Fites 4 i 5

Operacions amb dades

Màster Interuniversitari en Seguretat de les Tecnologies de la Informació i Comunicació

Cristian Requena Barreda

CANVIS DE LA IMPLEMENTACIÓ

Fins ara, el desenvolupament realitzat treballava amb mostres de so de 8 bits, és a dir, amb valors de l'interval [-128, 127]. Per tal de millorar l'exactitud de les dades, s'ha incrementat la mida de les mostres, essent ara de 16 bits. Així doncs, el valor de les mostres ara és dins del rang [-32768, 32767], que permet disposar d'un increment de precisió notable.

Per dur a terme aquesta modificació, ha estat necessari realitzar força canvis tant pel que fa a la captura de les dades d'entrada com a l'hora de treballar amb elles, ja que l'obtenció es realitza mitjançant una matriu de bytes que ha de ser convertida.

```
/**
 * Aquesta funció actualitza la variable data[] amb els doubles corresponents als bytes d'entrada.
 * @param byteArray
 */
public static void obtenirDades(byte[] byteArray) {
    assert byteArray.length == 2 * data.length;
    for (int i = 0; i < data.length; i++) {
        data[i] = (short) (((0xFF & byteArray[2 * i + 1]) << 8) | (0xFF & byteArray[2 * i]));
        data[i] /= Short.MAX_VALUE;
    }
}
```

Taula 1: Mètode de conversió byte -> short

D'altra banda, una vegada obtinguts els nous valors, s'ha modificat el mètode de repintat del gràfic que visualitza el valor de les mostres, per adequar-ne el funcionament al nou rang de dades.

DESENVOLUPAMENT

La metodologia emprada per a avaluar una mostra de so, implica realitzar un tractament a les dades de pressió sonora, que són les que es capturen directament mitjançant un micròfon. Sense efectuar cap operació, no és possible conèixer, per exemple, la freqüència amb la que s'exerceix aquesta pressió, és a dir, la freqüència del so capturat.

Als papers de referència, per exemple, [1], s'efectua sempre una operació d'aproximació mitjançant sèries de Fourier de la pressió sonora com a variable en funció del temps. Aquesta operació és coneguda com a transformada discreta de Fourier, existint una implementació concreta i computacionalment òptima anomenada Transformada Ràpida de Fourier[1], i és la que s'ha implementat a la solució.

La transformada de Fourier intenta aproximar al màxim una funció mitjançant un sumatori de factors composats per sinus i cosinus. És per això que s'adapta idealment a la veu humana: la pressió sonora emesa en forma de veu s'aproxima molt a una ona sinusoïdal, i la transformada es realitza prenent com a sumands del sumatori un rang de freqüències.

Alhora, la transformada de Fourier pot ser directa o inversa, canviant, en el segon cas, el signe dels exponents emprats per calcular els sumands.

$$f_j = \sum_{k=0}^{n-1} x_k e^{-\frac{2\pi i}{n} jk} \qquad j = 0, \ldots, n-1.$$

Imatge 1: Transformada de Fourier

1 Fast Fourier Transform, FFT.

1. Implementació de l'algorisme de la Transformada Discreta Ràpida de Fourier.

```
public class Fft {
    public static double[] fft(final double[] inputReal, double[] inputImag, boolean DIRECT) {
                int n = inputReal.length;
                double ld = Math.log(n) / Math.log(2.0);
                if (((int) ld) - ld != 0) {
                    return null;
                }
                int nu = (int) ld;
                int n2 = n / 2;
                int nu1 = nu - 1;
                double[] xReal = new double[n];
                double[] xImag = new double[n];
                double tReal, tImag, p, arg, c, s;
                double constant;
                if (DIRECT)
                    constant = -2 * Math.PI;
                else
                    constant = 2 * Math.PI;
                for (int i = 0; i < n; i++) {
                    xReal[i] = inputReal[i];
                    xImag[i] = inputImag[i];
                }
                int k = 0;
                for (int l = 1; l <= nu; l++) {
                    while (k < n) {
                        for (int i = 1; i <= n2; i++) {
                            p = bitreverseReference(k >> nu1, nu);
                            arg = constant * p / n;
                            c = Math.cos(arg);
                            s = Math.sin(arg);
                            tReal = xReal[k + n2] * c + xImag[k + n2] * s;
                            tImag = xImag[k + n2] * c - xReal[k + n2] * s;
                            xReal[k + n2] = xReal[k] - tReal;
                            xImag[k + n2] = xImag[k] - tImag;
                            xReal[k] += tReal;
                            xImag[k] += tImag;
                            k++;
                        }
                        k += n2;
                    }
                    k = 0;
                    nu1--;
                    n2 /= 2;
                }
                k = 0;
                int r;
                while (k < n) {
                    r = bitreverseReference(k, nu);
                    if (r > k) {
                        tReal = xReal[k];
                        tImag = xImag[k];
                        xReal[k] = xReal[r];
                        xImag[k] = xImag[r];
                        xReal[r] = tReal;
                        xImag[r] = tImag;
                    }
                    k++;
                }
                // Discretitzar.
                double[] newArray = new double[xReal.length];
                for (int i = 0; i < newArray.length;i++) {
                        newArray[i]=xReal[i]*xReal[i] + xImag[i] * xImag[i];
                }

                return newArray;
    }
```

Taula 2: Implementació FFT

2. Càlcul i visualització de l'espectre de so.

La captura del so es realitza, tal com s'ha especificat anteriorment, mitjançant mostres de 16 bits. A més a més, cal indicar que aquestes mostres es prenen 44100 vegades per segon (44100 Hz).

Per obtenir l'espectre de so (és a dir, un gràfic que relacioni la magnitud del so en funció de la seva freqüència), cal aplicar la transformada de Fourier a les dades d'entrada, però cal considerar la mida de les dades amb les que es realitzarà aquesta operació, perquè com més gran sigui el conjunt, menys periodicitat de refresc es pot donar. D'altra banda, l'avantatge de disposar d'una mida gran és que la sortida de la funció acumula les diferents freqüències en bandes, de manera que 512 mostres permeten obtenir 256 bandes, mentre que 1024 n'obtenen 512. A més quantitat de bandes s'obté una millor caracterització del so, perquè es pot estudiar la freqüència del mateix amb més precisió.

Arbitràriament he triat un valor de 2048 mostres, que obtenen un conjunt de 1024 bandes, que, al seu torn, caracteritzen un conjunt de 22050Hz, pel que cadascuna d'elles cobreix un rang de 21.53Hz. D'altra banda, es produiran uns 21 refrescs per segon, és a dir, cadascun d'ells serà de 0.046 segons.

Finalment, es realitza una conversió final amb els resultats de la transformada de Fourier, per a obtenir els valors en escala logarítmica:

```
        // Conversió d'FFT a escala logarítmica (dB)
        for (int i=0;i<espectre.length;i++) {
                espectre[i]=20*Math.log10(espectre[i] + 1); // Aplico un valor de referència d'1.
        }
```

Taula 3: Conversió a dB.

3. Càlcul i visualització del *Cepstrum* (espectre invers de l'espectre de so).

D'acord amb el paper de referència [1], cal obtenir el *cepstrum*, és a dir, la transformada inversa de Fourier de la transformada calculada anteriorment. Amb aquesta operació, s'obté una matriu de valors on les posicions inferiors són afectades pel moviment dels articuladors bucals del so i les superiors ho són pel so vocàlic.

És a dir, la part baixa de la matriu permetrà caracteritzar com s'articula un so, mentre que la part alta permetrà saber quin so és.

A partir d'aquestes dades, caldrà fer-ne una selecció (coeficients *ceptrals*), també anomenats *centroids*.

Cal tenir en compte que les dades d'entrada del càlcul de l'espectre invers (és a dir, la transformada inversa de Fourier de l'espectre) no poden ser emprades directament després de l'obtenció de les mateixes, sinó que és necessari de calcular-ne el logaritme del seu valor absolut per tal de normalitzar-les.

```
        // Aplicar IDFT a l'espectre per obtenir el cepstrum.
        // Primer cal obtenir el log en base 10 del valor absolut del vector.
        double[] temp = new double[sampleRate];
        for (int i=0;i<temp.length;i++) {
                temp[i] = Math.log10(Math.abs(espectre[i]));
        }
```

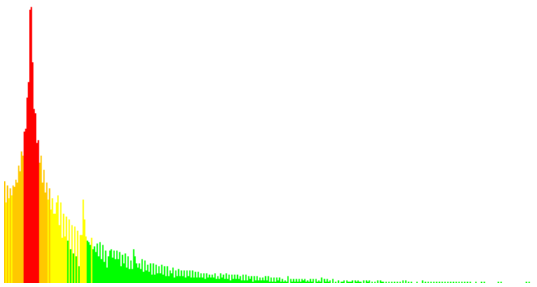
Taula 4: Normalització de l'espectre abans del càlcul de l'IFFT.

JOC DE PROVES

S'ha realitzat un joc de proves amb diversos sons per tal de comprovar el gràfic de l'espectre de so, tot coneixent que cadascuna de les seves bandes cobreix 21.53Hz. A l'eix d'abcisses es disposa del rang de freqüències, e 0 a 22050Hz, mentre que al d'ordenades s'hi ubica l'amplitud del so, en escala de dB.

Imatge 2: Prova d'ona sinusoïdal de 120Hz.

Es mostra una major amplitud a les columnes 5 i 6, equivalents a la franja de 107.65 a 129.18Hz i 129.18 a 150,71hz, respectivament.



Imatge 3: Prova d'ona sinusoïdal de 400Hz.

En aquest cas, la màxima amplitud es dóna a les franges 19 i 20, és a dir, entre 409,17 i 430,6 Hz.

La poca exactitud d'aquestes mesures correspon a pèrdues de qualitat degudes a les característiques dels altaveus i micròfon emprats.

També s'ha realitzat una altra prova, més dinàmica, amb un so d'ona sinusoïdal que varia de 20Hz a 20kHz, per tal de comprovar que es cobreix tot l'espectre de so. Tot i això, degut a les carències dels altaveus i del micròfon, tant les freqüències baixes com les molt altes no són capturades de manera acurada. Tanmateix, no és un problema per al projecte, ja que les freqüències en les que es genera la veu humana són mitjanes.

És disponible al següent enllaç: https://www.dropbox.com/s/t7l6jy1mcf1chss/Demo%20TFM.ts.

D'altra banda, s'ha comprovat l'algorisme de càlcul del *cepstrum*, però aquest varia amb molta rapidesa i encara no sembla que s'adeqüi al funcionament necessari.

BIBLIOGRAFIA

[1]: Monrose, et al. (2001). Cryptographic Key Generation from Voice. *IEEE Symposium on Security and Privacy*.

Implementacions FFT:
http://www.wikijava.org/wiki/The_Fast_Fourier_Transform_in_Java_%28part_1%29
https://www.ee.columbia.edu/~ronw/code/MEAPsoft/doc/html/FFT_8java-source.html
https://sites.google.com/site/piotrwendykier/software/jtransforms
http://www.fftw.org/download.html