

Detecció de pitch

Processat d'audio i veu

Pràctica 3

Daniel Navarrete Jimenez

Grup 41

10/11/2019

Introducció

La pràctica 3 tracta d'entendre i desenvolupar un codi amb el que estudiar i determinar el pitch de les diferents trames dels fitxers d'àudio d'una base de dades donada. Aquest sistema es fa a partir de la tècnica de correlació, tenint en compte la correlació en 1 normalitzada, la màxima i la potència. També determina si el só en qüestió és sonor o sord. La pràctica està desenvolupada amb C++ i fem ús de les llibreries estàndard a més de la llibreria cmath.

Com a afegit, continuem utilitzant les eines de git per portar un control de les versions del projecte, ús de meson/ninja per al manteniment i afegim el doxygen amb el que realitzem un document de seguiment del projecte.

Estructura del proyecto

Primer de tot obtenim el directori de la pràctica desde github. Clonem amb la comana git clone url-directori. A partir d'aquí creem la nostra propia branca amb git branche i ens desplacem amb git checkout NOMBRANCA. Quant volguem pujar alguna actualització al github clonat, fem el mateix procediment de sempre amb les comanes git add . per tal d'afegir el directori al control de versions, git commint -a -m 'Nom de la versió' per actualitzar – ho i finalment, per pujar – ho a github i actualitzar la versió, fem git push. Ara veiem com s'ha actualitzat la branca, en el meu cas, Navarrete-Daniel.

```
nada para hacer commit, el árbol de trabajo esta limpio
daniel@daniel-HP-Pavilion-15-Notebook-PC:~/Documentos/P3pav/p3$ git checkout Navarrete-Daniel
Revisando archivos: 100% (1389/1389), listo.
Cambiado a rama 'Navarrete-Daniel'
Tu rama está adelantada a 'origin/Navarrete-Daniel' por 1 commit.
(usa "git push" para publicar tus commits locales)
daniel@daniel-HP-Pavilion-15-Notebook-PC:~/Documentos/P3pav/p3$ git status
En la rama Navarrete-Daniel
Tu rama está adelantada a 'origin/Navarrete-Daniel' por 1 commit.
(usa "git push" para publicar tus commits locales)

nada para hacer commit, el árbol de trabajo esta limpio
daniel@daniel-HP-Pavilion-15-Notebook-PC:~/Documentos/P3pav/p3$ git push
Username for 'https://github.com': danileRond
Password for 'https://danileRond@github.com':
Contando objetos: 77, listo.
Delta compression using up to 4 threads.
Comprimiendo objetos: 100% (75/75), listo.
Escribiendo objetos: 100% (77/77), 244.26 KiB | 842.00 KiB/s, listo.
Total 77 (delta 18), reused 0 (delta 0)
remote: Resolving deltas: 100% (18/18), completed with 18 local objects.
To https://github.com/DanileRond/p3.git
   0d71c6d..9ee693d  Navarrete-Daniel -> Navarrete-Daniel
daniel@daniel-HP-Pavilion-15-Notebook-PC:~/Documentos/P3pav/p3$
```

🔗 Navarrete-Daniel (less than a minute ago)		🔗 Compare & pull request
Branch: Navarrete-Dani...	New pull request	Create new file Upload files Find file Clone or download
This branch is 3 commits ahead, 1 commit behind master.		🔗 Pull request 📄 Compare
DanileRond Versió 10N 2.21		Latest commit 9ee693d 9 minutes ago
bin	Versió 10N 2.21	9 minutes ago
html	Versió 10N 2.21	9 minutes ago
latex	Versió 9/N 21.15	5 hours ago
pitch_db	Versió 10N 2.21	9 minutes ago
scripts	Versió 10N 2.21	9 minutes ago
src	Versió 10N 2.21	9 minutes ago
.gitignore	Ficheros iniciales de la práctica P3	13 days ago
Doxyfile	Versió 9/N 21.15	5 hours ago
README.md	Ficheros iniciales de la práctica P3	13 days ago
meson.build	Versió 9/N 21.15	5 hours ago
pav_4111.f0	Versió 10N 2.21	9 minutes ago
pav_4111.f0ref	Versió 10N 2.21	9 minutes ago
pav_4111.wav	Versió 9/N 21.15	5 hours ago

Para todo ello, previamente hemos tenido que crear nuestro propio usuario de GitHub y crear nuestro propio repositorio a partir del directorio clonado e importado.

Llenguatge de programació

Tal i com hem dit a la introducció, el llenguatge emprat per al desenvolupament d'aquesta pràctica és C++. Aquest llenguatge està basat en els conceptes de la programació orientada a objectes

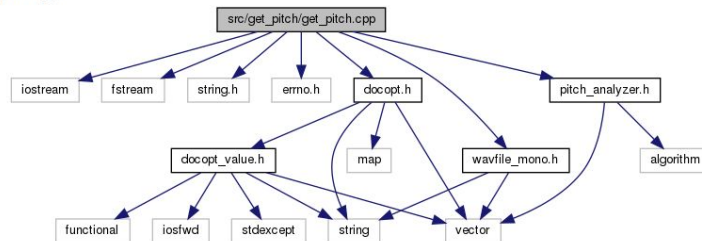
Generació de la documentació

Amb l'objectiu de fer un seguiment i document endreçat del projecte simultani al desenvolupament d'aquest, fem ús del programa doxygen. D'acord amb l'enunciat i les instruccions, finalment obtenim uns arxius HTML i LaTeX amb el contingut. En ells es mostra els elements que formen el projecte i com es relacionen entre ells.

get_pitch.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <errno.h>
#include "wavfile_mono.h"
#include "pitch_analyzer.h"
#include "docopt.h"
```

Include dependency graph for get_pitch.cpp:



Macros

```
#define FRAME_LEN 0.030 /* 30 ms. */
#define FRAME_SHIFT 0.015 /* 15 ms. */
```

Práctica 3 de PAV - detección de pitch v2.0

Postprocess the estimation in order to suppress errors. For instance, a median filter or time-warping may be used.

Member `upc::PitchAnalyzer::autocorrelation` (`const std::vector< float > &x`, `std::vector< float > &r`) `const`

[HECHO]:
Compute the autocorrelation `r[i]`

Member `upc::PitchAnalyzer::compute_pitch` (`std::vector< float > &x`) `const`

[HECHO]:
Find the lag of the maximum value of the autocorrelation away from the origin.
Choices to set the minimum value of the lag are:

- The first negative value of the autocorrelation.
- The lag corresponding to the maximum value of the pitch.

In either case, the lag should not exceed that of the minimum value of the pitch.

Member `upc::PitchAnalyzer::set_window` (`Window` type)

[HECHO]:
Implement the Hamming window

Member `upc::PitchAnalyzer::unvoiced` (`float pot`, `float r1norm`, `float rmaxnorm`) `const`

[HECHO]:
Implement a rule to decide whether the sound is voiced or not.

- You can use the standard features (`pot`, `r1norm`, `rmaxnorm`), or compute and use other ones.

Generated by **doxygen** 1.8.13

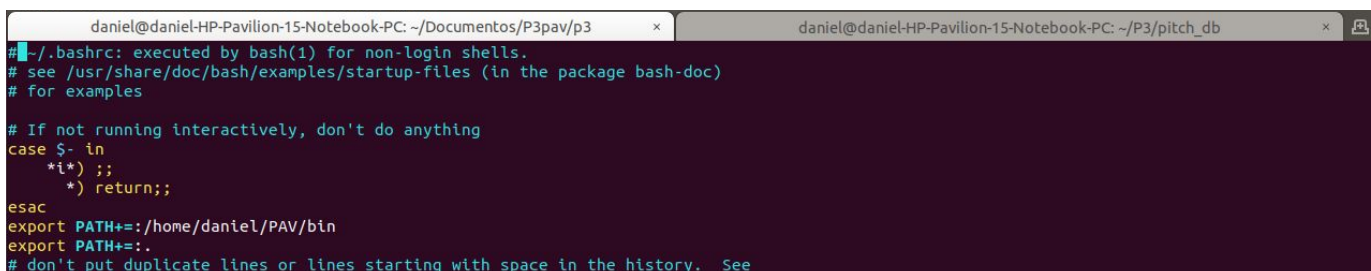
Per executar i generar aquests fitxers fem una crida al programa doxygen. El mode bàsic per fer-ho és fer la comanda doxygen -g al directori, desde el terminal.

També, com veiem en la segona imatge d'aquest apartat, podem gestionar les tasques necessàries a realitzar (TODO) i les ja fetes (HECHO) a partir de les corresponents comandes. Amb \TODO o /HECHO.

Construcció del detector de pitch: get_pitch

Al programa que s'executa se li ha de passar com arguments l'arxiu WAVE i el fitxer .f0 on s'emmagatzema el resultat del pitch de cada frame.

Per tal de facilitar la crida `bin/get_pitch arx.wav fitx.f0`, i com que aquesta es farà repetides vegades per a una extensa base de dades, tindrem uns scripts que s'encarregaran d'aixó. En aquest cas `run_get_pitch.sh`. I en la mateixa linea, per a facilitar-ho encara més i no haver de senyalar a cada vegada els directoris afegim a la variable PATH del fitxer `~/bashrc` la ruta del directori del programa.



```
daniel@daniel-HP-Pavilion-15-Notebook-PC: ~/Documentos/P3pav/p3
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
  *) ;;
  *) return;;
esac
export PATH+=:/home/daniel/PAV/bin
export PATH+=:
# don't put duplicate lines or lines starting with space in the history. See
```

En el meu cas, el directori dels executables està a `daniel/PAV`. Guardem i sortim del editor vi.

Aixó ho hem fet amb el mecanisme que ens proporciona `meson/ninja: install`. Executem `ninja install -C bin` i tots els programes marcats al `meson.build` amb l'opció `install: true`, es copiaran al directori dessitjat.

```
project(
  'Práctica 3 de PAV - detección de pitch', 'cpp',
  default_options: ['prefix=/home/daniel/PAV', 'libdir=lib'],
  version : 'v2.0'
)

inc = include_directories(['src/include', 'src/docopt_cpp'])
```

Efectivament, cridem els executables i responen de la manera esperada.

```
daniel@daniel-HP-Pavilion-15-Notebook-PC:~/Documentos/P3pav/p3$ run_get_pitch
pitch_db/train2/rl002.wav ----
pitch_db/train2/rl004.wav ----
pitch_db/train2/rl006.wav ----
pitch_db/train2/rl008.wav ----
pitch_db/train2/rl010.wav ----
pitch_db/train2/rl012.wav ----
pitch_db/train2/rl014.wav ----
pitch_db/train2/rl016.wav ----
pitch_db/train2/rl018.wav ----
pitch_db/train2/rl020.wav ----
pitch_db/train2/rl022.wav ----
pitch_db/train2/rl024.wav ----
pitch_db/train2/rl026.wav ----
pitch_db/train2/rl028.wav ----
[]
```

```
daniel@daniel-HP-Pavilion-15-Notebook-PC:~/Documentos/P3pav/p3$ pitch_evaluate pitch_db/train2/sb*.f0ref
### Compare pitch_db/train2/sb002.f0ref and pitch_db/train2/sb002.f0
Num. frames: 200 = 130 unvoiced + 70 voiced
Unvoiced frames as voiced: 2/130 (1.54 %)
Voiced frames as unvoiced: 10/70 (14.29 %)
Gross voiced errors (+20.00 %): 17/60 (28.33 %)
MSE of fine errors: 2.01 %

==> pitch_db/train2/sb002.f0: 86.11 %
-----
```

Exercicis i entrega

Completem el codi dels fitxers demanats per a la realització de la detecció del pitch.

Càlcul de l'autocorrelació

```
void PitchAnalyzer::autocorrelation(const vector<float> &x, vector<float> &r) const {
    for (unsigned int l = 0; l < r.size(); ++l) {
        /// \HECHO Compute the autocorrelation r[l]
        r[l] = 0;
        for(unsigned int k = l; k < x.size(); ++k){
            r[l] = r[l] + (x[k])*(x[k-l]);
        }
        r[l] = r[l]/r.size();
    }

    if (r[0] == 0.0F) //to avoid log() and divide zero
        r[0] = 1e-10;
}
```

Partim de la defició de l'autocorrelació d'un senyal discret:

$$r[k] = \frac{1}{N} \cdot \sum_{n=0}^{N-1-k} xw[n] \cdot xw[n+k]$$

Finestra Hamming

A continuació, plementem la finestra de Hamming, també segons la definició.

$$w[n] = a_0 - \underbrace{(1 - a_0)}_{a_1} \cdot \cos\left(\frac{2\pi n}{N}\right), \quad 0 \leq n \leq N,$$

On el coeficient a_0 està definit com $\frac{25}{46}$ per a la finestra de Hamming proposada per Richard W. Hamming.

Amb això, implementem el següent codi amb l'ajuda de la llibreria math.h

```
void PitchAnalyzer::set_window(Window win_type) {
    if (frameLen == 0)
        return;

    window.resize(frameLen);
    double c = 25/46;
    switch (win_type) {
        case HAMMING:
            /// \HECHO Implement the Hamming window

            for(unsigned int i = 0; i < frameLen; i++){
                window[i] = c - (1-c) * cos(2 * i * M_PI/(frameLen));
            }

            break;
        case RECT:
        default:
            window.assign(frameLen, 1);
    }
}
```

Determinar el millor candidat per a període pitch.

A partir del primer màxim trobat sense comptar l'origen de l'autocorrelació.

```
vector<float>::const_iterator iR = r.begin(), iRMax = iR + npitch_min, iRref;

for(iRref = iR + npitch_min; iRref < iR + npitch_max; iRref++) {
    if(*iRref > *iRMax) {
        iRMax = iRref;
    }
}
```

Mitjançant comparació, passem per cada un dels valors del pitch de cada frame i ens quedem amb el major.

Criteri de desició per decidir si un so es sonor o sord

Per tal de pensar un criteri que pugui determinar si un so és sord o sonor, podem evaluar diferents paràmetres. Podem fer-ho a partir de la relació del $R[1 \text{ o } n\text{Pitch}]/R[0]$ o $R[0]$, la potència. Per a decidir els llindars, ho hem fet amb prova i error amb la base de dades donada.

El codi implementat és el següent

```
bool PitchAnalyzer::unvoiced(float pot, float rlnorm, float rmaxnorm) const {
    /// \HECHO Implement a rule to decide whether the sound is voiced or not.
    /// * You can use the standard features (pot, rlnorm, rmaxnorm),
    /// or compute and use other ones.
    if(rlnorm < 0.9 || rmaxnorm < 0.2 || pot < -38){
        return true;}
    else return false;
}
```

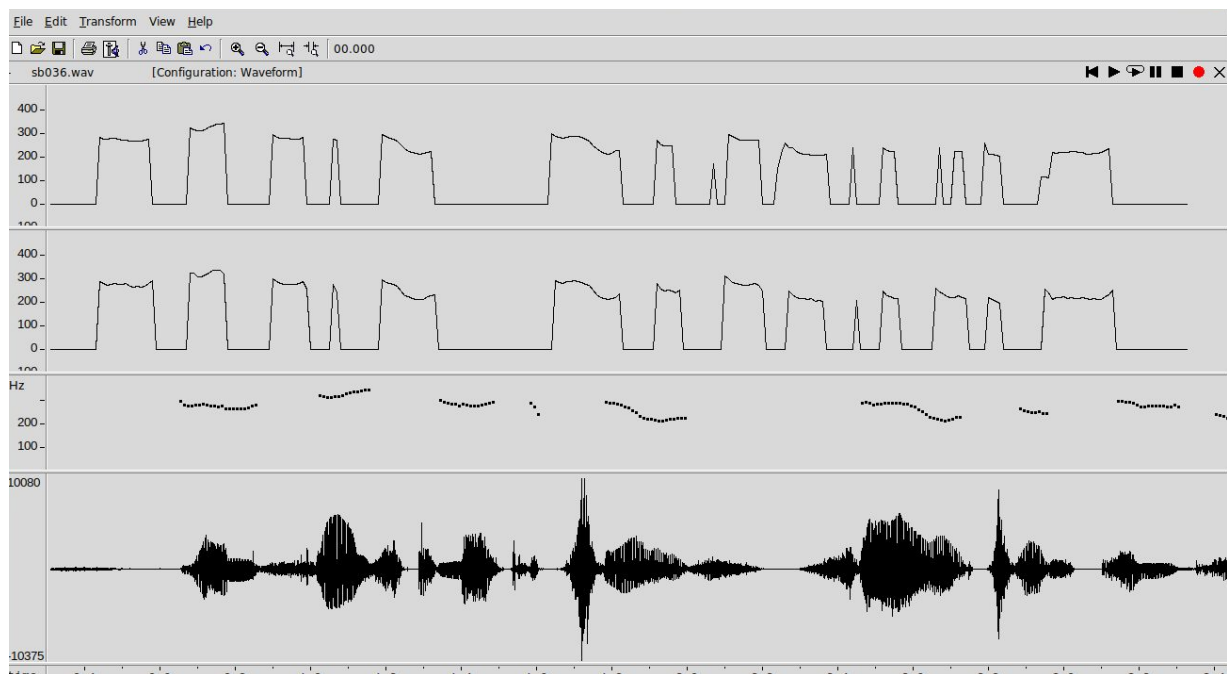
Fent servir el programa wavesurfer, analitzem les condicions per determinar si un segment és sonor o sord. Sabem que és bo el reconeixement a partir de la relació $R[1 \text{ o } n\text{Pitch}]/R[0]$, així com la potència. A Wavesurfer tenim l'opció de veure la potència del senyal en els diferents punts i també la freq de pitch de les trames sonores del mateix senyal. Observant, veiem que la potència mínima aproximada per a la que hi ha un senyal sonor és -38 dB. És a partir d'aquesta magnitud que, juntament amb les altres relacions que determinen el senyal sonor o sord, la classifiquem.

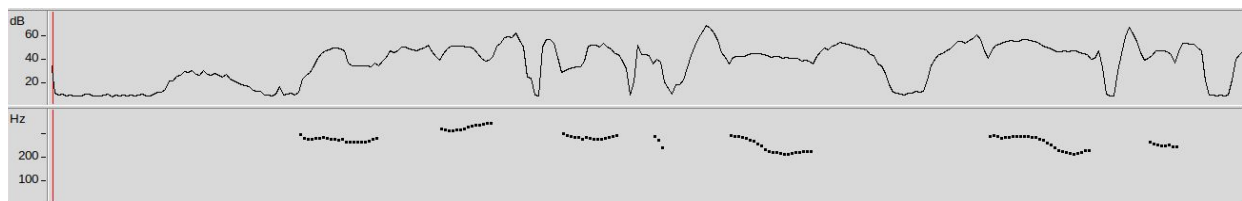
D'aquests altres parametres a banda de la potència, als quals no podem accedir per wavesurfer, els representem mitjançant el següent codi, printats en un fitxer de text anomenat autoR

```
108 //change to #if 1 and compile
109 FILE * correlationfile = fopen("autoR","a");
110 #if 1
111 if (r[0] > 0.0F)
112     cout << pot << '\t' << r[1]/r[0] << '\t' << r[lag]/r[0] << endl;
113     fprintf(correlationfile,"%f\t%f\t%f\n",pot,r[1]/r[0],r[lag]/r[0]);
114 #endif
115 fclose(correlationfile);
```

De manera semblant a com vam fer a la pràctica 1.

A continuació fem servir el detector de pitch amb wavesurfer sobre qualsevol senyal de mostra i comparem el resultat amb el obtingut per el codi propi. A més comparem amb el fitxer de referència.





El

resultat, en comparació amb el fitxer de referència, te una bona similitud. En aquest cas ja que és una senyal de veu femenina. En un model de veu masculina, empitjora considerablement.

També, en comparació amb el pitch trobat per wavesurfer, observem bona similitud (cal negligir el desfase de les mostres del pitch)

Exercicis d'ampliació

Implementem determinades tècniques per a la optimització del sistema de detecció de pitch.

Tècniques de preprocessat

La tècnica que fem servir en aquest cas ha estat el center clipping.

La tècnica consisteix a declarar com a nuls o valor '0', tots els valors petits del senyal per tal d'eliminar el soroll y quedar-nos amb els sons sonors. Procedim així sota del 20% i ho restem al senyal original per aconseguir el resultat desitjat.

```

//Center Clipping
float max_x = 0.0;
unsigned int i;
//Search for the maximum value
for( i = 0; i < x.size(); i++){
    if(x[i] > max_x)
        max_x = x[i];
}
float center_x = 0.015*max_x;
//Apply center clipping
for ( i = 0; i < x.size(); i++){
    x[i] = x[i] / max_x; //Normalizamos

    if( x[i] > center_x){
        x[i] = x[i] - center_x;}

    else if(x[i] < -center_x){
        x[i] = x[i] + center_x;
    }
    else x[i] = 0;
}

```

Tècniques de postprocessat

La tècnica emprada per al postprocessat, en aquest cas, serà l'estudiada a classe: El filtre de mitjana.

Aquesta tècnica elimina errors de detecció de sonoritat. D'una sèrie de valors (3) es queda amb el central. El codi implementat per a aquesta part és el següent:

```
std::vector<float> aux(f0);
unsigned int j = 0;
float maximo,minimo;

for(j = 2; j < aux.size() - 1; ++j) {
    minimo = min(min(aux[j-1], aux[j]), aux[j+1]);
    maximo = max(max(aux[j-1], aux[j]), aux[j+1]);
    f0[j] = aux[j-1] + aux[j] + aux[j+1] - minimo - maximo;
}
```

Conclusions

Per concloure, hem aconseguit, amb èxit desenvolupar els exercicis obligatoris. A més de la primera part dels exercicis optatius, en les que volem estudiar els resultats obtinguts.

Generant els fitxers f0 amb cada un dels arxius WAVE de la base de dades 'pitch_db' a partir del nostre programa i posteriorment evaluant-lo, aconseguim un fscore del 89% tal i com veiem a la imatge,

```
### Summary
Num. frames:      11200 = 7045 unvoiced + 4155 voiced
Unvoiced frames as voiced:      381/7045 (5.41 %)
Voiced frames as unvoiced:      398/4155 (9.58 %)
Gross voiced errors (+20.00 %): 95/3757 (2.53 %)
MSE of fine errors:      2.36 %

==> TOTAL: 89.87 %
-----
daniel@daniel-HP-Pavilion-15-Notebook-PC:~/Documentos/P3pav/p3$
```

on observem també el percentatge d'MSE al 2.36%

Cal diferenciar també les mostres de veu masculines de les femenines, éssent, en el segon cas, superiors en qüestions de resultats al primer cas. Tot i així, la diferència és bastant petita un cop realitzat les tècniques de preprocessat. Amb la tècnica de center-clipping millorem aprop d'un 10% el total del fscore mentre que amb el postprocessat (filtre de mitjana) ens mantenim per sota del 89%, tal i com ja teniem amb la primera tècnica.

Els resultats són bastant positius comparant amb els fitxers de referència i, un cop extret el pitch de les trames sonores amb wavesurfer, també molt semblant a aquestes altres. Cal dir que, com hem vist en els apartats anteriors, aquestes surten diferenciades per un desfase en la forma. Entre els tres fitxers f0, f0ref i el de wavesurfer, trobem diferències en alguns pics petits que de forma errònea s'han decidit com el que no estava previst (sord o sonor). Tot i així, veiem que el resultat ha estat

bastant bo. En un futur, es podria mirar de millorar el codi del filtre de mitjana ja que dubto que la millora real prevista sigui tan petita. Pel que fa a la gestió de versions amb git, penso que ha estat molt productiu i còmode de fer servir. Amb una primera bona explicació de les funcions més bàsica, s'ha pogut arribar a fer i entendre bastantes coses. Com a inconvenient, al fer la pràctica sol, potser no s'ha pogut exprimir del tot aquesta funció.

Per altre banda, el doxygen ha estat poc usat. El seguiment del projecte per aquest mitjà, penso que acaba sent poc revisat per la gran quantitat d'informació, no sempre útil. Potser per a projectes molt més extensos, acaba sent tant útil com preten ja que sembla molt interessant la manera en que permet dur el seguiment de les tasques al codi i la representació gràfica i escrita de la disposició de les classes i estructura del projecte.