = []
```

4.1.3.11 th

```
frammenti_prev.th = pow(real_signal[i],2) / value_mio
```

4.1.3.12 threshold

```
list frammenti_prev.threshold = []
```

4.1.3.13 value

```
float frammenti_prev.value = 0.05
```

4.1.3.14 value_mio

```
frammenti_prev.value_mio
```

4.2 logic_guit Namespace Reference

Variables

- string nota = "Amag_open"

    nota and chitarra are useful to compose the name of the score file to load.
- string chitarra = "poly"
- string arg1 = "./scores/" + chitarra + "/" + nota + "_filefft.out"
- string arg2 = "./scores/" + chitarra + "/" + nota + "_filefilter.out"
- string arg3 = "./scores/" + chitarra + "/" + nota + "_monodet.out"
- string arg4 = "./scores/" + chitarra + "/" + nota + "_integer.out"
- datafft = genfromtxt(arg1)
- data = genfromtxt(arg2)
- fund = genfromtxt(arg3)
- env = genfromtxt(arg4)
- float envprev = 10.0
- freq = genfromtxt("./algo_outputs/notes.txt")
- int existsum = 0
- float sens = 0.1
- float Pshp = 1.03
- float Pshs = 1.025
- float Pthp = 0.6∗sens
- float Pths = 0.7∗sens
- list vectmono = [0]∗51
- list vectnote = [0]∗51
- list memoria = [0]∗51
- list exist = [0]∗51
- float allowance = 0.0

    using th attack to enable / disable the allowance
- f = open("./algo_outputs/names.txt","r")
- labels = f.readlines()
- list names = [ ]
- name = str(labels[k]).split('\n')
- float pause = 0.09
- int condizione = 0

    For each chunk (i) we iterate over the 51 notes (j)
- int dummywarn = 0
- list vectcond = [1.5]∗51
- list jvect = [1.4]∗51
- list vectplot = [0]∗51
- list vectfft = [0]∗51
- list vect = [0]∗51
- list existplot = [0]∗51
- list baseline = [0]∗51
- portion = int(len(data)/4)

    Portion is used to establish the duration of the plot.
- int scarto = 0

- tuple time = (i-scarto) ∗ 256 / 44100. ∗ 1000
- tuple ratio2 = (datafft[i,j+12]+datafft[i,j+19])∗0.5/datafft[i,j-7]
- ratio1 = datafft[i,j+12]/datafft[i,j-7]
- float attack = env[i]/envprev∗0.5

  calculating the attack based on the enevelopes at the current chunck and the one before
- minimo = min(vect)

  plot-only variables
- color

4.2.1 Variable Documentation

4.2.1.1 allowance

```
float logic_guit.allowance = 0.0
```

using th attack to enable / disable the allowance

4.2.1.2 arg1

```
string logic_guit.arg1 = "./scores/" + chitarra + "/" + nota + "_filefft.out"
```

4.2.1.3 arg2

```
string logic_guit.arg2 = "./scores/" + chitarra + "/" + nota + "_filefilter.out"
```

4.2.1.4 arg3

```
string logic_guit.arg3 = "./scores/" + chitarra + "/" + nota + "_monodet.out"
```

4.2.1.5 arg4

```
string logic_guit.arg4 = "./scores/" + chitarra + "/" + nota + "_integer.out"
```

4.2.1.6 attack

```
float logic_guit.attack = env[i]/envprev*0.5
```

calculating the attack based on the enevelopes at the current chunck and the one before

4.2.1.7 baseline

```
list logic_guit.baseline = [0]*51
```

4.2.1.8 chitarra

```
string logic_guit.chitarra = "poly"
```

4.2.1.9 color

```
logic_guit.color
```

4.2.1.10 condizione

```
int logic_guit.condizione = 0
```

For each chunk (i) we iterate over the 51 notes (j)

Resetting the variables.

data.T is data transposed meaning the 51 possible notes
I create the vector of relative maximums vectnote and vectplot (used only for plotting) and then I check the memoria variable to evaluate if increasing or not the stability index exist

Then I iterate againg over the possible notes

for evaluating the various conditions

4.2.1.11 data

```
logic_guit.data = genfromtxt(arg2)
```

4.2.1.12 datafft

```
logic_guit.datafft = genfromtxt(arg1)
```

4.2.1.13 dummywarn

```
int logic_guit.dummywarn = 0
```

4.2.1.14 env

```
logic_guit.env = genfromtxt(arg4)
```

4.2.1.15 envprev

```
logic_guit.envprev = 10.0
```

4.2.1.16 exist

```
list logic_guit.exist = [0]*51
```

4.2.1.17 existplot

```
list logic_guit.existplot = [0]*51
```

4.2.1.18 existsum

```
int logic_guit.existsum = 0
```

4.2.1.19 f

```
logic_guit.f = open("./algo_outputs/names.txt","r")
```

4.2.1.20 freq

```
logic_guit.freq = genfromtxt("./algo_outputs/notes.txt")
```

4.2.1.21 fund

```
logic_guit.fund = genfromtxt(arg3)
```

4.2.1.22 jvect

```
list logic_guit.jvect = [1.4]*51
```

4.2.1.23 labels

```
logic_guit.labels = f.readlines()
```

4.2.1.24 memoria

```
list logic_guit.memoria = [0]*51
```

4.2.1.25 minimo

```
logic_guit.minimo = min(vect)
```

plot-only variables

4.2.1.26 name

```
logic_guit.name = str(labels[k]).split('\n')
```

4.2.1.27 names

```
list logic_guit.names = [ ]
```

4.2.1.28 nota

```
string logic_guit.nota = "Amag_open"
```

nota and chitarra are useful to compose the name of the score file to load.

The scores are generated from the C++ algorithm and are moved to their folder by a script Here we are calling the four scores and extracting the data from them using the python module genfromtxt

4.2.1.29 pause

```
float logic_guit.pause = 0.09
```

4.2.1.30 portion

```
logic_guit.portion = int(len(data)/4)
```

Portion is used to establish the duration of the plot.

It is defined as a fraction of the duration of the audiofile If we want to analyze a precise number of chunks, we need to insert that integer number

4.2.1.31 Pshp

```
float logic_guit.Pshp = 1.03
```

4.2.1.32 Pshs

```
float logic_guit.Pshs = 1.025
```

4.2.1.33 Pthp

```
float logic_guit.Pthp = 0.6*sens
```

4.2.1.34 Pths

```
float logic_guit.Pths = 0.7*sens
```

4.2.1.35 ratio1

```
logic_guit.ratio1 = datafft[i,j+12]/datafft[i,j-7]
```

4.2.1.36 ratio2

```
tuple logic_guit.ratio2 = (datafft[i,j+12]+datafft[i,j+19])*0.5/datafft[i,j-7]
```

4.2.1.37 scarto

```
int logic_guit.scarto = 0
```

4.2.1.38 sens

```
float logic_guit.sens = 0.1
```

4.2.1.39 time

```
int logic_guit.time = (i-scarto) * 256 / 44100.  * 1000
```

4.2.1.40 vect

```
logic_guit.vect = [0]*51
```

4.2.1.41 vectcond

```
list logic_guit.vectcond = [1.5]*51
```

4.2.1.42 vectfft

```
logic_guit.vectfft = [0]*51
```

4.2.1.43 vectmono

```
list logic_guit.vectmono = [0]*51
```

4.2.1.44 vectnote

```
list logic_guit.vectnote = [0]*51
```

4.2.1.45 vectplot

```
list logic_guit.vectplot = [0]*51
```

4.3 logic_piano Namespace Reference

Functions

- def diagnostic (nbuffer, scoreout, vectplot, vectnote, vactmaxrel, energymin, fftmin, wait=8)
- def print_folding (segment, all_data)
- def print_correspondence (index)

Variables

- string note = "While_my_guitar"

  note and instrument are useful to compose the name of the score file to load.
- list names = ["A","A#","B","C","C#","D","D#","E","F","F#","G","G#"]
- string instrument = "piano"
- freq = genfromtxt("./algo_outputs/notes_piano.txt")
- string arg1 = "./scores/" + instrument + "/" + note + "_filefft.out"
- string arg2 = "./scores/" + instrument + "/" + note + "_filefilter.out"
- string arg4 = "./scores/" + instrument + "/" + note + "_integer.out"
- datafft = genfromtxt(arg1)
- data = genfromtxt(arg2)
- env = genfromtxt(arg4)
- scoreout = open("/home/luciamarock/Documents/cpp_tests/PitchDetector/src/integer.out", "r")
- list midiscore = [ ]
- temp = line.split("\n")
- line = temp[0]

- list elemout = [ ]
- elem = line.split("\t")
- float envprev = 10.0
- float noise_threshold = 3.0
- float sens = 5.0
- float firstharmth = 4.3/sens
- float scndharmth = 3.999/sens
- float rtfithreshold = 4.601162791/sens
- float stability_threshold = 3.5 / sens
- list vectgrey = [0]∗89
- list vectnote = [0]∗89
- list vectplot = [0]∗89
- list memory = [0]∗89
- list exist = [0]∗89
- list not_presence = [0]∗89
- float allowance = 0.0

  using the attack to enable / disable the allowance
- int activenotes = 0
- float pause = 0.09
- int dummywarn = 0
- list jvect = [1.4]∗89
- list vectfft = [0]∗89
- list vect = [0]∗89
- list baseline = [0]∗89
- list redline = [0]∗89
- list scndharmthline = [0]∗89
- list rtfithresholdline = [0]∗89
- int exclude = 1
- int i2monitor = 943
- int j2monitor = 42
- out_f = open("vectplot_printout.txt", "w")
- out_n = open("output_evaluated_printout.txt", "w")
- out_s = open("midiscore_printout.txt", "w")
- portion = int(len(midiscore)/1.)

  Portion is used to establish the duration of the plot.
- int plot_duration = 390
- int scarto = 0
- int bluemax = 0
- int blueidx = -1
- int fftmax = 0

  For each chunk (i) we iterate over the 89 notes (j)
- tuple time = (i-scarto) ∗ 256 / 44100. ∗ 1000
- int energymin = bluemax∗rtfithreshold

  Then I iterate againg over the possible notes.
- int fftmin = fftmax∗firstharmth
- bool NOpattern = False
- tuple avfft = (vectfft[j] + vectfft[j+12] + vectfft[j+19])/3.
- tuple avrfti = (vectnote[j] + vectnote[j+12] + data[i,j+19]/170)/3.
- tuple totalenergy = avrfti ∗ avfft ∗ 2.
- maxrfti = n.max(vectplot)
- maxidx = n.argmax(vectplot)
- float AVenergymin = maxrfti∗stability_threshold
- float attack = env[i]/envprev∗0.5

  calculating the attack based on the enevelopes at the current chunck and the one before

- tuple time_from_beginning = (i+1)/44100. ∗ 256
- color
- list th_line = [ ]
- list fft_line = [ ]
- list vectcond = [1.5]∗89

  plot-only variables
- list vectmono = [0]∗89

4.3.1 Function Documentation

4.3.1.1 diagnostic()

```
def logic_piano.diagnostic (
              nbuffer,
              scoreout,
              vectplot,
              vectnote,
              vactmaxrel,
              energymin,
              fftmin,
              wait = 8 )
```

4.3.1.2 print_correspondence()

```
def logic_piano.print_correspondence (
              index )
```

4.3.1.3 print_folding()

```
def logic_piano.print_folding (
              segment,
              all_data )
```

4.3.2 Variable Documentation

4.3.2.1 activenotes

```
int logic_piano.activenotes = 0
```

4.3.2.2 allowance

```
float logic_piano.allowance = 0.0
```

using the attack to enable / disable the allowance

4.3.2.3 arg1

```
string logic_piano.arg1 = "./scores/" + instrument + "/" + note + "_filefft.out"
```

4.3.2.4 arg2

```
string logic_piano.arg2 = "./scores/" + instrument + "/" + note + "_filefilter.out"
```

4.3.2.5 arg4

```
string logic_piano.arg4 = "./scores/" + instrument + "/" + note + "_integer.out"
```

4.3.2.6 attack

```
float logic_piano.attack = env[i]/envprev*0.5
```

calculating the attack based on the enevelopes at the current chunck and the one before

4.3.2.7 AVenergymin

```
float logic_piano.AVenergymin = maxrfti*stability_threshold
```

4.3.2.8 avfft

```
list logic_piano.avfft = (vectfft[j] + vectfft[j+12] + vectfft[j+19])/3.
```

4.3.2.9 avrfti

```
list logic_piano.avrfti = (vectnote[j] + vectnote[j+12] + data[i,j+19]/170)/3.
```

4.3.2.10 baseline

```
list logic_piano.baseline = [0]*89
```

4.3.2.11 blueidx

```
logic_piano.blueidx = -1
```

4.3.2.12 bluemax

```
list logic_piano.bluemax = 0
```

4.3.2.13 color

```
logic_piano.color
```

4.3.2.14 data

```
logic_piano.data = genfromtxt(arg2)
```

4.3.2.15 datafft

```
logic_piano.datafft = genfromtxt(arg1)
```

4.3.2.16 dummywarn

```
int logic_piano.dummywarn = 0
```

4.3.2.17 elem

```
logic_piano.elem = line.split("\t")
```

4.3.2.18 elemout

```
list logic_piano.elemout = []
```

4.3.2.19 energymin

```
int logic_piano.energymin = bluemax*rtfithreshold
```

Then I iterate againg over the possible notes.

for evaluating the various conditions poses

4.3.2.20 env

```
logic_piano.env = genfromtxt(arg4)
```

4.3.2.21 envprev

```
logic_piano.envprev = 10.0
```

4.3.2.22 exclude

```
int logic_piano.exclude = 1
```

4.3.2.23 exist

```
list logic_piano.exist = [0]*89
```

4.3.2.24 fft_line

```
list logic_piano.fft_line = [ ]
```

4.3.2.25 fftmax

```
list logic_piano.fftmax = 0
```

For each chunk (i) we iterate over the 89 notes (j)

data.T is data transposed meaning the 89 possible notes
I create the vector of relative maximums vectnote and then I check the memory variable to evaluate if increasing or not the stability index exist

4.3.2.26 fftmin

```
int logic_piano.fftmin = fftmax*firstharmth
```

4.3.2.27 firstharmth

```
float logic_piano.firstharmth = 4.3/sens
```

4.3.2.28 freq

```
logic_piano.freq = genfromtxt("./algo_outputs/notes_piano.txt")
```

4.3.2.29 i2monitor

```
int logic_piano.i2monitor = 943
```

4.3.2.30 instrument

```
string logic_piano.instrument = "piano"
```

4.3.2.31 j2monitor

```
int logic_piano.j2monitor = 42
```

4.3.2.32 jvect

```
list logic_piano.jvect = [1.4]*89
```

4.3.2.33 line

```
logic_piano.line = temp[0]
```

4.3.2.34 maxidx

```
logic_piano.maxidx = n.argmax(vectplot)
```

4.3.2.35 maxrfti

```
logic_piano.maxrfti = n.max(vectplot)
```

4.3.2.36 memory

```
list logic_piano.memory = [0]*89
```

4.3.2.37 midiscore

```
list logic_piano.midiscore = []
```

4.3.2.38 names

```
list logic_piano.names = ["A","A#","B","C","C#","D","D#","E","F","F#","G","G#"]
```

4.3.2.39 noise_threshold

```
float logic_piano.noise_threshold = 3.0
```

4.3.2.40 NOpattern

```
bool logic_piano.NOpattern = False
```

4.3.2.41 not_presence

```
list logic_piano.not_presence = [0]*89
```

4.3.2.42 note

```
string logic_piano.note = "While_my_guitar"
```

note and instrument are useful to compose the name of the score file to load.

The scores are generated from the C++ algorithm and are moved to their folder by a script Here we are calling the four scores and extracting the data from them using the python module genfromtxt

4.3.2.43 out_f

```
logic_piano.out_f = open("vectplot_printout.txt", "w")
```

4.3.2.44 out_n

```
logic_piano.out_n = open("output_evaluated_printout.txt", "w")
```

4.3.2.45 out_s

```
logic_piano.out_s = open("midiscore_printout.txt", "w")
```

4.3.2.46 pause

```
float logic_piano.pause = 0.09
```

4.3.2.47 plot_duration

```
int logic_piano.plot_duration = 390
```

4.3.2.48 portion

```
logic_piano.portion = int(len(midiscore)/1.)
```

Portion is used to establish the duration of the plot.

It is defined as a fraction of the duration of the audiofile If we want to analyze a precise number of chunks, we need to insert that integer number

4.3.2.49 redline

```
list logic_piano.redline = [0]*89
```

4.3.2.50 rtfithreshold

```
float logic_piano.rtfithreshold = 4.601162791/sens
```

4.3.2.51 rtfithresholdline

```
list logic_piano.rtfithresholdline = [0]*89
```

4.3.2.52 scarto

```
int logic_piano.scarto = 0
```

4.3.2.53 scndharmth

```
float logic_piano.scndharmth = 3.999/sens
```

4.3.2.54 scndharmthline

```
list logic_piano.scndharmthline = [0]*89
```

4.3.2.55 scoreout

```
logic_piano.scoreout = open("/home/luciamarock/Documents/cpp_tests/PitchDetector/src/integer.←
out", "r")
```

4.3.2.56 sens

```
float logic_piano.sens = 5.0
```

4.3.2.57 stability_threshold

```
float logic_piano.stability_threshold = 3.5 / sens
```

4.3.2.58 temp

```
logic_piano.temp = line.split("\n")
```

4.3.2.59 th_line

```
list logic_piano.th_line = [ ]
```

4.3.2.60 time

```
int logic_piano.time = (i-scarto) * 256 / 44100.  * 1000
```

4.3.2.61 time_from_beginning

```
tuple logic_piano.time_from_beginning = (i+1)/44100.  * 256
```

4.3.2.62 totalenergy

```
tuple logic_piano.totalenergy = avrfti * avfft * 2.
```

4.3.2.63 vect

```
list logic_piano.vect = [0]*89
```

4.3.2.64 vectcond

```
list logic_piano.vectcond = [1.5]*89
```

plot-only variables

Resetting the variables

4.3.2.65 vectfft

```
logic_piano.vectfft = [0]*89
```

4.3.2.66 vectgrey

```
list logic_piano.vectgrey = [0]*89
```

4.3.2.67 vectmono

```
list logic_piano.vectmono = [0]*89
```

4.3.2.68 vectnote

```
list logic_piano.vectnote = [0]*89
```

4.3.2.69 vectplot

```
list logic_piano.vectplot = [0]*89
```

Chapter 5

Class Documentation

5.1 Tuple Struct Reference

```
#include <resonators.h>
```

Public Attributes

- float a [100]
- float b [100]
- float c [100]
- float d [100]

5.1.1 Member Data Documentation

5.1.1.1 a

```
float Tuple::a[100]
```

5.1.1.2 b

```
float Tuple::b[100]
```

5.1.1.3 c

```
float Tuple::c[100]
```

5.1.1.4 d

```
float Tuple::d[100]
```

The documentation for this struct was generated from the following file:

- resonators.h

Chapter 6

File Documentation

6.1 .dep.inc File Reference

6.2 fft_funct.cpp File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "fft_funct.h"
#include "parabola.h"
```

Macros

- #define PI 3.14159265358979323
- #define Ns 2048

Functions

- void CalcolaW (int n)
- void fft_funct (float *buffer, int size, float tet, float f0, int nharms, int Fs)

Variables

- double Wre [Ns/2]
- double Wim [Ns/2]

6.2.1 Macro Definition Documentation

6.2.1.1 Ns

```
#define Ns 2048
```

6.2.1.2 PI

```
#define PI 3.14159265358979323
```

6.2.2 Function Documentation

6.2.2.1 CalcolaW()

```
void CalcolaW (
            int n )
```

6.2.2.2 fft_funct()

```
void fft_funct (
            float * buffer,
            int size,
            float tet,
            float f0,
            int nharms,
            int Fs )
```

In the fft_funct the first thing to do is to calculate the closest power of two to the size of the input vector. Then the new buffer size is called NC (in case the input vector's dimension is already a power of two, then NC is equal to the input vector's length)

Then another vector is calculated, it will be used for the parabolic interpolation and it is called fc[]

The Goertzel algorithm is used to calculate the FFT, this algorithm uses decimation in time dividing the input sequence in blocks until each block contains only to samples to be combined, please read the document Tesina di informatica II.pdf for having a clearer understanding of the procedure

Xre is the rewritten input vector with inverted bit position (see the Goertzel algorithm), Xim is not calculated because in fact we don't use the imaginary part

The coefficient calculation Wre and Wim could have been done at the beginning

The input buffer is overwritten with the amplitude of the fft points

Here the closest peak calculation is performed in order to find the peaks on which the parabolic interpolation should be calculated
For each note (fc[]) we search the Fourier frequency (index[]) closest to the fundamental frequency of that note

Then we pass to the parabola calculation the 3 points around that RTFI frequency and we calculate the parabola coefficients A, B and C
and use those coefficients to calculate the estimated amplitude of the RTFI frequency

Finally we overwrite the Fourier output with the values corrected from the parabolic interpolation.
Indeed we don't take full advantage of the parabolic interpolation which could be used to estimate the correct position in frequency of the point. This is a choice we made because we supposed that we don't need to estimate the correct position in frequency of the peak because we already know it, that's why we use the RTFI technique; but in case of the guitar this parabolic interpolation could be also used for estimating the correct frequency during "bendings" even tough we could be precise only on high pitched notes.
Another thing that should be evaluated is the possibility to analyze a larger buffer, this should be done considering the CPU power we have at our disposal, from this Goertzel implementation documentation we have the complexity of the algorithm which is roughly NlogN

6.2.3 Variable Documentation

6.2.3.1 Wim

```
double Wim[Ns/2]
```

6.2.3.2 Wre

```
double Wre[Ns/2]
```

6.3 fft_funct.h File Reference

Functions

- void fft_funct (float ∗buffer, int size, float tet, float f0, int nharms, int Fs)

6.3.1 Function Documentation

6.3.1.1 fft_funct()

```
void fft_funct (
            float * buffer,
            int size,
            float tet,
            float f0,
            int nharms,
            int Fs )
```

In the fft_funct the first thing to do is to calculate the closest power of two to the size of the input vector. Then the new buffer size is called NC (in case the input vector's dimension is already a power of two, then NC is equal to the input vector's length)

Then another vector is calculated, it will be used for the parabolic interpolation and it is called fc[]

The Goertzel algorithm is used to calculate the FFT, this algorithm uses decimation in time dividing the input sequence in blocks until each block contains only to samples to be combined, please read the document Tesina di informatica II.pdf for having a clearer understanding of the procedure

Xre is the rewritten input vector with inverted bit position (see the Goertzel algorithm), Xim is not calculated because in fact we don't use the imaginary part

The coefficient calculation Wre and Wim could have been done at the beginning

The input buffer is overwritten with the amplitude of the fft points

Here the closest peak calculation is performed in order to find the peaks on which the parabolic interpolation should be calculated
For each note (fc[]) we search the Fourier frequency (index[]) closest to the fundamental frequency of that note

Then we pass to the parabola calculation the 3 points around that RTFI frequency and we calculate the parabola coefficients A, B and C
and use those coefficients to calculate the estimated amplitude of the RTFI frequency

Finally we overwrite the Fourier output with the values corrected from the parabolic interpolation.
Indeed we don't take full advantage of the parabolic interpolation which could be used to estimate the correct position in frequency of the point. This is a choice we made because we supposed that we don't need to estimate the correct position in frequency of the peak because we already know it, that's why we use the RTFI technique; but in case of the guitar this parabolic interpolation could be also used for estimating the correct frequency during "bendings" even tough we could be precise only on high pitched notes.
Another thing that should be evaluated is the possibility to analyze a larger buffer, this should be done considering the CPU power we have at our disposal, from this Goertzel implementation documentation we have the complexity of the algorithm which is roughly NlogN

6.4 frammenti_prev.py File Reference

Namespaces

- frammenti_prev

Functions

- def frammenti_prev.rotate (average, new_value)

Variables

- string frammenti_prev.instrument = "piano"
- string frammenti_prev.note = "Beat_It_cut"
- float frammenti_prev.MOVING_AVARAGE_P = 0.1
- float frammenti_prev.ATTACK_FACTOR = 0.6
- list frammenti_prev.average = [0.] ∗ 20
- list frammenti_prev.moving_avg = [ ]
- list frammenti_prev.moving_avg_mia = [ ]
- list frammenti_prev.threshold = [ ]
- float frammenti_prev.value = 0.05
- list frammenti_prev.real_signal = [ ]
- float frammenti_prev.p = MOVING_AVARAGE_P;
- frammenti_prev.value_mio
- frammenti_prev.th = pow(real_signal[i],2) / value_mio
- frammenti_prev.color

6.5 logic_guit.py File Reference

Namespaces

- logic_guit

Variables

- string logic_guit.nota = "Amag_open"

  nota and chitarra are useful to compose the name of the score file to load.
- string logic_guit.chitarra = "poly"
- string logic_guit.arg1 = "./scores/" + chitarra + "/" + nota + "_filefft.out"
- string logic_guit.arg2 = "./scores/" + chitarra + "/" + nota + "_filefilter.out"
- string logic_guit.arg3 = "./scores/" + chitarra + "/" + nota + "_monodet.out"
- string logic_guit.arg4 = "./scores/" + chitarra + "/" + nota + "_integer.out"
- logic_guit.datafft = genfromtxt(arg1)
- logic_guit.data = genfromtxt(arg2)
- logic_guit.fund = genfromtxt(arg3)
- logic_guit.env = genfromtxt(arg4)
- float logic_guit.envprev = 10.0
- logic_guit.freq = genfromtxt("./algo_outputs/notes.txt")
- int logic_guit.existsum = 0
- float logic_guit.sens = 0.1
- float logic_guit.Pshp = 1.03
- float logic_guit.Pshs = 1.025
- float logic_guit.Pthp = 0.6∗sens
- float logic_guit.Pths = 0.7∗sens
- list logic_guit.vectmono = [0]∗51
- list logic_guit.vectnote = [0]∗51

- list logic_guit.memoria = [0]∗51
- list logic_guit.exist = [0]∗51
- float logic_guit.allowance = 0.0

    using th attack to enable / disable the allowance
- logic_guit.f = open("./algo_outputs/names.txt","r")
- logic_guit.labels = f.readlines()
- list logic_guit.names = [ ]
- logic_guit.name = str(labels[k]).split('\n')
- float logic_guit.pause = 0.09
- int logic_guit.condizione = 0

    For each chunk (i) we iterate over the 51 notes (j)
- int logic_guit.dummywarn = 0
- list logic_guit.vectcond = [1.5]∗51
- list logic_guit.jvect = [1.4]∗51
- list logic_guit.vectplot = [0]∗51
- list logic_guit.vectfft = [0]∗51
- list logic_guit.vect = [0]∗51
- list logic_guit.existplot = [0]∗51
- list logic_guit.baseline = [0]∗51
- logic_guit.portion = int(len(data)/4)

    Portion is used to establish the duration of the plot.
- int logic_guit.scarto = 0
- tuple logic_guit.time = (i-scarto) ∗ 256 / 44100. ∗ 1000
- tuple logic_guit.ratio2 = (datafft[i,j+12]+datafft[i,j+19])∗0.5/datafft[i,j-7]
- logic_guit.ratio1 = datafft[i,j+12]/datafft[i,j-7]
- float logic_guit.attack = env[i]/envprev∗0.5

    calculating the attack based on the enevelopes at the current chunck and the one before
- logic_guit.minimo = min(vect)

    plot-only variables
- logic_guit.color

6.6 logic_piano.py File Reference

Namespaces

- logic_piano

Functions

- def logic_piano.diagnostic (nbuffer, scoreout, vectplot, vectnote, vactmaxrel, energymin, fftmin, wait=8)
- def logic_piano.print_folding (segment, all_data)
- def logic_piano.print_correspondence (index)

Variables

- string logic_piano.note = "While_my_guitar"

  note and instrument are useful to compose the name of the score file to load.
- list logic_piano.names = ["A","A#","B","C","C#","D","D#","E","F","F#","G","G#"]
- string logic_piano.instrument = "piano"
- logic_piano.freq = genfromtxt("./algo_outputs/notes_piano.txt")
- string logic_piano.arg1 = "./scores/" + instrument + "/" + note + "_filefft.out"
- string logic_piano.arg2 = "./scores/" + instrument + "/" + note + "_filefilter.out"
- string logic_piano.arg4 = "./scores/" + instrument + "/" + note + "_integer.out"
- logic_piano.datafft = genfromtxt(arg1)
- logic_piano.data = genfromtxt(arg2)
- logic_piano.env = genfromtxt(arg4)
- logic_piano.scoreout = open("/home/luciamarock/Documents/cpp_tests/PitchDetector/src/integer.out", "r")
- list logic_piano.midiscore = [ ]
- logic_piano.temp = line.split("\n")
- logic_piano.line = temp[0]
- list logic_piano.elemout = [ ]
- logic_piano.elem = line.split("\t")
- float logic_piano.envprev = 10.0
- float logic_piano.noise_threshold = 3.0
- float logic_piano.sens = 5.0
- float logic_piano.firstharmth = 4.3/sens
- float logic_piano.scndharmth = 3.999/sens
- float logic_piano.rtfithreshold = 4.601162791/sens
- float logic_piano.stability_threshold = 3.5 / sens
- list logic_piano.vectgrey = [0]∗89
- list logic_piano.vectnote = [0]∗89
- list logic_piano.vectplot = [0]∗89
- list logic_piano.memory = [0]∗89
- list logic_piano.exist = [0]∗89
- list logic_piano.not_presence = [0]∗89
- float logic_piano.allowance = 0.0

  using the attack to enable / disable the allowance
- int logic_piano.activenotes = 0
- float logic_piano.pause = 0.09
- int logic_piano.dummywarn = 0
- list logic_piano.jvect = [1.4]∗89
- list logic_piano.vectfft = [0]∗89
- list logic_piano.vect = [0]∗89
- list logic_piano.baseline = [0]∗89
- list logic_piano.redline = [0]∗89
- list logic_piano.scndharmthline = [0]∗89
- list logic_piano.rtfithresholdline = [0]∗89
- int logic_piano.exclude = 1
- int logic_piano.i2monitor = 943
- int logic_piano.j2monitor = 42
- logic_piano.out_f = open("vectplot_printout.txt", "w")
- logic_piano.out_n = open("output_evaluated_printout.txt", "w")
- logic_piano.out_s = open("midiscore_printout.txt", "w")
- logic_piano.portion = int(len(midiscore)/1.)

  Portion is used to establish the duration of the plot.
- int logic_piano.plot_duration = 390
- int logic_piano.scarto = 0

- int logic_piano.bluemax = 0
- int logic_piano.blueidx = -1
- int logic_piano.fftmax = 0

  For each chunk (i) we iterate over the 89 notes (j)
- tuple logic_piano.time = (i-scarto) ∗ 256 / 44100. ∗ 1000
- int logic_piano.energymin = bluemax∗rtfithreshold

  Then I iterate againg over the possible notes.
- int logic_piano.fftmin = fftmax∗firstharmth
- bool logic_piano.NOpattern = False
- tuple logic_piano.avfft = (vectfft[j] + vectfft[j+12] + vectfft[j+19])/3.
- tuple logic_piano.avrfti = (vectnote[j] + vectnote[j+12] + data[i,j+19]/170)/3.
- tuple logic_piano.totalenergy = avrfti ∗ avfft ∗ 2.
- logic_piano.maxrfti = n.max(vectplot)
- logic_piano.maxidx = n.argmax(vectplot)
- float logic_piano.AVenergymin = maxrfti∗stability_threshold
- float logic_piano.attack = env[i]/envprev∗0.5

  calculating the attack based on the enevelopes at the current chunck and the one before
- tuple logic_piano.time_from_beginning = (i+1)/44100. ∗ 256
- logic_piano.color
- list logic_piano.th_line = [ ]
- list logic_piano.fft_line = [ ]
- list logic_piano.vectcond = [1.5]∗89

  plot-only variables
- list logic_piano.vectmono = [0]∗89

6.7 main.cpp File Reference

Description of the main program.

```
#include <stdio.h>
#include <stdlib.h>
#include <sndfile.h>
#include <unistd.h>
#include "streamer.h"
#include "resonators.h"
#include <math.h>
#include "fft_funct.h"
#include <fftw3.h>
```

Macros

- #define N 512

Functions

- int main ()

6.7.1 Detailed Description

Description of the main program.

Envelope Calculation:

- The loop calculates the envelope of the current chunk. The envelope is a smooth curve that captures the shape of the audio waveform. It's often used to emphasize the overall shape or magnitude of a signal.
- The calculated envelope value is used in subsequent calculations and is written to an output file (`integer.out`).

FFT (Fast Fourier Transform):

- The loop performs FFT on the current chunk using the `fft_funct` function. FFT is a mathematical algorithm that transforms a time-domain signal (in this case, an audio chunk) into its frequency-domain representation.
- The FFT result is stored in the `myfft_buffer` array for further analysis and is also used to calculate an averaged value, which is written to an output file (`filefft.out`).

RTFI Calculation:

- The loop calculates the Relative Temporal Feature Intensity (RTFI) for each RTFI note within the current chunk.
- It iterates over the samples in the chunk and performs calculations involving coefficients, input audio data (`buf`), and intermediate arrays (`rey`, `imgy`, `pry`, `iry`, `energy`).

Monophonic Detection:

- The loop performs monophonic detection on the current chunk. Monophonic detection involves identifying the dominant or primary note being played in the audio signal. The loop iterates over a range of possible note frequencies and calculates a "capture" value for each note's wavelength.
- The note frequency that produces the maximum capture value is considered the detected monophonic note for the chunk. The detected note frequency is recorded in an output file (`monodet.out`).
- The calculated RTFI values are written to an output file (`filefilter.out`).

Author

luciamarock

Date

August 7th, 2023

6.7.2 Macro Definition Documentation

6.7.2.1 N

```
#define N 512
```

6.7.3 Function Documentation

6.7.3.1 main()

```
int main ( )
```

sf is the pointer to the .wav file, library sndfile is used to open the WAV file.

coefficients is a tuple returned by the init function of resonators.cpp the tuple contains 3 coefficients for calculating the RTFI resonators and the list of notes which are the center frequency of the resonators

The samples[] vector contains the number of samples that describe an entire wavelength associated with each RTFI note at the specific Sampling Frequency, this is used later on in the mono detection algorithm

nchunks is the number of chunks contained in the read audio file and it is received from function chunker() in module streamer.cpp
It bases on the sampvect which is hard-coded to be 256 (N/2) samples

Iterating on the input buffer (of dimension 256 samples) in order to build the vectors to be analyzed, the real data are contained in the vector buf which represent the entire audio file; the samples of the buf vector are always scaled by a factor of 2147483648 which is hard-coded
The vector container[] is made of the concatenation of three consecutive buffers and it is used for the mono detection; its length is designed taking into account the maximum wavelength of the guitar which is at about 82 Hz
Vector myfft_buffer[] is used for the computation of the FFT with my algorithm, it is composed by the last 4 buffers thus bringing the total amount of he vector equal to 1024 samples, so the fft is calculated on this vector. The fft is not really used to find the peaks, in fact we should have a calculation vector of 8192 to discriminate for real the partials at lowest frequencies, but it still can be relied on to find the correct amplitudes to the partials since the RTFIs take time to reflect the real amplitude of the partials, in the future the fft with parabolic interpolation can also be used for the tracking of bendings on th guitar.
Finally integer is used to calculate the envelope of the buffers, it gets printed on a file called integer.out

Next the fft_funct() is called which calculates the fft on the same vector that is passed to

We then consider an averaged value with the former fft calculation and we print the values to the file filefft.out

In the same loop of the RTFI calculation we perform also the monophonic detection.
The loop is on the RTFI frequencies and for the monophonic detection there is another sub-loop that searches inside the container[] vector summing up the 3 samples at distance samples[k] which is the wavelength (in samples) of the current note (RTFI frequency). For each note the maximum sum is memorized and then it is registered if superior of any other previous maximum. At the en of the main loop the wavelength which gave he maximum sum remains registered so we can state that was the monophonic note which was detected. This algorithm might suffer spurious spikes in the waveform and it works with clean waveforms at best and with a rapid dacay

The RTFI calculation is performed and the values are written in the file filefilter.out
The monophonic detection instead is printed on the file monodet.out

6.8 parabola.cpp File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "parabola.h"
```

Functions

- void parabola (float &A, float &B, float &C, float x0, float y0, float x1, float y1, float x2, float y2)

6.8.1 Function Documentation

6.8.1.1 parabola()

```
void parabola (
            float & A,
            float & B,
            float & C,
            float x0,
            float y0,
            float x1,
            float y1,
            float x2,
            float y2 )
```

Given three points, it calculates the coefficients of the parabola passing through those points

6.9 parabola.h File Reference

Functions

- void parabola (float &A, float &B, float &C, float x0, float y0, float x1, float y1, float x2, float y2)

6.9.1 Function Documentation

6.9.1.1 parabola()

```
void parabola (
            float & A,
            float & B,
            float & C,
            float x0,
            float y0,
            float x1,
            float y1,
            float x2,
            float y2 )
```

Given three points, it calculates the coefficients of the parabola passing through those points

6.10 resonators.cpp File Reference

```
#include <cstdlib>
#include <stdio.h>
#include <iostream>
#include <math.h>
#include <vector>
#include <fstream>
#include <iomanip>
#include "resonators.h"
#include <fftw3.h>
```

Functions

- struct Tuple init (int Fs, float f0, int nnotes, float tet)

6.10.1 Function Documentation

6.10.1.1 init()

```
struct Tuple init (
            int Fs,
            float f0,
            int nnotes,
            float tet )
```

This init function is called directly from the main() program at the beginning it loops until nharms (which in this case is 70) in order to calculate the center frequencies for the RTFI resonators, the f0 is also hard coded and it's set to 77.782 Hz for the Guitar. The notes vector is returned as coefficent d All of these parameters should be read from a configuration file instead of being hard-coded

Finally the coefficients for the RTFI resonators are calculated, no mistery in this block just a bit of math.

6.11 resonators.h File Reference

Classes

- struct Tuple

Functions

- struct Tuple init (int Fs, float f0, int nnotes, float tet)

6.11.1 Function Documentation

6.11.1.1 init()

```
struct Tuple init (
            int Fs,
            float f0,
            int nnotes,
            float tet )
```

This init function is called directly from the main() program at the beginning it loops until nharms (which in this case is 70) in order to calculate the center frequencies for the RTFI resonators, the f0 is also hard coded and it's set to 77.782 Hz for the Guitar. The notes vector is returned as coefficent d All of these parameters should be read from a configuration file instead of being hard-coded

Finally the coefficients for the RTFI resonators are calculated, no mistery in this block just a bit of math.

6.12 somma.cpp File Reference

```
#include <iostream>
#include "somma.h"
```

Functions

- int addition (int primo, int secondo)

6.12.1 Function Documentation

6.12.1.1 addition()

```
int addition (
            int primo,
            int secondo )
```

6.13 somma.h File Reference

Functions

- int addition (int primo, int secondo)

6.13.1 Function Documentation

6.13.1.1 addition()

```
int addition (
            int primo,
            int secondo )
```

6.14 streamer.cpp File Reference

```
#include <iostream>
#include "streamer.h"
```

Functions

- int chunker (int buffdim, int elemcount)

6.14.1 Function Documentation

6.14.1.1 chunker()

```
int chunker (
            int buffdim,
            int elemcount )
```

This function basically divides the total length (in number of samples) of the audio file in input by the length of the chosen samples buffer, in this case 256, this way we obtain the number of buffers contained in the audio file.
There is of course a bunch of remaining samples this can be ignored, in fact they are not returned, the calculation was made just for educational purposes because it is done calling another function

6.15 streamer.h File Reference

Functions

- int chunker (int buffdim, int elemcount)

6.15.1 Function Documentation

6.15.1.1 chunker()

```
int chunker (
            int buffdim,
            int elemcount )
```

This function basically divides the total length (in number of samples) of the audio file in input by the length of the chosen samples buffer, in this case 256, this way we obtain the number of buffers contained in the audio file.
There is of course a bunch of remaining samples this can be ignored, in fact they are not returned, the calculation was made just for educational purposes because it is done calling another function

Index