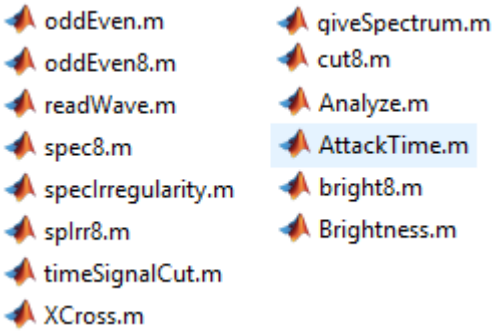
DSP – Recognition of feature of sound

Final MATLAB Project: 'Analyze'

The coded project's purpose is to compute unique parameters of sound, in order to use them to compare different sounds. The sound features included in the project are: attack time, xcrossing, brightness, spectrum irregularity, odd and even components of the spectrum. The first two features are calculated on time domain, while the others are calculated on frequency domain, on 8 different parts of the time domain.

1. The coded functions of the project:



2. The functions explained:
   - The main function Analyze():

   This is the main function of the project, here is where are called all the functions needed to compute the values of the features. The input values needed to be insert by user are: the file, which is the audio file where the descriptors are calculated, the 'cut percent', which is the threshold of the cut which decide which part of the audio wave is interesting to be computed, and the 'resolution coefficent', which set the ratio of the resolution of the spectrum.
   The output values of the function are:
   1. attack time from channel 1,
   2. attack time from channel 2,
   3. x crossing from channel 1,
   4. x crossing from channel 2,
   5. 8 values vector of brightness from channel 1 split in 8 parts spectrum,
   6. 8 values vector of brightness from channel 2 split in 8 parts spectrum,
   7. 8 values vector of spectrum irregularity from channel 1 split in 8 parts spectrum,
   8. 8 values vector of spectrum irregularity from channel 2 split in 8 parts spectrum,
   9. 8 values vector of odd component of the spectrum from channel 1 split in 8 parts spectrum,
   10. 8 values vector of odd component of the spectrum from channel 2 split in 8 parts spectrum,
   11. 8 values vector of even component of the spectrum from channel 1 split in 8 parts spectrum,
   12. 8 values vector of even component of the spectrum from channel 2 split in 8 parts spectrum.

```
function [attackTime1, attackTime2, xcross1, xcross2, vectBR1, vectBR2, vectIr1,
vectIr2, oddVector1, oddVector2, evenVector1, evenVector2] = Analyze (file,
cutPerc, coeffResol)

[ch1, ch2, fs]=readWave(file);
sig1=timeSignalCut(ch1, cutPerc);
sig2=timeSignalCut(ch2, cutPerc);
attackTime1=AttackTime(sig1, fs);
attackTime2=AttackTime(sig2, fs);
xcross1=XCross(sig1, 500);
xcross2=XCross(sig2, 500);
cutVect1=cut8(sig1);
cutVect2=cut8(sig2);
```

```
logSpectMatr1=spec8(sig1, cutVect1, fs*(coeffResol));
logSpectMatr2=spec8(sig2, cutVect2, fs*(coeffResol));
vectBR1=bright8(logSpectMatr1);
vectBR2=bright8(logSpectMatr2);
vectIr1=spIrr8(logSpectMatr1);
vectIr2=spIrr8(logSpectMatr2);
[oddVector1, evenVector1]=oddEven8(logSpectMatr1);
[oddVector2, evenVector2]=oddEven8(logSpectMatr2);

end
```

- `readWave():`

This function called on an audio file returns the two channels(ch1, ch2) and the sampling frequency(fs) of the file.

```
function [ch1,ch2,fs] = readWave(track)
[Y,fsTMP]=audioread(track);
ch1=Y(:,1);
ch2=Y(:,2);
fs=fsTMP;

end
```

- `timeSignalCut():`

This function from a channel(ch) and a percent value(perc) returns the cutted channel, which is the signal of interest(sig), and the two indexes where the channel had been cut, the beginning(Begin) and the ending(End) of the signal of interest on the channel.
The algorithm find these two indexes with the first two loops and set the signal as the channel between the two indexes.

```
function [sig,Begin,End] = timeSignalCut(ch, perc)
for i=1:length(ch)
    if ch(i)>=max(ch)*(perc)
        break;
    end
end
Begin=i;
i=length(ch);
while i>2
    if ch(i)>=max(ch)*(perc)
        break;
    end
    i=i-1;
end
End=i;
sig=ch(Begin:End);

end
```

- `AttackTime():`

This function from a signal and its sampling frequency calculates the attack time. The attack time is given by the ratio between the number of samples from the start of the signal to the max value of the signal, divided by the sampling frequency.

$$attackTime[s] = \frac{\#samples\ from\ sig(1)\ to\ \max(sig(:))}{sampling\ frequency[\frac{1}{s}]}$$

```
function [AtTime]= AttackTime(sig, fs)
```

```
[Ymax,i]=max(sig);
AtTime=i/fs;
end
```

- `XCross():`

This function from a signal and the desired interval calculate the number of xcrossing of the signal after the max value. E.g. if the interval is set to 500, it calculates the xcrossing value on the 500 values of the array after the max value. Then it returns the value of the feature.

```
function [xcross]= XCross(sig, intv)

A=0;
[ymax,i]=max(sig);
if intv<=1
    intv=length(sig)*1/10;
end
if i+intv>=length(sig)
    intv=length(sig)-i;
end
for j = i : i+intv
     if sig(j)<0 && sig(j+1)>0
       A=A+1;
     end
     if sig(j)>0 && sig(j+1)<0
       A=A+1;
     end
end
xcross=A;

end
```

- `cut8():`

This function from a signal inside and array it returns an array of indices of the array, which indices divide the signal in 8 same length parts.

```
function [cutVect] = cut8(sig)

tmp=length(sig)/8;
cutVect=[1,tmp,2*tmp,3*tmp,4*tmp,5*tmp,6*tmp,7*tmp,length(sig)];

end
```

- `spec8():`

This function from: a signal, the cut vector and the resolution, create a matrix with 8 rows, where in each row there is the spectrum of the correspondent part of the signal in the time domain, described by the cut vector, with the inserted resolution.

```
function [logSpec] = spec8(sig, cutVect, resolution)

logSpec(1, :)=giveSpectrum(sig(cutVect(1):cutVect(2)), resolution);
logSpec(2, :)=giveSpectrum(sig(cutVect(2):cutVect(3)), resolution);
logSpec(3, :)=giveSpectrum(sig(cutVect(3):cutVect(4)), resolution);
logSpec(4, :)=giveSpectrum(sig(cutVect(4):cutVect(5)), resolution);
logSpec(5, :)=giveSpectrum(sig(cutVect(5):cutVect(6)), resolution);
logSpec(6, :)=giveSpectrum(sig(cutVect(6):cutVect(7)), resolution);
logSpec(7, :)=giveSpectrum(sig(cutVect(7):cutVect(8)), resolution);
logSpec(8, :)=giveSpectrum(sig(cutVect(8):cutVect(9)), resolution);
```

```
end
```

- `giveSpectrum():`

This function from a time domain signal and the resolution value, returns the frequency domain spectrum normalized to 3, and splitted in the half, with the given resolution.

```
function [logSpec]=giveSpectrum(sig,Res)
tmp=zeros(1,Res);
for i=1:length(sig)
    tmp(i)=sig(i);
end
spec=abs(fft(tmp));
logSpec=log10(1000*spec/max(spec));
logSpec=logSpec(1:floor(length(logSpec)/2));

end
```

- `bright8():`

This function from the matrix that contains the spectrums of the parts of the signal, create a vector which contains the values of brightnesses of the spectrums(brVect).

```
function [brVect] = bright8(logSpecMatr)

brVect=[0,0,0,0,0,0,0,0];
[x,y]=size(logSpecMatr);
for i=1:x
    brVect(i)=Brightness(logSpecMatr(i,:));
end

end
```

- `BrightNess():`

This function from a spectrum calculates the value of the Brightness of that. The formula of the brightness is:

$$Br = \frac{\sum_{i=0}^{n} A(i) \cdot i}{\sum_{i=0}^{n} A(i)}$$

```
function [BR]= Brightness(logSpec)

den=0;
num=0;
for i=1 : length(logSpec)
    add=(logSpec(i)^2)*i;
    num=num+add;
end
add=0;
for i=1 : length(logSpec)
    add=logSpec(i)^2;
    den=den+add;
end
BR=num/den;

end
```

- `spIrr8():`

This function from the matrix which contains the 8 spectrums of the parts of the signal returns an 8-elements vector with the values of the spectrum irregularity of each part.

```
function [IrrVect] = spIrr8(logSpecMatr)
```

```
IrrVect= [0, 0, 0, 0, 0, 0, 0, 0];
[x, y]=size(logSpecMatr);
for i=1:x
    IrrVect(i)=specIrregularity(logSpecMatr(i, :));
end

end
```

- `specIrregularity():`

This function calculates the spectrum irregularity of the given spectrum. The spectrum irregularity formula is:

$$Ir = \log\left(20\sum_{k=2}^{N-1}\left|\log\frac{A_k}{\sqrt[3]{A_{k-1}\cdot A_k\cdot A_{k+1}}}\right|\right)$$

```
function [Ir] = specIrregularity(logSpec)

Sum=0;
for k= 2: length(logSpec)-1
    Sum=Sum+abs(log10(logSpec(k)/((logSpec(k1))*(logSpec(k))*(logSpec(k+1)))^(1/3)));
end
Ir=log10(20*Sum);

end
```

- `oddEven8():`

This function from the matrix which contains the 8 spectrums of the parts of the signal returns two 8-elements vectors. The first "oddVector" contains the values of the ratio of the odd components of each of the 8 spectrums. The second "evVector" contains the values of the ratio of the even components of each of the 8 spectrums.

```
function [oddVector, evVector]= oddEven8(logSpecMatr)

oddVector=[0, 0, 0, 0, 0, 0, 0, 0];
evVector=[0, 0, 0, 0, 0, 0, 0, 0];
[x,y]=size(logSpecMatr);
for i=1:x
    [oddVector(i), evVector(i)]=oddEven(logSpecMatr(i, :));
end

end
```

- oddEven():

This function calculates two values: 'odd'(the ratio of the odd components of the spectrum) and 'ev'(the ratio of the even components of the spectrum). These are the formulas of the two descriptors:

$$Ev = \frac{\sqrt{\sum_{i=1}^{M} A_{2i}^2(i)}}{\sqrt{\sum_{j=1}^{N} A_j^2(j)}}$$

$$Od = \frac{\sqrt{\sum_{i=2}^{L} A_{2i-1}^2(i)}}{\sqrt{\sum_{j=1}^{N} A_j^2(j)}}$$

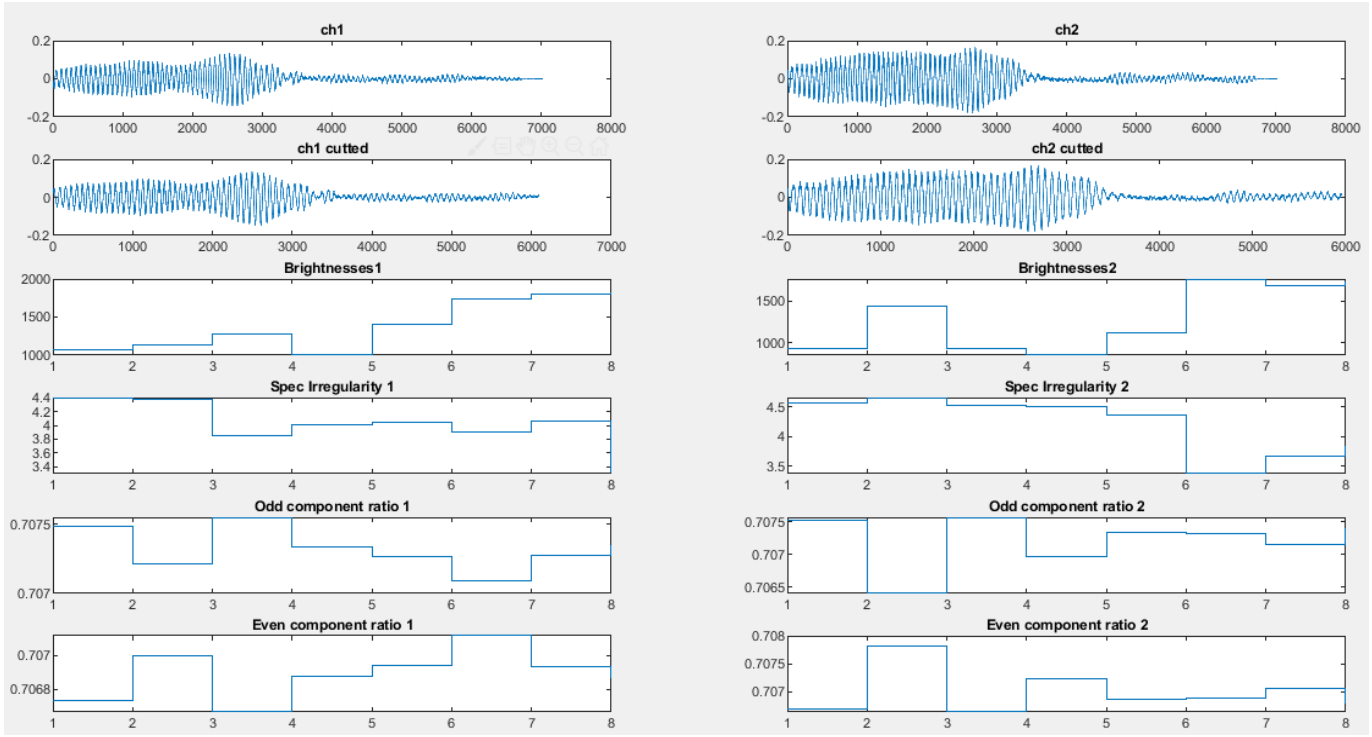
```
function [odd, ev]= oddEven(logSpec)

den=0;
for j=1:length(logSpec)
    den=(logSpec(j)^2)+den;
end
den=den^(1/2);
evNum=0;
for i=2:2:length(logSpec)
    evNum=evNum+(logSpec(i)^2);
end
evNum=evNum^(1/2);
oddNum=0;
for i=1:2:length(logSpec)
    oddNum=oddNum+(logSpec(i)^2);
end
```

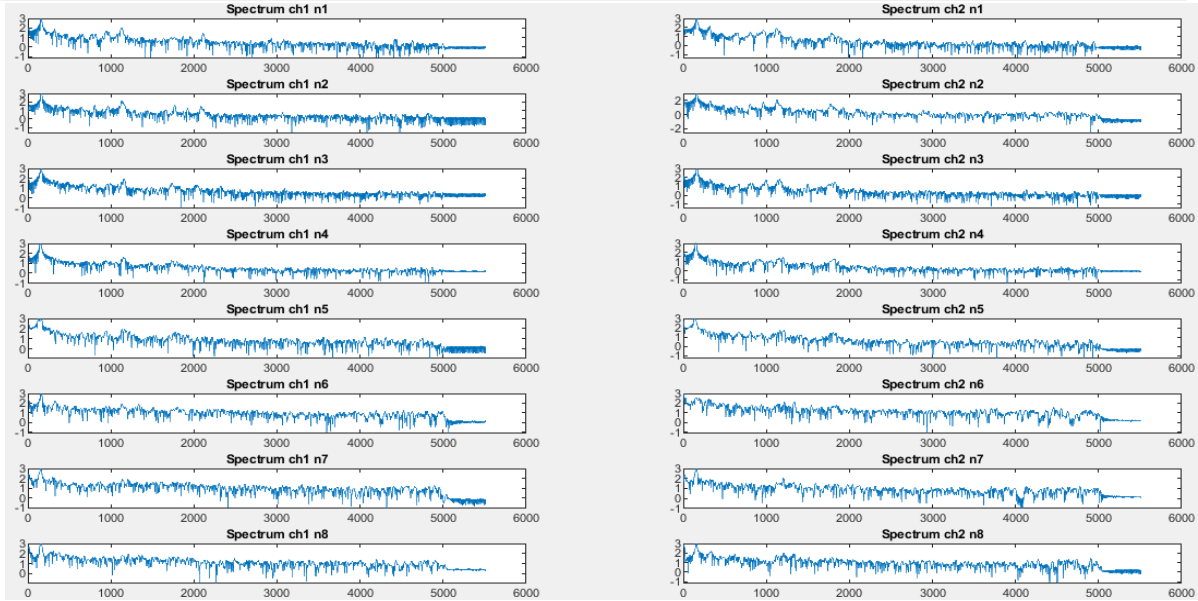
```
oddNum=oddNum^(1/2);
ev=evNum/den;
odd=oddNum/den;

end
```

3. Results with the given sound waves:

➢ On cuckoo.wav :

```
[ATtackTime1,ATtackTime2,xcross1,xcross2,vectBR1,vectBR2,vectIr1,vectIr2,oddVector1,oddVe
ctor2,evenVector1,evenVector2]=Analyze('cuckoo.wav', 0.10, 1/4)
```



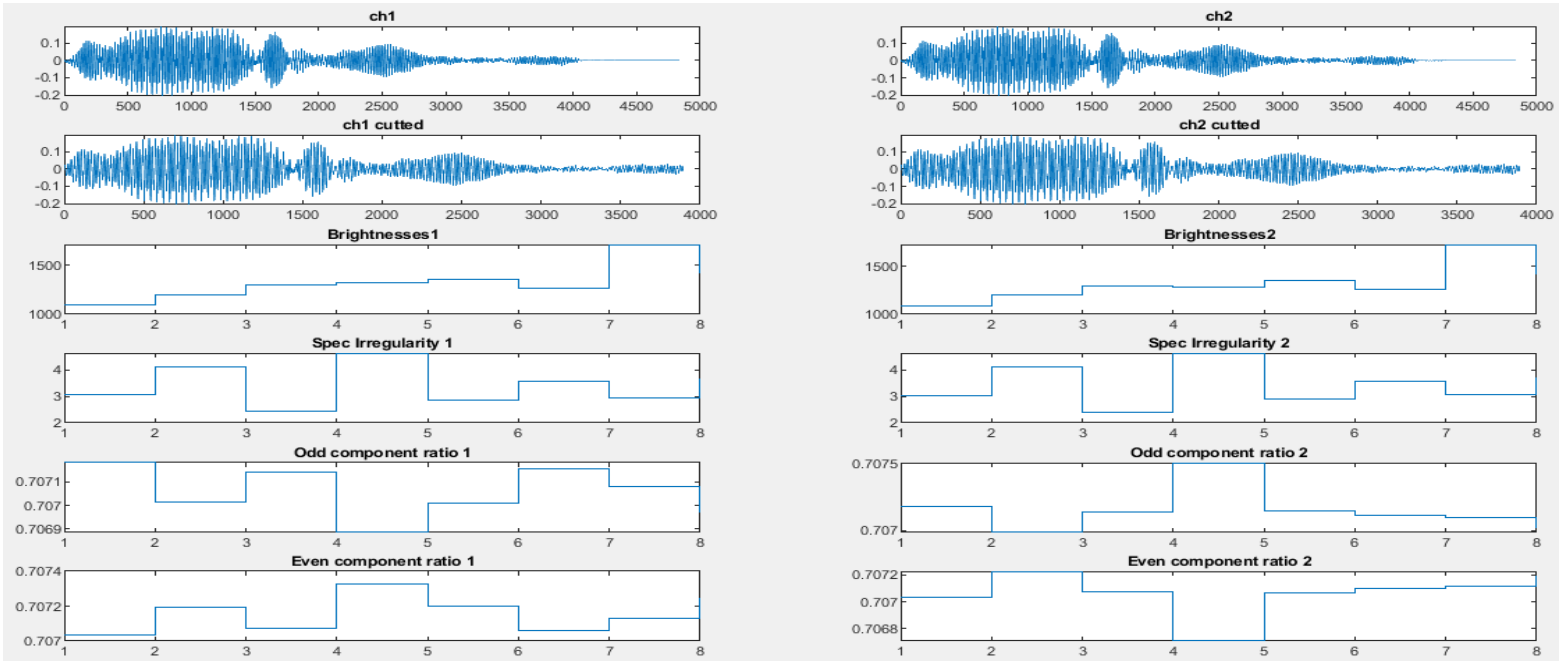



```
ATtackTime1 =

    0.0566

ATtackTime2 =

    0.0611

xcross1 =

    13

xcross2 =

    13

vectBR1 =

   1.0e+03 *

    1.0721    1.1355    1.2836    1.0123    1.4011    1.7362    1.7997    1.8340

vectBR2 =

   1.0e+03 *

    0.9348    1.4306    0.9316    0.8515    1.1176    1.7631    1.6859    1.7367
```

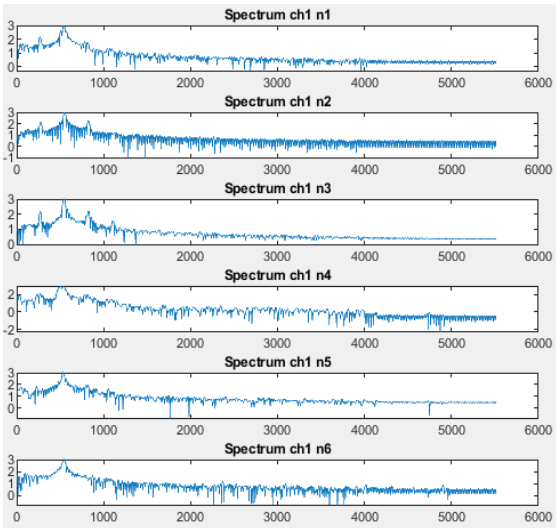
```
vectIr1 =

    4.4047    4.3767    3.8537    4.0038    4.0367    3.8975    4.0640    3.2969

vectIr2 =

    4.5792    4.6618    4.5282    4.5160    4.3688    3.3756    3.6698    3.8491

oddVector1 =

    0.7075    0.7072    0.7075    0.7073    0.7073    0.7071    0.7073    0.7073

oddVector2 =

    0.7075    0.7064    0.7076    0.7070    0.7073    0.7073    0.7072    0.7074

evenVector1 =

    0.7067    0.7070    0.7067    0.7069    0.7069    0.7071    0.7069    0.7069

evenVector2 =

    0.7067    0.7078    0.7066    0.7072    0.7069    0.7069    0.7071    0.7068
```

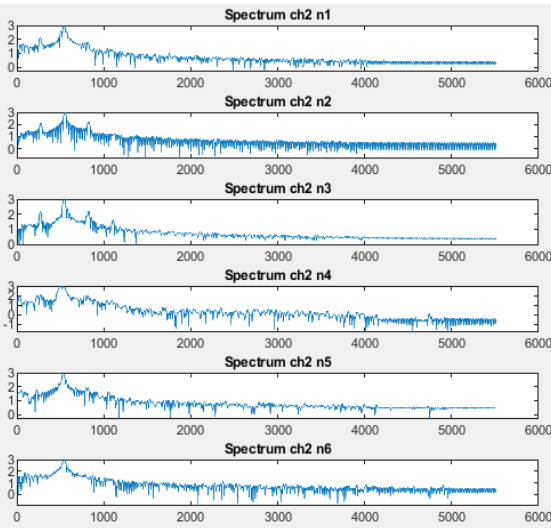
- On hawk1.wav :

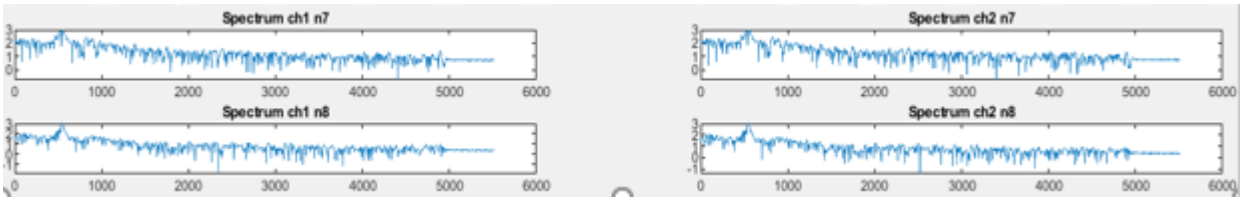
```
[ATtackTime1,ATtackTime2,xcross1,xcross2,vectBR1,vectBR2,vectIr1,vectIr2,oddV
ector1,oddVector2,evenVector1,evenVector2]=Analyze('hawk1.wav', 0.10, 1/4)
```




```
ATtackTime2 =

    0.0157

xcross1 =

    50

xcross2 =

    50
```

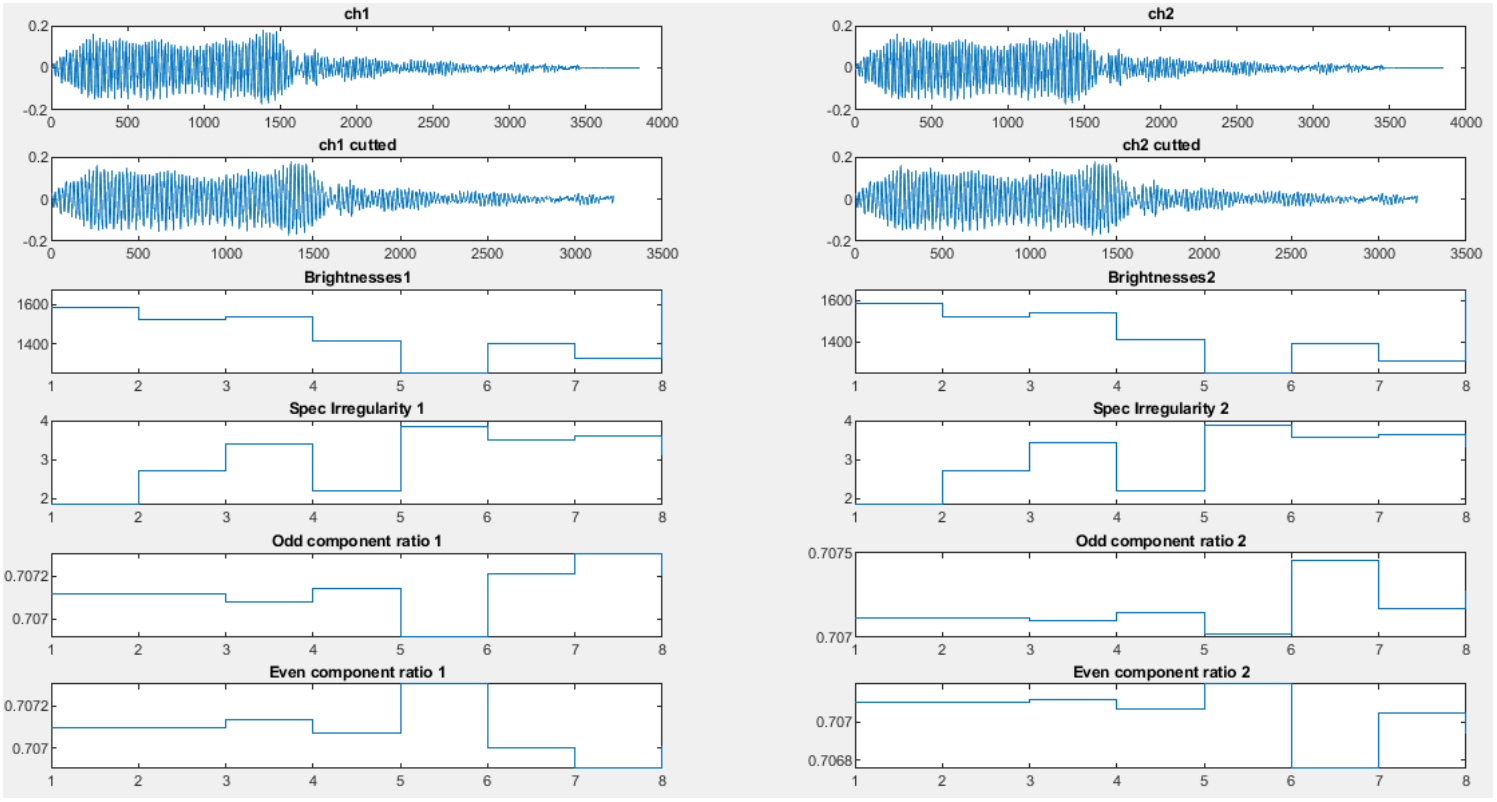


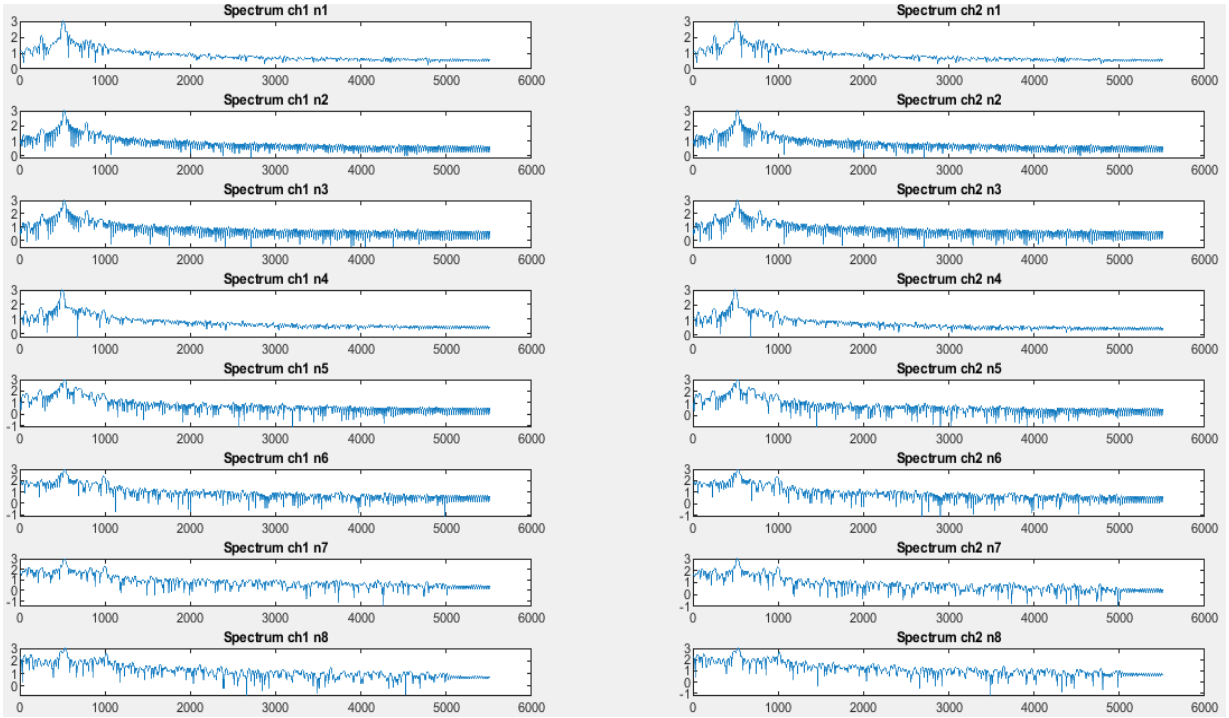





```
vectBR1 =

   1.0e+03 *

    1.0955    1.2018    1.2931    1.3242    1.3582    1.2635    1.7099    1.4152

vectBR2 =

   1.0e+03 *

    1.0867    1.2000    1.2929    1.2893    1.3588    1.2642    1.7313    1.4115

vectIr1 =

    3.0745    4.1076    2.4237    4.6369    2.8372    3.5720    2.9555    3.7048

vectIr2 =

    3.0142    4.1043    2.4148    4.6384    2.8991    3.5661    3.0524    3.7274

oddVector1 =

    0.7072    0.7070    0.7071    0.7069    0.7070    0.7072    0.7071    0.7070

oddVector2 =

    0.7072    0.7070    0.7071    0.7075    0.7071    0.7071    0.7071    0.7070

evenVector1 =

    0.7070    0.7072    0.7071    0.7073    0.7072    0.7071    0.7071    0.7072

evenVector2 =

    0.7070    0.7072    0.7071    0.7067    0.7071    0.7071    0.7071    0.7072
```

- On hawk1.wav:
  [ATtackTime1,ATtackTime2,xcross1,xcross2,vectBR1,vectBR2,vectIr1,vectIr2,oddVector1,oddVector2,evenVector1,evenVector2]=Analyze('hawk2.wav', 0.10, 1/4)





ATtackTime1 =

0.0311

ATtackTime2 =

0.0311

xcross1 =

45

xcross2 =

45

vectBR1 =

1.0e+03 *

1.5864 1.5210 1.5390 1.4108 1.2460 1.3972 1.3249 1.6778

vectBR2 =

1.0e+03 *

1.5872 1.5213 1.5389 1.4097 1.2459 1.3890 1.3092 1.6520

vectIr1 =

1.8320 2.6985 3.4112 2.2079 3.8362 3.5165 3.5898 3.0931

vectIr2 =

1.8342 2.6953 3.4302 2.2120 3.8757 3.5803 3.6525 3.2977

oddVector1 =

0.7071 0.7071 0.7071 0.7071 0.7069 0.7072 0.7073 0.7072

oddVector2 =

0.7071 0.7071 0.7071 0.7071 0.7070 0.7075 0.7072 0.7073

evenVector1 =

0.7071 0.7071 0.7071 0.7071 0.7073 0.7070 0.7069 0.7070

evenVector2 =

0.7071 0.7071 0.7071 0.7071 0.7072 0.7068 0.7070 0.7069