Jan 03, 2019 Version 2

Plant leaf tooth feature extraction V.2

DOI
dx.doi.org/10.17504/protocols.io.v2ie8ce

Wang Hu[1], Li Chu[1], Tian Yan[2], Zhou Haoyu[1], Tian Di[3]

[1]China Shipbuilding Industry Corporation; [2]Huazhong University of Science and Technology; [3]Wenhua College

Di Tian

DOI: dx.doi.org/10.17504/protocols.io.v2ie8ce

Protocol Citation: Wang Hu, Li Chu, Tian Yan, Zhou Haoyu, Tian Di 2019. Plant leaf tooth feature extraction. **protocols.io** **https://dx.doi.org/10.17504/protocols.io.v2ie8ce**

Manuscript citation:

Protocol status: Working
We use this protocol and it's working

Created: November 30, 2018

Last Modified: January 03, 2019

Protocol Integer ID: 18218

Keywords: leaf tooth features, curvature, corner detection

Abstract

Previous studies extract features that are not strictly defined in botany; therefore, a uniform standard to compare the accuracies of various feature extraction methods cannot be used. For efficient and automatic retrieval of plant leaves from a leaf database, in this study, we propose an image-based description and measurement of leaf teeth by referring to the leaf structure classification system in botany. First, image preprocessing is carried out to obtain a binary map of plant leaves. Then, corner detection based on the curvature scale-space (CSS) algorithm is used to extract the inflection point from the edges; next, the leaf tooth apex is extracted by screening the convex points; then, according to the definition of the leaf structure, the characteristics of the leaf teeth are described and measured in terms of number of orders of teeth, tooth spacing, number of teeth, sinus shape, and tooth shape.

Experiment 1

1

To verify whether the proposed leaf structure feature description algorithm is scientific and effective, we implemented the algorithm using MATLAB 2017 (MathWorks, Natick, MA, USA) on a standard desktop PC (4.2 GHz CPU, 24 GB RAM). Processing of a single leaf took approximately 1.4 s. This could undoubtedly be improved through further optimization and/or using parallel computing.

Command

Leaf tooth Feature Extraction (windows 10)

```
toothFeature_finished('D:\Experiment 1\data\1.jpg')
```

Command

toothFeature_finished.m (windows 10)

```
% Leaf tooth Feature Extraction
%
%Locates and measures the teeth found at the margin of a leaf.
%Input: leafFile, a leaf object or a file containing a single leaf
object. This
% object has fields 'image_name' = RGB image of single leaf; 'x' and
'y' = set of
% Cartesian coordinates of boundary of leaf.
%
%
%Output:toothNumber,sinusShape,flagRegular,order,flag
%  Tooth number: the total number of teeth found
%  flagRegular:Tooth spacing
%  sinusShape: Sinus shape
%  order: Number of orders of teeth
%  toothShape: Tooth shape
%
%Requires Matlab Image Processing Toolbox.
%
% Authors:
%WangHu, TianDi
% CSIC, 2016-2018
% holmoak@qq.com

% v.1.0  March 2012
function [data]=toothFeature_finished(image_name)   %%Main function
"data=toothFeature_finished('C:\Users\Beaut\Desktop\testimages\6.jpg')
;"

Para=[1.5,162,4,0.25,0,1,3];
C=Para(1);
T_angle=Para(2);
sig=Para(3);
H=Para(4);
L=Para(5);
Endpoint=Para(6);
Gap_size=Para(7);
I=imread(image_name);
clc;
close all;
```

```
I1=preprocess(I);
BW=edge(I1,'canny',[L,H]);  % Detect edges
[curve,curve_start,curve_end,curve_mode,curve_num]=extract_curve(BW,Ga
p_size);  % Extract curves
[cout,K1]=get_corner(curve,curve_start,curve_end,curve_mode,curve_num,
BW,sig,Endpoint,C,T_angle); % Detect corners
[convex,toothNumber]=convexFinder(cout,7,I1);%convex extraction
%curve会有分段的情况
 [r,c]=size(curve);
 A=curve{1,1};
 for i=2:c
     A=[A
        curve{1,i}];
 end
%K1为轮廓点坐标，对应的曲率
 B=K1{1,1};
 for i=2:c
     B=[B
        K1{1,i}];
 end
%concave classify Angular and Rounded sinus
[concave,sinusShape]=concaveFinder(A,convex,B);
%% Orders of teeth distribution
[distan,convex,order]=toothOrder(convex,concave);%order
flagRegular=toothSpacing(convex);
 % Tooth shape
toothShape=toothShapeFinder(curve,convex,concave,I1,I);
data=
[toothNumber,sinusShape,flagRegular,order,toothShape];%%sinusShape=1
Angular sinus ,sinusShape=0 Rounded sinus; flagRegular=1 Irregular,
flagRegular=0 Regular
disp(data);                                                %%
toothShape=1 CC ;toothShape=2 CV ;toothShape=3 ST ;toothShape=12 FL ;
  toothShape=21 RT
 end

function
[curve,curve_start,curve_end,curve_mode,cur_num]=extract_curve(BW,Gap_
size)
%   Function to extract curves from binary edge map, if the endpoint
of a
%   contour is nearly connected to another endpoint, fill the gap and
continue
%   the extraction. The default gap size is 1 pixles.
```

```
[L,W]=size(BW);
BW1=zeros(L+2*Gap_size,W+2*Gap_size);
BW_edge=zeros(L,W);
BW1(Gap_size+1:Gap_size+L,Gap_size+1:Gap_size+W)=BW;
[r,c]=find(BW1==1);%
cur_num=0;

while size(r,1)>0
    point=[r(1),c(1)];
    cur=point;
    BW1(point(1),point(2))=0;
    [I,J]=find(BW1(point(1)-Gap_size:point(1)+Gap_size,point(2)-
Gap_size:point(2)+Gap_size)==1);
    b=0;
    while size(I,1)>0
        dist=(I-Gap_size-1).^2+(J-Gap_size-1).^2;
        [min_dist,index]=min(dist);
        point=point+[I(index),J(index)]-Gap_size-1;
        cur=[cur;point];
        BW1(point(1),point(2))=0;
        [I,J]=find(BW1(point(1)-Gap_size:point(1)+Gap_size,point(2)-
Gap_size:point(2)+Gap_size)==1);
        b=b+1;
    end

    % Extract edge towards another direction
    point=[r(1),c(1)];
    BW1(point(1),point(2))=0;
    [I,J]=find(BW1(point(1)-Gap_size:point(1)+Gap_size,point(2)-
Gap_size:point(2)+Gap_size)==1);
    while size(I,1)>0
        dist=(I-Gap_size-1).^2+(J-Gap_size-1).^2;
        [min_dist,index]=min(dist);
        point=point+[I(index),J(index)]-Gap_size-1;
        cur=[point;cur];
        BW1(point(1),point(2))=0;
        [I,J]=find(BW1(point(1)-Gap_size:point(1)+Gap_size,point(2)-
Gap_size:point(2)+Gap_size)==1);
    end

    if size(cur,1)>(size(BW,1)+size(BW,2))/25
        cur_num=cur_num+1;
        curve{cur_num}=cur-Gap_size;
    end
    [r,c]=find(BW1==1);

end
```

```
end

for i=1:cur_num
    curve_start(i,:)=curve{i}(1,:);
    curve_end(i,:)=curve{i}(size(curve{i},1),:);
    if (curve_start(i,1)-curve_end(i,1))^2+...
        (curve_start(i,2)-curve_end(i,2))^2<=32
        curve_mode(i,:)='loop';
    else
        curve_mode(i,:)='line';
    end

    BW_edge(curve{i}(:,1)+(curve{i}(:,2)-1)*L)=1;
end
% figure(1)
% imshow(~BW_edge)
% title('Edge map')
% imwrite(~BW_edge,'edge.jpg');
end

 function [cout,K1]=get_corner(curve,curve_start,curve_end,curve_mode,curve_num,BW,sig,Endpoint,C,T_angle)%角点检测

corner_num=0;
cout=[];
GaussianDieOff = .0001;
pw = 1:30;
ssq = sig*sig;
width = max(find(exp(-(pw.*pw)/(2*ssq))>GaussianDieOff));
if isempty(width)
    width = 1;
end
t = (-width:width);
gau = exp(-(t.*t)/(2*ssq))/(2*pi*ssq);
gau=gau/sum(gau);

for i=1:curve_num;
    [m,n] = size(curve{1,i});
    x=curve{i}(:,1);
    y=curve{i}(:,2);
    K1{i}(:,1)=x;
    K1{i}(:,2)=y;
    W=width;
    L=size(x,1);
    if L>W
        % Calculate curvature
        if curve_mode(i,:)=='loop'
            x1=[x(L-W+1:L);x;x(1:W)];
            y1=[y(L-W+1:L);y;y(1:W)];
        else
            x1=[ones(W,1)*2*x(1)-x(W+1:-1:2);x;ones(W,1)*2*x(L)-x(L-
1:-1:L-W)];
            y1=[ones(W,1)*2*y(1)-y(W+1:-1:2);y;ones(W,1)*2*y(L)-y(L-
1:-1:L-W)];
        end

        xx=conv(x1,gau);
        xx=xx(W+1:L+3*W);
        yy=conv(y1,gau);
        yy=yy(W+1:L+3*W);
        Xu=[xx(2)-xx(1) ; (xx(3:L+2*W)-xx(1:L+2*W-2))/2 ; xx(L+2*W)-
xx(L+2*W-1)];
        Yu=[yy(2)-yy(1) ; (yy(3:L+2*W)-yy(1:L+2*W-2))/2 ; yy(L+2*W)-
yy(L+2*W-1)];
        Xuu=[Xu(2)-Xu(1) ; (Xu(3:L+2*W)-Xu(1:L+2 *W-2))/2 ; Xu(L+2*W)-
Xu(L+2*W-1)];
        Yuu=[Yu(2)-Yu(1) ; (Yu(3:L+2*W)-Yu(1:L+2*W-2))/2 ; Yu(L+2*W)-
Yu(L+2*W-1)];
        K=abs((Xu.*Yuu-Xuu.*Yu)./((Xu.*Xu+Yu.*Yu).^1.5));
        %Kreal=(Xu.*Yuu-Xuu.*Yu)./((Xu.*Xu+Yu.*Yu).^1.5);
        K1{i}(:,3)=K(13:m+12)';


        K=ceil(K*100)/100;

        % Find curvature local maxima as corner candidates
        extremum=[];
        N=size(K,1);
        n=0;
        Search=1;

        for j=1:N-1
            if (K(j+1)-K(j))*Search>0
                n=n+1;
                extremum(n)=j;  % In extremum, odd points is minima
and even points is maxima
                Search=-Search;
            end
        end
        if mod(size(extremum,2),2)==0
            n=n+1;
            extremum(n)=N;
        end

        n=size(extremum,2);
        flag=ones(size(extremum));

        % Compare with adaptive local threshold to remove round
corners
        for j=2:2:n
            %I=find(K(extremum(j-1):extremum(j+1))==max(K(extremum(j-
1):extremum(j+1))));
            %extremum(j)=extremum(j-1)+round(mean(I))-1; % Regard
middle point of plateaus as maxima

            [x,index1]=min(K(extremum(j):-1:extremum(j-1)));
            [x,index2]=min(K(extremum(j):extremum(j+1)));
            ROS=K(extremum(j)-index1+1:extremum(j)+index2-1);
            K_thre(j)=C*mean(ROS);
            if K(extremum(j))<K_thre(j)
                flag(j)=0;
            end
        end
        extremum=extremum(2:2:n);
        flag=flag(2:2:n);
        extremum=extremum(find(flag==1));

        % Check corner angle to remove false corners due to boundary
noise and trivial details
        flag=0;
        smoothed_curve=[xx,yy];
        while sum(flag==0)>0
            n=size(extremum,2);
            flag=ones(size(extremum));
            for j=1:n
                if j==1 & j==n

ang=curve_tangent(smoothed_curve(1:L+2*W,:),extremum(j));
                elseif j==1

ang=curve_tangent(smoothed_curve(1:extremum(j+1),:),extremum(j));
                elseif j==n
                    ang=curve_tangent(smoothed_curve(extremum(j-
1):L+2*W,:),extremum(j)-extremum(j-1)+1);
                else
                    ang=curve_tangent(smoothed_curve(extremum(j-
1):extremum(j+1),:),extremum(j)-extremum(j-1)+1);
                end
                if ang>T_angle & ang<(360-T_angle)
                if ang>T_angle & ang<(360-T_angle)
                    flag(j)=0;
                end
            end

            if size(extremum,2)==0
                extremum=[];
            else
                extremum=extremum(find(flag~=0));
            end
        end

        extremum=extremum-W;
        extremum=extremum(find(extremum>0 & extremum<=L));
        n=size(extremum,2);
        for j=1:n
            corner_num=corner_num+1;
            cout(corner_num,:)=curve{i}(extremum(j),:);
        end
    end
end


% Add Endpoints
if Endpoint
    for i=1:curve_num
        if size(curve{i},1)>0 & curve_mode(i,:)=='line'

            % Start point compare with detected corners
            compare_corner=cout-ones(size(cout,1),1)*curve_start(i,:);
            compare_corner=compare_corner.^2;
            compare_corner=compare_corner(:,1)+compare_corner(:,2);
            if min(compare_corner)>25       % Add end points far from detected corners
                corner_num=corner_num+1;
                cout(corner_num,:)=curve_start(i,:);
            end

            % End point compare with detected corners
            compare_corner=cout-ones(size(cout,1),1)*curve_end(i,:);
            compare_corner=compare_corner.^2;
            compare_corner=compare_corner(:,1)+compare_corner(:,2);
            if min(compare_corner)>25
                corner_num=corner_num+1;
                cout(corner_num,:)=curve_end(i,:);
            end
        end
    end
end
 end

function ang=curve_tangent(cur,center)

for i=1:2
    if i==1
        curve=cur(center:-1:1,:);
    else
        curve=cur(center:size(cur,1),:);
    end
    L=size(curve,1);

    if L>3
        if sum(curve(1,:)~=curve(L,:))~=0
            M=ceil(L/2);
            x1=curve(1,1);
            y1=curve(1,2);
            x2=curve(M,1);
            y2=curve(M,2);
            x3=curve(L,1);
            y3=curve(L,2);
        else
            M1=ceil(L/3);
            M2=ceil(2*L/3);
            x1=curve(1,1);
            y1=curve(1,2);
            x2=curve(M1,1);
            y2=curve(M1,2);
            x3=curve(M2,1);
            y3=curve(M2,2);
        end

        if abs((x1-x2)*(y1-y3)-(x1-x3)*(y1-y2))<1e-8  % straight line
            tangent_direction=angle(complex(curve(L,1)-
curve(1,1),curve(L,2)-curve(1,2)));
        else
            % Fit a circle
            x0 = 1/2*(-y1*x2^2+y3*x2^2-y3*y1^2-y3*x1^2-
y2*y3^2+x3^2*y1+y2*y1^2-y2*x3^2-y2^2*y1+y2*x1^2+y3^2*y1+y2^2*y3)/(-
y1*x2+y1*x3+y3*x2+x1*y2-x1*y3-x3*y2);
            y0 = -1/2*(x1^2*x2-x1^2*x3+y1^2*x2-y1^2*x3+x1*x3^2-
x1*x2^2-x3^2*x2-y3^2*x2+x3*y2^2+x1*y3^2-x1*y2^2+x3*x2^2)/(-
y1*x2+y1*x3+y3*x2+x1*y2-x1*y3-x3*y2);
            % R = (x0-x1)^2+(y0-y1)^2;
            radius_direction=angle(complex(x0-x1,y0-y1));
            adjacent_direction=angle(complex(x2-x1,y2-y1));
            tangent_direction=sign(sin(adjacent_direction-
radius_direction))*pi/2+radius_direction;
        end

    else % very short line
        tangent_direction=angle(complex(curve(L,1)-
curve(1,1),curve(L,2)-curve(1,2)));
    end
    direction(i)=tangent_direction*180/pi;
end
ang=abs(direction(1)-direction(2));
end

function img1=mark(img,x,y,w)%use to draw figure of concave and
convex points

[M,N,C]=size(img);
img1=img;

if isa(img,'logical')
    img1(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),:)=...
        (img1(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),:)<1);
    img1(x-floor(w/2)+1:x+floor(w/2)-1,y-
floor(w/2)+1:y+floor(w/2)-1,:)=...
        img(x-floor(w/2)+1:x+floor(w/2)-1,y-
floor(w/2)+1:y+floor(w/2)-1,:);
else
    img1(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),:)=...
        (img1(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),:)<128)*255;
    img1(x-floor(w/2)+1:x+floor(w/2)-1,y-
floor(w/2)+1:y+floor(w/2)-1,:)=...
        img(x-floor(w/2)+1:x+floor(w/2)-1,y-
floor(w/2)+1:y+floor(w/2)-1,:);
end
end
   % red point -- One order leaf tooth
function img2=mark2(img,x,y,w)
     [M,N,C]=size(img);
     img2=img;
     img2(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),1)=255;
     img2(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),2)=0;
     img2(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),3)=0;


    img2(x-floor(w/2)+1:x+floor(w/2)-1,y-
floor(w/2)+1:y+floor(w/2)-1,:)=...
        img(x-floor(w/2)+1:x+floor(w/2)-1,y-
floor(w/2)+1:y+floor(w/2)-1,:);
end
   % blue point -- two orders leaf tooth
function img3=mark1(img,x,y,w)
     [M,N,C]=size(img);
     img3=img;

     img3(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),1)=0;
     img3(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),2)=0;
     img3(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),3)=255;


    img3(x-floor(w/2)+1:x+floor(w/2)-1,y-
floor(w/2)+1:y+floor(w/2)-1,:)=...
        img(x-floor(w/2)+1:x+floor(w/2)-1,y-
floor(w/2)+1:y+floor(w/2)-1,:);
end

function [I,C,T_angle,sig,H,L,Endpoint,S,Gap_size,Name] =
parse_inputs(varargin);

error(nargchk(0,8,nargin));

Para=[1.5,162,6,0.25,0,1,60,3]; %Default experience value;H=0.35

if nargin>=2
    I=varargin{1};
    for i=2:nargin
        if size(varargin{i},1)>0
            Para(i-1)=varargin{i};
        end
    end
```

```
end

if nargin==1
    I=varargin{1};
end

if nargin==0 | size(I,1)==0
    [fname,dire]=uigetfile('*.bmp;*.jpg;*.gif','Open the image to be
detected');
    I=imread([dire,fname]);
end

C=Para(1);
T_angle=Para(2);
sig=Para(3);
H=Para(4);
L=Para(5);
Endpoint=Para(6);
S=Para(7);
Gap_size=Para(8);
end%暂不用

function [h1]=preprocess(img)%preprocess

I = rgb2gray(img);
threshold = graythresh(I);%
bw = ~im2bw(I,threshold);%
se = strel('disk',2);%
bw = imclose(bw,se);%
bw = imfill(bw,'holes');%
h1=~bw;

end

function [convex,toothNumber]=convexFinder(cout,r,img)%classsify
concave and convex points

img=double(img);
convex=[];
toothNumber=0;

for k=1:size(cout,1)
target=0;
base=0;

for m=-r:r
```

```
    for n=-r:r
        if m^2+n^2<=r^2
            if img(cout(k,1)+m,cout(k,2)+n)==0
                target=target+1;
            elseif img(cout(k,1)+m,cout(k,2)+n)==1
                base=base+1;
            end
        end
    end
end
if target<0.8*base% it is concave when the number of black points
less than the white points
   toothNumber=toothNumber+1;
   convex(toothNumber,1)=cout(k,1);
   convex(toothNumber,2)=cout(k,2);


end
end
end

function [concave,sinusShape]=concaveFinder(curve,convex,K1)%concave
fingding

concave=[];
cur=[];
m=1;
%calculate the distance between the edge points and the points
between two
%convex points
for i=1:size(convex,1)-1
    x1=convex(i,2);
    y1=convex(i,1);
    x2=convex(i+1,2);
    y2=convex(i+1,1);
    [rc1,lc1]=find(curve(:,2)==x1);
    [rc2,lc2]=find(curve(:,1)==y1);
    j=intersect(rc1,rc2);%求交集
    [rc1,lc1]=find(curve(:,2)==x2);
    [rc2,lc2]=find(curve(:,1)==y2);
    k=intersect(rc1,rc2);

   x3=curve(j:k,2);
   y3=curve(j:k,1);
  [rx3,cx3]=size(x3);
   x1=ones(rx3,1).*x1;
   y1=ones(rx3,1).*y1;
   y1=ones(rx3,1).*y1;
   x2=ones(rx3,1).*x2;
   y2=ones(rx3,1).*y2;
   distan=abs((x2-x1).*y3-(y2-y1).*x3-(x2-x1).*y1+x1.*(y2-
y1))./sqrt((x2-x1).*(x2-x1)+(y2-y1).*(y2-y1));

   [mem,Pos]=max(distan(:,1));
   concave(m,1)=curve(j+Pos-1,1);
   concave(m,2)=curve(j+Pos-1,2);
   cur(m,1)=K1(j+Pos-1,3);
   m=m+1;



end
%classify sinus shape
if mean(cur)>0.0512
    sinusShape=1;
else
    sinusShape=0;
end

end

function [distan,convex,order]=toothOrder(convex,concave)  %
calculate the number of orders of teeth
distan=[];
x1=concave(1:size(concave(:,1))-1,2);
y1=concave(1:size(concave(:,1))-1,1);
x2=concave(2:size(concave(:,1)),2);
y2=concave(2:size(concave(:,1)),1);
x3=convex(2:size(convex(:,1))-1,2);
y3=convex(2:size(convex(:,1))-1,1);
distan=abs((x2-x1).*y3-(y2-y1).*x3-(x2-x1).*y1+x1.*(y2-y1))./sqrt((x2-
x1).*(x2-x1)+(y2-y1).*(y2-y1));
%excluded the abnormal points
[n,m]=size(distan);
[rc,lc]=find(distan>2*mean(distan));
distan(rc(1:size(rc)))=[];
convex(rc(1:size(rc))+1,:)=[];
%normalization
%distan=distan/max(distan);
[rc,lc]=find(distan<mean(distan));
size_rc =size(rc);
if size_rc(1,1)>(n/2)
    distan(1,2)=2;
    order=2;
    for i=2:size(distan)-1
       if  (distan(i)>distan(i-1))&&(distan(i)>distan(i+1))
           distan(i,2)=1;
       else
           distan(i,2)=2;
       end
    end
    distan(size(distan),2)=2;
else
    %one order only
    order=1;
    distan(:,2)=1;
end

convex(2:size(convex(:,1))-1,3)=distan(:,2);

end

function flag=toothSpacing(convex)% Tooth spacing Regularor or
Irregular
     first=[];
     secord=[];
     firstdistan=[];
     secorddistan=[];
     [rc1,lc1]=find(convex(:,3)==1);
     [rc2,lc2]=find(convex(:,3)==2);
     %the distance between 1st order of teeth
     x1=convex(rc1(1:size(rc1)-1),2);
     y1=convex(rc1(1:size(rc1)-1),1);
     x2=convex(rc1(2:size(rc1)),2);
     y2=convex(rc1(2:size(rc1)),1);
     firstdistan=sqrt((x2-x1).*(x2-x1)+(y2-y1).*(y2-y1));
     %the distance between 2nd order of teeth
     x1=convex(rc2(1:size(rc2)-1),2);
     y1=convex(rc2(1:size(rc2)-1),1);
     x2=convex(rc2(2:size(rc2)),2);
     y2=convex(rc2(2:size(rc2)),1);
     secorddistan=sqrt((x2-x1).*(x2-x1)+(y2-y1).*(y2-y1));
     %判断是否是Regular的叶齿
     if isempty(rc2)
     %若仅有一级齿
         firstdistan=sort(firstdistan);
         [rc,lc]=size(firstdistan);
         %去除异常值点
         while firstdistan(rc)*0.7>firstdistan(rc-1)
             firstdistan(rc)=[];
             [rc,lc]=size(firstdistan);
             [rc,lc]=size(firstdistan);
         end
          d2=min(firstdistan);
          [rc,lc]=find(firstdistan==d2);
          firstdistan(rc)=[];
          [rc,lc]=size(firstdistan);
         if  rc>=20  %一级齿个数大于10

d1=mean(firstdistan(size(firstdistan)-15:size(firstdistan)-5));
         d2=mean(firstdistan(1:10));
         else
             d1=firstdistan(size(firstdistan));
             d2=firstdistan(1);
         end
     else
         %若有二级齿
         firstdistan=sort(firstdistan);
         secorddistan=sort(secorddistan);
         t=size(firstdistan);
         if size(firstdistan)>5 & size(secorddistan)>5 %一二级齿个数大于5
         d1=mean(firstdistan(size(firstdistan)-4:size(firstdistan)));
         d2=mean(secorddistan(1:5));
         else
             d1=firstdistan(size(firstdistan));
             d2=secorddistan(1);
         end
     end

     if d2>=d1*0.6
        flag=0;%Regular
     else
        flag=1;%Irregular
     end

end

function
toothShape=toothShapeFinder(curve1,convex,concave,I1,I)%calculate
Tooth shape

 x1=convex(1,2);y1=convex(1,1);
 x2=concave(1,2);y2=concave(1,1);
 curve=curve1{1,1};
%linear fitting
    [rc1,lc1]=find(curve(:,2)==x1);
    [rc2,lc2]=find(curve(:,1)==y1);
    j=intersect(rc1,rc2);%求交集
    [rc1,lc1]=find(curve(:,2)==x2);
    [rc2,lc2]=find(curve(:,1)==y2);
    k=intersect(rc1,rc2);
    d=abs(k-j)/2;
    m=2;

   if j<k
   x=curve(j:k,2);
   y=curve(j:k,1);
   [p,s]=polyfit(x,y,m);
   end
    flag=0;
    a1=p(1,1);
    a2=p(1,2);
    a3=p(1,3);
    if a1>-0.007&&a1<0.005
        flag=3;
    else
       crossPoints = [];
       number_crossPoint=0;
        yN=(y2-y1)/(x2-x1)*(x-x1)+y1;
        yN_int=round((y2-y1)/(x2-x1)*(x-x1)+y1);
        xN=x;
        % C point finding
        for index_line =round(2+(0.05*(k-j))):round(0.95*(k-j))
            for index_curve =2:k-j
                if (xN(index_line)==x(index_curve) &&
yN_int(index_line)==y(index_curve))
                    number_crossPoint = number_crossPoint+1;
                    crossPoints =
[crossPoints;index_line,index_curve,y(index_curve),x(index_curve)];
                end
            end
        end
    %%%%%%%%%%%%%%%%%%%%%%%%% showing the shape of the tooth
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         figure
%         plot(yN_int,xN);
%         hold on;
%         plot(y,x);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%
        size_cp = size(crossPoints);
        index_mid = round(size_cp(1,1)/2);
        [NumberWhite,NumberBlack,NumberWhiteCE,NumberBlackCE]
=deal(0);

        if isempty(crossPoints)==true
            %%  CrossPoint C exist
            for index_x = 3:k-j-1
              for indexImage_y = yN_int(index_x):y(index_x)
                  if I1(indexImage_y,x(index_x))==false
                    NumberBlack = NumberBlack + 1;
                  else
                    NumberWhite = NumberWhite + 1;
                  end
              end
            end
            if NumberWhite>NumberBlack
                flag =1; % concave
            else
                flag =2; % convex
            end

        else
        %%CrossPoint C  doesn't exist
%             hold on;
%             plot(crossPoints(:,3),crossPoints(:,4),'rp');

            for index_x = 3:crossPoints(index_mid,2)
              for indexImage_y = yN_int(index_x):y(index_x)
                  if I1(indexImage_y,x(index_x))==false
                    NumberBlack = NumberBlack + 1;
                  else
                    NumberWhite = NumberWhite +1;
                  end
              end
            end

            %%%%%%%%%%%%%%%%% The second part %%%%%%%%%%%%%%%%
            for index_x = crossPoints(index_mid,2)+2:k-j-1
              for indexImage_y = yN_int(index_x):y(index_x)
                  if I1(indexImage_y,x(index_x))==false
                    NumberBlackCE = NumberBlackCE + 1;
                  else
                    NumberWhiteCE = NumberWhiteCE +1;
                  end
              end
            end

            if (NumberWhite<NumberBlack &&
            if (NumberWhite<NumberBlack &&
NumberWhiteCE>NumberBlackCE)
                flag =21; % Fl
            elseif (NumberWhite>NumberBlack &&
NumberWhiteCE<NumberBlackCE)
                flag =12; % RT
            elseif (NumberWhite>NumberBlack &&
NumberWhiteCE>=NumberBlackCE)
                flag =1;
            elseif (NumberWhite<=NumberBlack &&
NumberWhiteCE<=NumberBlackCE)
                flag =2;

            end

        end
    end
    toothShape=flag;
end
```