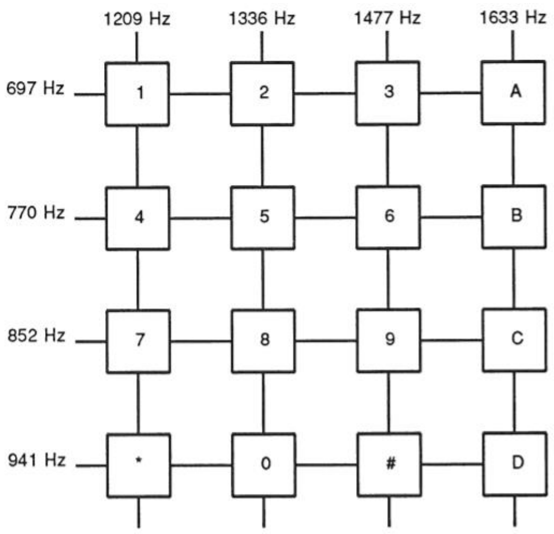
DTMF

Íñigo González-Varas Vallejo

Dual Tone Multi-Frequency (DTMF) is a signaling method used in telephone networks for call set-up, call forwarding control and group calls.
Each digit from 0 to 9, as well as the letters A to D and the symbols (#) and (*) are assigned a pair of tones, one of them of low frequency ***f1*** (lower than 1kHz) and the other of high frequency ***f2*** (between 1 and 2 kHz), following the attached scheme:



Requirements

The implementation of a **DTMF receiver** meets the following requirements:

- Detect frequencies with a **tolerance of +- 1.5%**.
- Tones that **exceed** the **+-3.5% tolerance range will not be detected**, so speech and other signals such as DTMF digits will not be detected.
- The receiver must have a **minimum SNR of 15 dB with 26 dB attenuation**.
- The receiver must be able to **detect DTMF signals when two tones are received at different levels**. If ***nf2<nf1*** it is called **normal rollover** and if ***nf1<nf2*** it is called **reverse rollover**, (nfi = level at frequency i).
- Spacing: **less than 10 digits/second**, **50 ms between tones**, **tone present for at least 40 ms**.
- Operation in the presence of **speech without identifying it as DTMF signals**.

Working principles

It is implemented on a **discrete signal processor**. To detect DTMF signaling tones, the **discrete Fourier transform of the signal** is taken **and the energy at the frequencies of interest is analyzed**. A time window of **12.75 ms** and a **sampling rate of 8 kHz** are used.

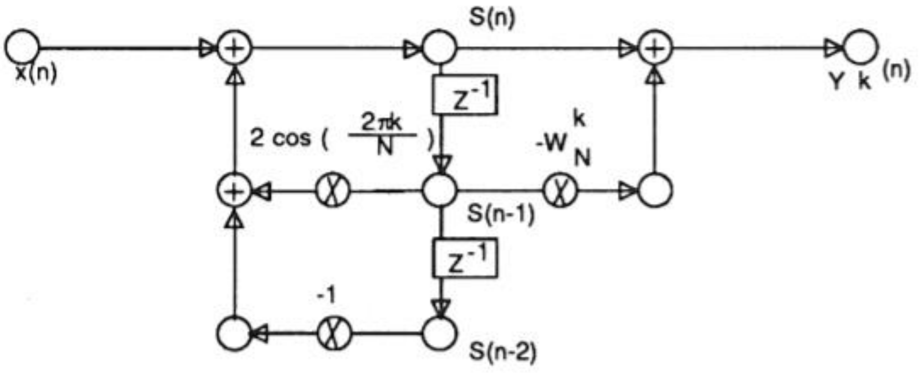
To **avoid confusion with speech and noise** there are two mechanisms:

- **The total energy** of the window we are analyzing must be calculated. Then we must determine if it exceeds a threshold. If it does not exceed this threshold, we are dealing with noise and not with signal.

- The following property of speech is used: if there is **energy at a frequency *f1***, given the **harmonic nature of speech**, it is likely that at a frequency ***f2 = 2f1* there is also energy**. Therefore, each time a **frequency is analysed to see if it is a DTMF** signalling **tone**, **its double** is analysed. If the **energy** is **below a threshold**, it is concluded to be a signaling **tone**.

Goertzel Algorithm

It is used to **calculate the discrete Fourier transform at a point**. You only want to analyse **48 frequencies** (**8 signalling frequencies, their 8 harmonics and the +- 1.5 % margins of the previous 16**), so it is much more **efficient** (in terms of memory and execution time) to **run this algorithm** than to perform the 256-point Fourier transform required for the signal sampled at 8 kHz. Its implementation is based on the following **flowchart**:



Where $\boldsymbol{Wnk = e^{\frac{-j2\pi}{N}}}$ y N = 102 samples. The **initial state** of all signals is set to **0**. k is obtained for each frequency of interest fo as $wo = \frac{2\pi k}{N}\ donde\ wo = \frac{2\pi fo}{fs}$. The filter is implemented as a type II direct form.

The objective is to calculate the energy, so once the **value at that point** has been calculated, it must be **squared to obtain the energy.**

Receiver Implementation

1. **The magnitude** of each of the **48 frequencies** of interest is **calculated** using **Goertzel**.
2. **The energy associated with each magnitude** is calculated.
3. The following **5 tests** are run:

   a. **Magnitude test**: the tone with the highest energy is obtained for each frequency band. If none of them exceeds a threshold, the test has failed.
   b. **Spin test**: the ratio of the high frequency tone to the low frequency tone is calculated. If it is outside the range -4 dB to 8 dB, the test has failed.
   c. **Offset frequency test**: the energy of the highest energy tone is compared with the other energies of the other tones. If the difference is below a threshold, the test has failed.
   d. **Tone to total energy test**: *c1*, low band energy, *c2* high band energy and *c3* sum of *c1* and *c2* are calculated. If any of these values is less than the total energy, the test has failed.

   e. **Harmonic ratio test**: the energy of the tone is compared with that of its second harmonic. If the energy of the second harmonic exceeds a threshold, we are dealing with a voice signal and the test will have failed.

4. If all tests are passed, the **tones** are **decoded as an integer from 0 to 15**, corresponding to **0-9, A-D, #, * respectively**. This value **is stored in memory in D(k)**. If **nothing has been detected, a -1 is stored**.
5. **In D**, the **same value** must have been **stored in two successive boxes for it to be validated**. If it has been stored in more than 2, **from the third value onwards it is not considered a new digit**, but a continuation of the previous ones.

Pseudocode implementation

Necessary functions

```
function xo = goertzel(signal, fs, f)
function that returns the TF in f of signal sampled at fs.

function Eo = energy(xo)
function that returns the energy at a point given the value of the signal at that point.

function [fLowMax, eBowMax, fHighMax, eHighMax, passed] = magnitude(energies)
% function that returns the maximum energy value in each frequency band, and the frequency that has that value. The energy is compared with the threshold, and if it is not exceeded, test1 returns false.

function pass = turn(fLowMax, fHighMax)
% function that calculates the dB ratio between the two input frequencies. If it is outside the range -4 dB to 8 dB, test2 returns false.

function approved = offset(energies, fLowMax, fHighMax)
% function that compares the energy at the maximum frequencies of each frequency band with the energy of the other frequencies in that band. If the difference does not exceed a certain threshold, it returns false.

function approved= energiatotal(energies)
% function that calculates the total energy of each frequency band and compares it to the total energy of the signal. If they are lower, test4 returns false

function approved = harmonic(energies, fLowMax, fHighMax)
% function that calculates the energy of the input frequencies and their second harmonics and compares them to a certain threshold. If the threshold is not exceeded it returns false

function symbol = decode(fLowMax, fHighMax)
% function decoding a dtmf symbol given its two frequencies
```

In addition to all these functions, a script (dtmf.m) will be necessary to control the execution of the program. This script must have the structure described in "Receiver implementation", as follows:

```
% DTMF
clear;
% 0.- cargar señal de test
TEST_01;

% 1.- Declaración de Variables
freqBaja = [697,770,852,941];
freqAlta = [1209,1336,1477,1633];
%frecuencias altas, bajas y sus frecuencias margen +- 1.5%
freqMargenes = [freqBaja.*0.985, freqBaja, freqBaja.*1.015, freqAlta.*0.985, freqAlta,↙
freqAlta*1.015];
freqArmonica = freqMargenes.*2;
freqTotal = [freqMargenes, freqArmonica];
fs = 8000; %Frecuencia de muestreo en Hz

N = 102; %muestras por ventana
inicioVentana = 1; %vectores en matlab empiezan en 1
finVentana = N;
finEjecucion = length(dtmf);
memoria = []; %array donde almacenamos la solución de cada iteración

%umbrales
% analizando las señales que recibimos,
% la energía de la ventana viene a ser superior a este valor cuando transmitimos señal
umbralVentana = 5*10^11;

% 2.- Bucle Ventanas
while finVentana < finEjecucion
    % Muestras de la señal pertenecientes a la ventana
    x = dtmf(inicioVentana:finVentana);
    % Cálculo del nivel a las frecuencias de interés con goertzel
    magnitud = goertzel(freqTotal, fs, x);
    % Cálculo energías
    E = abs(magnitud).^2;
    Eventana = sum(E);
    if Eventana > umbralVentana
        % Obtenemos las frecuencias de mayor energía en cada banda
        [fBajaMax, eBajaMax, fAltaMax, eAltaMax, flag] = testMagnitud(E);
        if flag
            flag = testGiro();
        end
        if flag
            flag = testOffset();
        end
        if flag
            flag = testEnergiaTotal();
        end
        if flag
            flag = testArmonico();
        end
        %Comprobamos si han pasado todos los test
%         if flag
%             %Si han pasado todos los test, descodificamos
%             memoria = [memoria, descodifica()];
%         else
%              %sino cargamos un -1
%              memoria = [memoria, -1];
%          end
    end
    % Actualización de Ventana, solapadas 50% con la anterior
    inicioVentana = inicioVentana + N/2;
    finVentana = finVentana + N/2;
end

% Tratamos los datos obtenidos en memoria, obteniendo lo que se saca por pantalla
secuencia = procesar(memoria);
% Sacamos por pantalla lo obtenido
disp(secuencia);
```