Rapport för laboration i digital ljudsyntes

Marc Coquand (författare)
maco0044
mcoquand@gmail.com

Oskar Olausson
osol0010
oskar.erik.olausson@gmail.com

17 mars 2016

Innehåll

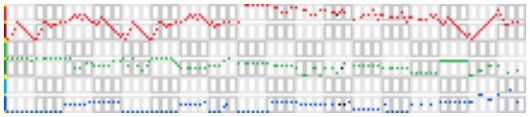
1 Inledning och problembeskrivning

Målsättningen med denna laborationen är att få fördjupande kunskaper i digital signalhantering genom att lära sig de grundläggande principerna för digital ljudsyntes av strängliknande instrument. Detta gjordes genom att läsa in en bild i Matlab som innehåller information om vilka toner som ska spelas upp och tonernas spacing.

Vi skapade därför ett tonbibliotek och satte samman tonerna till en låt. Därefter sparade vi låten till HD.

2 Genomförande

Uppgiften utfördes genom att man utvecklade ett program i Matlab som läser in en bild, konverterar bilden till information om vilka toner som ska spelas upp och tonernas spacing. Låten som valdes var en gotländsk polska som heter skaffare, bilden som beskriver låten syns i figur 1



Figur 1: Bilden som innehåller informationen om låtens toner och spacing

Varje pixel representerar en ton, färger som har RGB-värden högre än $(200, 200, 200)$ ignoreras och en helsvart pixel läses in som en halvton högre än pixels normala värde.

Programmet accepterar upp till 3 spår. Varje spår är 15 höjd och börjar på varandra. Pixeln högst upp i varje spår representerar högsta tonen och pixeln lägst ner i varje spår representerar lägsta tonen.

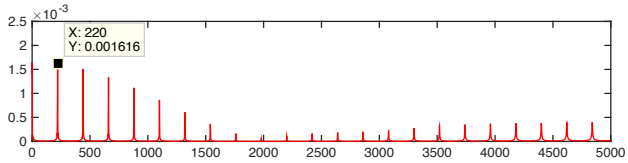
Efter att programmet konverterat bilden så läser den in varje spår och därefter konverterar det till rätt skala (bilden är i kromatisk skala annars). Där skapar den också tonerna med funktionen `newTone.m` vars källkod finns i bilagor.

Sist lägger den ihop alla kanaler till ett spår till en låt och normaliserar. Därefter sparar den låten.

Beskrivning om teorin bakom tonljuden hittas i bilagor.

3 Resultat

Den färdiga låten hittas på `https://github.com/oskarOlausson/Ljudprojekt/` i `MATLAB/skaffare.mp3`.



Figur 2: Fourierspektrum av en grundton skapad med programmet.

4 Slutsats

För att visa att tonerna var rena gjordes ett fourierspektrum av grundtonen som visas i figur **??**. Detta gjordes på alla toner men det är inte med i rapporten.

5 Reflektion

Tycker det är konstigt att denna laborationen är så tätt inpå labration 6.

6 Bilagor, matlabkod

6.1 Program

```
function out = fadeOut( in , duration, power )
%FADEOUT Summary of this function goes here
%   Detailed explanation goes here
    b=length(in)-duration;
    x=1-b:length(in)-b;
    %our sounds are vertical
    x=x';

    a=log(0.0001)/(duration);

    ramp = min(1,exp(1).^(x*a));
```

```
    %combine with sound and normalize
    out = in.* ramp.^power;
    out = normalize(out);

end
```

../fadeOut.m

```
%loading the picture where we have our song coded
pic = imread('SKAFFARE.png');

fs=getFreq();

pic = rgb2gray(pic);

disp('reading picture')
[guitar, guitar2, base] = imageToChroma(pic);

guitar=toKey(guitar);
guitar2=toKey(guitar2);
base=toKey(base);

%After first loop, play it again with a minor second
 guitar=[guitar raise(guitar,1)];
 guitar2=[guitar2 raise(guitar2,1)];
 base=[zeros(1,length(base))-1 raise(base,1)];

disp('making tones');

key=getKeyTones(0,1);
keyLow=getKeyTones(-1,0);

track1=zeros(fs*4,length(guitar));
track2=zeros(fs*4,length(guitar));
track3=zeros(fs*4,length(guitar));

disp('putting song together');
for index=1:length(guitar)
    track1(:,index) = fastTone(guitar(index),key);
    track2(:,index) = fastTone(guitar2(index),keyLow);
    track3(:,index) = fastTone(base(index),keyLow);
end

disp('mixing');
%mixing together
playable=normalize(1.5*track1+track2+track3);
```

```
40 [H,W]=size(playable);
%length of song, last notes ring out time
42 playthis=zeros(getSpacing()*W+        H*2,1);

44 for index=1:W
     place=getSpacing().*(index-1)+1;
46   playthis(place:place+H-1)=playthis(place:place+H-1)+playable
    (:,index);
end
48
playthis=normalize(playthis);
50 disp('done, playing song');
sound(playthis,getFreq());
52
disp('saving song');
54 filename='skaffare.wav';
 audiowrite(filename,playthis,getFreq());
56
 [in, FS]=audioread('skaffare.wav');
```

../fastPlayer.m

```
function tone = fastTone( fret , keys)
2 %FASTTONE Ordo one implementation of guitarTone
%    Needs that getKeyTones has been used
4     if (fret>=0)
          tone = keys( : , fret+1 );
6     else
          tone = zeros(getFreq()*4,1);
8     end

10 end
```

../fastTone.m

```
1 function out = filtered( in,f0 )

3     fs=getFreq();
      z=tf([1 0],1,fs);
5


7     R=0.999999;
      deltaAim=0.5;
9
      L = fs/f0 - deltaAim - 0.5;
```

```
11
    L = floor(L);
13  delta = fs/f0 - L - 0.5;

15  if (delta<0 || delta>1)
        L=L+1;
17      delta = fs/f0 - L - 0.5;
    end
19
    w0 = 2*pi*f0/fs;
21  a = sin((1-delta)*w0/2) / sin((1+delta)*w0/2);
    All = (z^(-1)+a)/(1+a*z^(-1));
23  Low = 0.5*(1+z^(-1));
    Kam = (R^L)*(z^(-L));
25
    Combined = Low*All / (1-All*Low*Kam);
27
    [num, den] = tfdata(Combined,'v');
29
    out = filter(num,den,in);
31
end
```

../filtered.m

```
1  function fs = getFreq()
   %GETFREQ works as a constant
3  fs=44100;

5  end
```

../getFreq.m

```
   function key = getKeyTones( from, to )
2  %GETKEYTONES sample all tones
   % from to - which octaves do we need to load
4
    startN=(12*from);
6   stopN=(12+to*12)+1;

8   fs=getFreq();
    len=stopN-startN;
10  height=fs*4;
    key=zeros(height,len);
12  tones=startN:stopN;
```

```
    for index=1:length(tones)

        key(:,index)=newTone(tones(index),0);

    end

end
```

../getKeyTones.m

```
function spacing = getSpacing(  )

    spacing = floor(getFreq()*0.12);


end
```

../getSpacing.m

```
function [guitar, guitar2, base] = imageToChroma( pic )
    %bright pixel (<200) doesnt count, entirely black pixels (0)
     are sharp
    range=15;

    %not counting first row
    pic = pic( : , 2:end );

    [H, W] = size(pic);

    guitar= zeros(1, W)-100;
    guitar2  = guitar;
    base = guitar;

    for r=1:H
        for c=1:W
            %bright colors are not counted
            pixel=pic(r,c);

            if pixel<200
                if r<=range
                    guitar(c)=range-r+0.5*(pixel==0);
                elseif r<=range*2
                    guitar2(c)=range*2-r+0.5*(pixel==0);
                else
                    base(c)=range*3-r+0.5*(pixel==0);
                end
```

```
            end
        end
    end
end
```

../imageToChroma.m

```
function tone = newTone( fret, octave )
    %GETTONE returns a tone that hopefully sounds like a guitar
    tone
    %Works by creating a frequency response for an insignal


    n  = 12*octave+fret;

    fs = getFreq();
    f0 = 220*2.^(n/12);

    in=zeros(fs*4,1);

    %minus one is a silent tone
    if fret==-100
        tone=in;
        return
    end

    len=30;
    in(1:len,1)=randn(len,1);

    tone=filtered(in,f0);

    %the notes fade out quickly when the next note comes
    tone = fadeOut(tone,fs*4-getSpacing()*2,2);

    tone = normalize(tone);

end
```

../newTone.m

```
function out = normalize( in )
    %This function normalizes the insignal into -1 to 1 range
    out = in/max(abs(in(:)));

end
```

../normalize.m

```
function out = raise( in , raisness)
%RAISE key changes

    in(in~=-100)=in(in~=-100)+raisness;
    out=in;

end
```

../raise.m