preprocessing.m

```
clear;
close all;
clc;
% Membaca file.wav
[x fs nbit]=wavread('nama file');
pjg=length(x);
% Membagi data menjadi frame
jml_frame=round(pjg/256);
jml=0;

for k=1:jml_frame
   x_win((k-1)*256+1:k*256)=x((k-1)*256+1:k*256).*hamming(256);
   x_win_tanda(1:256)=sign(x_win((k-1)*256+1:k*256));
   for m=2:256
      if x_win_tanda(m)~= x_win_tanda(m-1)
         jml=jml+1;
      end;
   end;
end
```

decompose1.m

```
Function [c,l] = wavedec(x,n,varargin)
%  WAVEDEC Dekomposisi Wavelet multilevel 1-D
% [C,L] = WAVEDEC(X,N,'wname') menghasilkan wavelet dekomposisi dari
    sinyal X
%   pada level N menggunakan ‘wname’ returns the wavelet
%   N =Level
%   [C,L] merupakan vektor yang merupakan output dari struktur dekomposisi.
%   For [C,L] = WAVEDEC(X,N,Lo_D,Hi_D),
%   Lo_D = dekomposisi low-pass filter
%   Hi_D = dekomposisi high-pass filter.

%   Susunan Struktur
%   C      =  [app. coef.(N)|det. coef.(N)|... |det. coef.(1)]
%   L(1)  =  length of app. coef.(N)
%   L(i)   =  length of det. coef.(N-i+2) for i = 2,...,N+1
%   L(N+2)  =  length(X)

% Argument
if errargn(mfilename,nargin,[3:4],nargout,[0:2]), error('*'), end
if errargt(mfilename,n,'int'), error('*'), end
if nargin==3
   [Lo_D,Hi_D] = wfilters(varargin{1},'d');
```

```
else
   Lo_D = varargin{1};   Hi_D = varargin{2};
end

% Inisialisasi
s = size(x); x = x(:)';
% row vector
c = [];
l = [length(x)];

for k = 1:n
   [x,d] = dwt(x,Lo_D,Hi_D);  % dekomposisi
   c     = [d c];             % menyimpan detail
   l     = [length(d) l];     % menyimpan panjang data
end

% Aproksimasi
c = [x c];
l = [length(x) l];

if s(1)>1, c = c'; l =` l';
end
```

decompose2.m

```
function lev = wmaxlev(sizeX,wname);
%  WMAXLEV level dekomposisi Wavelet maksimum
% L = WMAXLEV(S,'wname') merupakan maksimum level dekomposisi dari
    suatu sinyal
%   sizeX = ukuran (panjang) data X
%   wname = jenis wavelet yang dipilih dengan ordenya

% Argument
if errargn(mfilename,nargin,[2],nargout,[0 1]), error('*'); end
if     isempty(sizeX)
       lev = [];
       return;
elseif length(sizeX)==1
       lx = sizeX;
elseif min(sizeX)==1
       lx = max(sizeX);
else
       lx = min(sizeX);
end
```

```
wname = deblankl(wname);
[wtype,bounds] = wavemngr('fields',wname,'type','bounds');
switch wtype
  case {1,2}
    Lo_D = wfilters(wname);
    lw = length(Lo_D);

  case {3,4,5} , lw = bounds(2)-bounds(1)+1;

  otherwise
    errargt(mfilename,'invalid argument','msg');
    error('*');
end

%  Level terakhir
lev = fix(log(lx/(lw-1))/log(2));
if lev<1 , lev = 0; end
```

feature.m

```
% f_y = pitch_detec(x, window, hop, xformlength)
function f_y = pitch_detec(x, window, hop, xformlength)

% Windowing sinyal input
numwinds = ceil((size(x,1) - window)/hop) + 1;
windstart = 1;

h=1;
for(windnum = 1:numwinds)

% jika diperlukan untuk window terakhir
if(windnum ~= numwinds)
windx = x(windstart:windstart + window - 1);
   else
      windx = x(windstart:size(x,1));
      windx(size(windx,1) + 1:window) = 0;
      y(size(x, 1) + 1:windstart + window - 1) = 0;
end

% Window Hamming pada sampel
windx = windx .* hamming(window);

% STFT (konversi dari domain waktu ke domain frekuensi menggunakan FFT)
f_x = fft(windx, xformlength);
% HPS
% function f_y = hps(f_x)
```

```
f_x = f_x(1 : size(f_x,1) / 2);
f_x = abs(f_x);

% HPS, bagian1: downsampling
for i = 1:length(f_x)
     f_x2(i,1) = 1;
     f_x3(i,1) = 1;
     f_x4(i,1) = 1;
%  f_x5(i,1) = 1;
end

for i = 1:floor((length(f_x)-1)/2)
     f_x2(i,1) = (f_x(2*i,1) + f_x((2*i)+1,1))/2;
end

for i = 1:floor((length(f_x)-2)/3)
     f_x3(i,1) = (f_x(3*i,1) + f_x((3*i)+1,1) + f_x((3*i)+2,1))/3;
end

for i = 1:floor((length(f_x)-3)/4)
     f_x4(i,1) = (f_x(4*i,1) + f_x((4*i)+1,1) + f_x((4*i)+2,1) + f_x((4*i)+3,1))/4;
end

%   for i = 1:floor((length(f_x)-4)/5)
%   f_x5(i,1) = (f_x(5*i,1) + f_x((5*i)+1,1) + f_x((5*i)+2,1) + f_x((5*i)+3,1) +
     f_x((5*i)+4,1))/5;
%   end

% HPS, bagianII : perhitungan hasil
f_ym = (1*f_x) .* (1.0*f_x2);%  .* (1*f_x3)  .* f_x4; %.* f_x5;

% HPS, bagianIII : nilai maksimum
f_y1 = max(f_ym);
for c = 1 : size(f_ym)
     if(f_ym(c, 1) == f_y1)
          index = c;
     end
end

% Konversi ke frekuensi
f_y(h) = (index / xformlength) * 22000;

% Post-processing LPF
if(f_y(h) > 600)
   f_y(h) = 0;
   end
```

```
%  Increment pointer windstart
windstart = windstart + hop;

% f_y(h) = f_y1;
    h=h+1;
   f_y = abs(f_y)';
end
```

support_vector.m

```
function [ nsv , alpha , b0 ] = svc (X,Y,ker ,C)
if ( nargin <2 | nargin >4)
help svc
else
fprintf ('Klasifikasi Support Vektor\n')
fprintf (' _____ \n')
n = size (X ,1);
if ( nargin <4) C= Inf ;, end
if ( nargin <3) ker ='linear ';, end
fprintf ('Konstruksi ...\ n');
H = zeros(n,n);
for i =1:n
for j =1:n
H(i,j)=Y(i)*Y(j)*svkernel(ker,X(i,:),X(j,:));
end
end
c = - ones (n ,1);
H = H+1e-10* eye ( size (H ));
vlb = zeros (n ,1);
vub = C* ones (n ,1);
x0 = zeros (n ,1);
neqcstr = nobias ( ker );
if neqcstr
A = Y'; , b = 0;
else
A = []; , b = [];
end
fprintf ('Optimisasi ...\ n');
st = cputime ;
[ alpha lambda how ] = qp(H , c , A , b , vlb , vub , x0 , neqcstr );
fprintf ('Waktu eksekusi: %4.1 f seconds \n',cputime - st );
fprintf ('Status : % s\n',how );
w2 = alpha'*H*alpha ;
fprintf ('|w0 |^2 : % f\n',w2 );
fprintf ('Margin : % f\n' ,2/ sqrt (w2 ));
fprintf ('Sum alpha : % f\n',sum ( alpha ));
```

```
epsilon = svtol ( alpha );
svi = find ( alpha > epsilon );
nsv = length ( svi );
fprintf ('Support Vektor : % d (%3.1 f %%)\ n',nsv ,100* nsv /n);
b0 = 0;
if nobias ( ker ) ~= 0
svii = find ( alpha > epsilon & alpha < (C - epsilon ));
if length ( svii ) > 0
b0 = (1/ length ( svii ))* sum (Y( svii ) - H(svii , svi )* alpha ( svi ).* Y( svii ));
else
fprintf (‘Tidak ada support vector .\n');
          end
     end
end
```