目录

主函数代码

```
close all
clear all
clc

im1=imread('108_4.tif');
figure
imshow(im1);title('原图像');



I=normalize2(im1);%归一化和切割
figure
imshow(uint8(I));

im3=derection_bw(I);%二值化
figure
imshow(im3);

im4=thin(im3);%细化
figure
imshow(im4);

[x0,y0]=centralizing(double(im4));%找中心点
figure
```

```
imshow(im4);
hold on;plot(x0,y0,'go');hold off;

im4=~im4;
[x3,y3,x4,y4]=find_feature2(im4,x0,y0,100);%找特征点
figure
imshow(im4);hold on;
plot(x3,y3,'bo');plot(x4,y4,'ro');hold off ;
```

原图

1.归一化和切割

归一化调整的是指纹灰度均值和方差，调整到标准状态，同时可以屏蔽不必要的噪声。
归一化方法如下：

由于不同指纹区域的手指压力和强度不同， 所以将指纹分为 W*H 小块，设图像中像素点的灰度值为 I(i,j)，归一化后的图像 G(i,j) 来表示，灰度平均值和方差分别用 M_i 和 V_i 来表示，则归一化算法如下 [4]：

（1）先计算出图像灰度的平均值和方差：

$$M_i = \frac{1}{WH}\sum_{i=0}^{H-1}\sum_{j=0}^{W-1} I(i,j) \qquad (2\text{-}1)$$

$$V_i = \frac{1}{WH}\sum_{i=0}^{H-1}\sum_{j=0}^{W-1} (I(i,j)-M_i)^2 \qquad (2\text{-}2)$$

（2）指定期望的图像方差和平均值后，算出归一化后的图像 G(i,j)：

$$G(i,j)=\begin{cases} M_0 + \sqrt{\dfrac{V_0(I(i,j)-M_i)^2}{V_i}} & I(i,j) > M_i \\ M_0 - \sqrt{\dfrac{V_0(I(i,j)-M_i)^2}{V_i}} & I(i,j) \le M_i \end{cases} \qquad (2\text{-}3)$$

其中 M_0，V_0 为期望的平均值和方差（一般 M_0=150，V_0=2000）。

分割

分割图像基于块特征的指纹图像分割， 本文处理将采用均值方差法 [3]。该算法基于背景区灰度方差小，而指纹区方差大的思想，将指纹图像分成块，计算每一块的方差，如果该块的方差小于阈值为背景，否则为前景。具体步骤分以下三步：

(1)将图像分块，将其分为 N×N 的方块。

(2) 计算出每一块的均值和方差。

函数1：归一化和切割

```
function y=normalize2(x)
%对指纹图像进行归一化处理
x1=double(x);
[a,b,~]=size(x);
m=20;
a1=a/m;%行分块数
b1=b/m;%列分块数
M0=mean2(x1);
V=std2(x1);
V0=V*V;
M1=80;
V1=6000;
T=60;
```

```
a3=1;%块移动的行位置
b3=1;%块移动的列位置
for i=1:a
    for j=1:b
        if x1(i,j)>M0
            result(i,j)=M1+sqrt(V1*((x1(i,j)-M0)*(x1(i,j)-M0))/V0);
        else
            result(i,j)=M1-sqrt(V1*((x1(i,j)-M0)*(x1(i,j)-M0))/V0);
        end
    end
end

for k=1:a1
    for l=1:b1
        M2=mean2(result(a3:(a3+m-1),b3:(b3+m-1)));%均值
        V2=std2(result(a3:(a3+m-1),b3:(b3+m-1)));%方差
        V3=V2*V2;

        if V3<1000
            for i1=a3:a3+m-1
                for j1=b3:b3+m-1
                    result(i1,j1)=255;
                end
            end
        end

        if V2==0
            for i1=a3:a3+m-1
                for j1=b3:b3+m-1
                    result(i1,j1)=255;
                end
            end
        else
            Th=M2/V2;
            if Th>T
                for i1=a3:a3+m-1
                    for j1=b3:b3+m-1
                        result(i1,j1)=255;
                    end
                end
            end
        end
        b3=b3+m;
    end
    b3=1;
    a3=a3+m;
```

```
end
y=result;
```

归一化和切割效果：

2.二值化

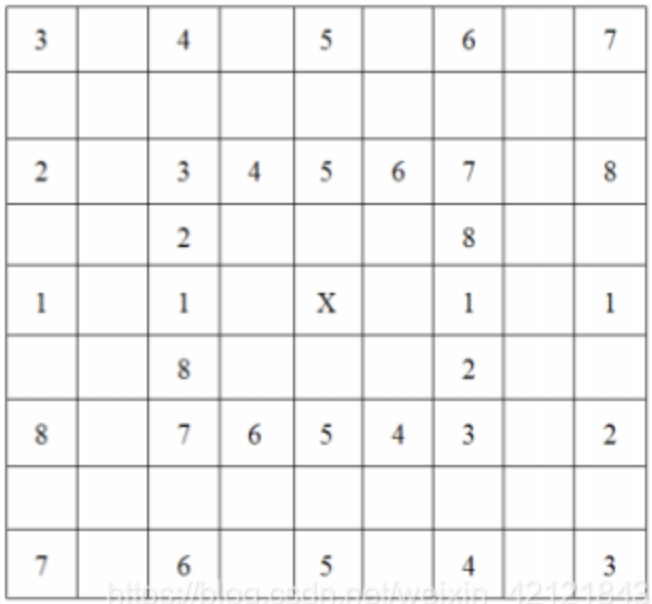
二值化方案一：局部阈值法

由于原始指纹图像不同区域深浅不一， 如对整幅图像用同一阈值进行二值分割， 会造成大量有用信息的丢失。 所以我们可以选用局部阈对图像进行二值化。局部阈值法即选取 N×N的块，求该区域的阈值并对该区域二值化，可以有效地保证信息的可靠性。

二值化方案二：基于方向场的二值化

采集到的指纹图像一般都有比较清晰的方向场，方向场估计得准确与否直接决定了图像二值化算法的效果。

为估计方向场，我们把指纹脊线的走向分为如下 8 个方向



3		4		5		6		7
2		3	4	5	6	7		8
		2				8		
1		1		X		1		1
		8				2		
8		7	6	5	4	3		2
7		6		5		4		3

先进行均值滤波，然后对图像的每一个像素，为确定在该像素处的脊线方向，在以该像素为中心的 *99 窗口内，分别计算 8*
*个方向上的经过处理后的灰度值，即将图中数字 1 到 8 的位置的像素灰度值去除其中最大 summax和最小值 summin， 若满足最大的 summax 和最小的 summin 与 4*I(x,y)) 之和大于 (3*summ/8), 则该像素点的脊线方向为summin,否则为 summax。确定完脊线方向后再由该方向场对图像进行二值化。

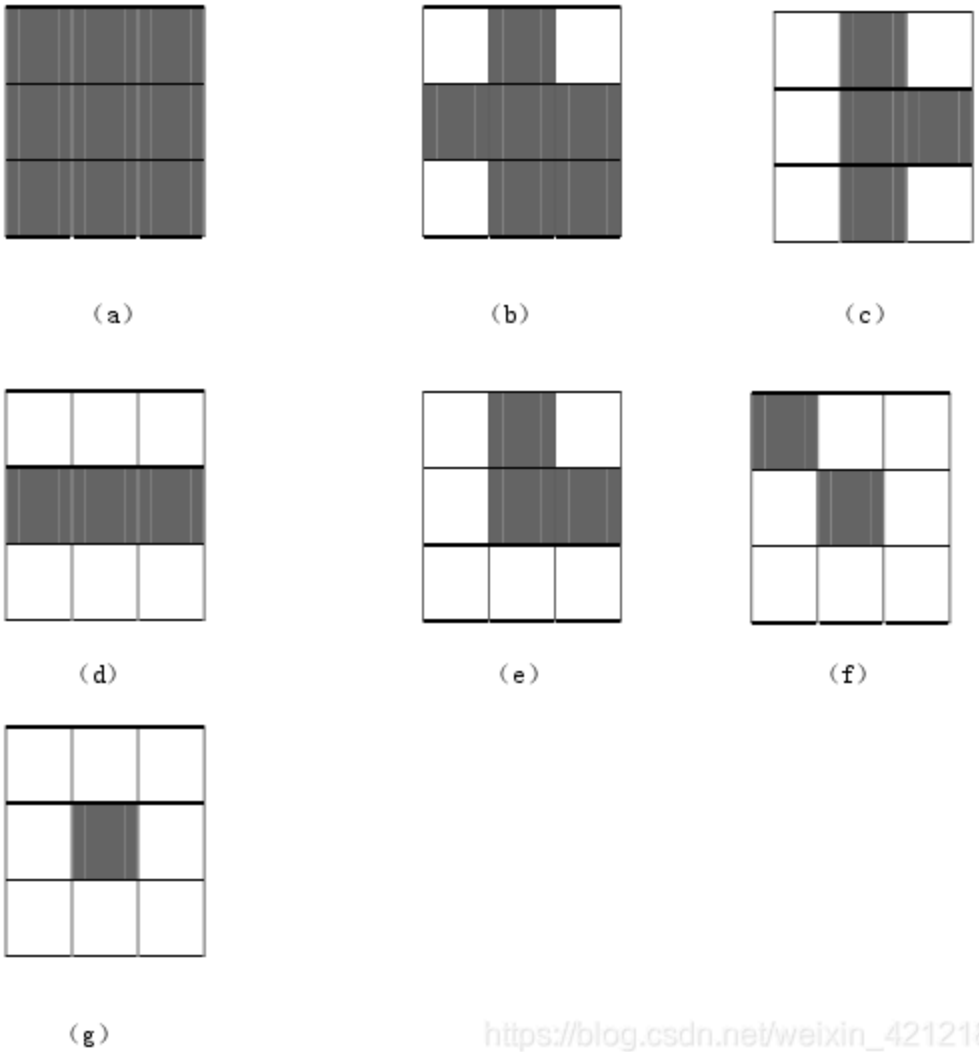
```
function im_out=derection_bw(I)
temp=(1/9)*[1 1 1;1 1 1;1 1 1]; % 模板系数、均值滤波
[m,n,~]=size(I);
Im=double(I);
In=zeros(m,n);
Icc=ones(m,n);
for a=2:m-1
    for b=2:n-1
        In(a,b)=Im(a-1,b-1)*temp(1,1)+Im(a-1,b)*temp(1,2)+Im(a-1,b+1)*temp(1,3)+Im(a,b-1)*t
    end
end
I=In;
Im=zeros(m,n);

for x=5:m-5
    for y=5:n-5
        sum1=I(x,y-4)+I(x,y-2)+I(x,y+2)+I(x,y+4);
        sum2=I(x-2,y+4)+I(x-1,y+2)+I(x+1,y-2)+I(x+2,y-4);
        sum3=I(x-2,y+2)+I(x-4,y+4)+I(x+2,y-2)+I(x+4,y-4);
        sum4=I(x-2,y+1)+I(x-4,y+2)+I(x+2,y-1)+I(x+4,y-2);
        sum5=I(x-2,y)+I(x-4,y)+I(x+2,y)+I(x+4,y);
        sum6=I(x-4,y-2)+I(x-2,y-1)+I(x+2,y+1)+I(x+4,y+2);
        sum7=I(x-4,y-4)+I(x-2,y-2)+I(x+2,y+2)+I(x+4,y+4);
```

```
        sum8=I(x-2,y-4)+I(x-1,y-2)+I(x+1,y+2)+I(x+2,y+4);
        sumi=[sum1,sum2,sum3,sum4,sum5,sum6,sum7,sum8];
        summax=max(sumi); summin=min(sumi); summ=sum(sumi);
        b=summ/8;
        if (summax+summin+ 4*I(x,y))> (3*summ/8)
            sumf = summin;
        else
            sumf =summax;
        end
        if sumf > b
            Im(x,y)=128;
        else
            Im(x,y)=255;
        end
    end
end
for i=1:m
    for j =1:n
        Icc(i,j)=Icc(i,j)*Im(i,j);
    end
end
for i=1:m
    for j =1:n
        if (Icc(i,j)==128)
            Icc(i,j)=0;
        else
            Icc(i,j)=1;
        end
    end
end
im_out=Icc;
```

二值化效果：

3.细化

细化过程中要根据每个像素的八个相邻点的情况来判断该点是否可以剔除或保留，其实质类似于腐蚀操作



从上图可以看出：(a)图不能移除，因为它是个内部点，我们要求的是骨架，如果连内部点也移除了，骨架也会被掏空的；与(a) 相同，(b)不能移除；(c)可以移除，这样的点不是骨架；(d)不能移除，因为移除掉后，原来相连的部分断开了；(e)可以移除，这样的点不是骨架；(f)不能移除，因为它是直线的端点，如果这样的点移除了，那么最后整个直线也被移除了，剩不下什么；(g)不能移除，因为孤立点的骨架就是它自身。根据上图我们可以归纳出如下结论：内部点、孤立点和直线端点都不能移除；当 P 为边界点时，如果去掉 P 后不增加连通分量则可以移除 P 点。

代码实现：

```
function im_out = thin2(I)%有孤岛待去除
I=im3;
[M,N]=size(I);
for i=2:M-1
    for j=2:N-1
        if I(i,j)==0
            if (I(i-1,j)==0&&I(i,j+1)==0)||(I(i-1,j)==0&&I(i,j-1)==0)||(I(i+1,j)==0&&I(i,j-
                I(i,j)=1;
```

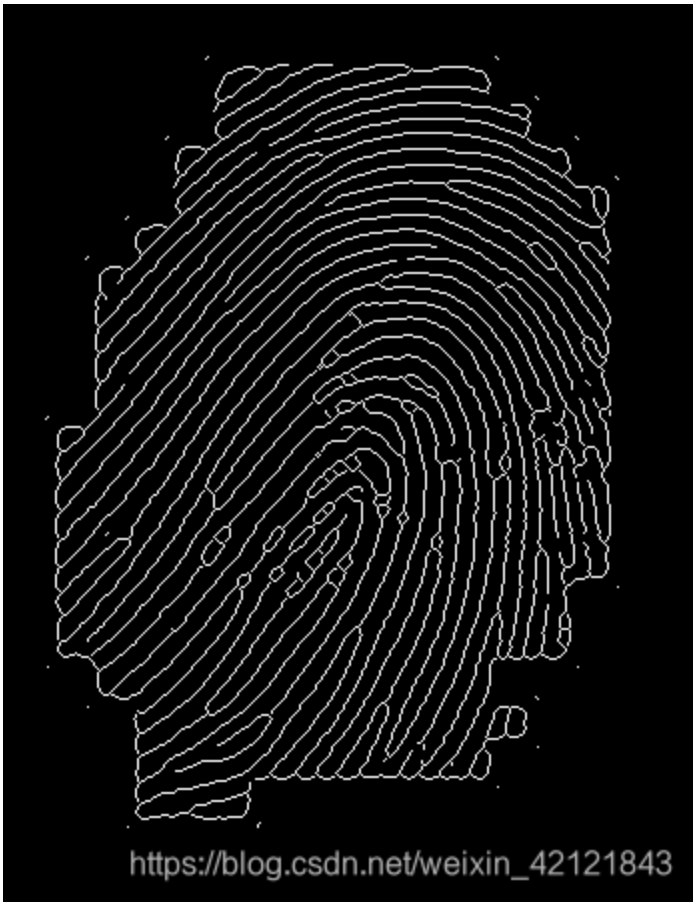
```
            else
                I(i,j)=0;
            end
        end
    end
end
end
```

细化算法1结果

经实验，效果不佳，于是采用MATLAB自带的bwmorph函数进行细化，除此之外，还加上了去除毛刺和空洞的部分

```
function y=thin(x)
I=ordfilt2(x,5,ones(3,3));%对指纹图像进行3×3滤波
u=I;
[m,n]=size(u)
for x=2:m-1
    for y=2:n-1
        if u(x,y)==0
            if u(x,y-1)+u(x-1,y)+u(x,y+1)+u(x+1,y)>=3
                u(x,y)=1;
            else
                u(x,y)=u(x,y);
            end
        end
    end
end
for a=2:m-1
    for b=2:n-1
        if u(a,b)==1
            if abs(u(a,b+1)-u(a-1,b+1))+abs(u(a-1,b+1)-u(a-1,b))+abs(u(a-1,b)-u(a-1,b-1))+a
                if (u(a,b+1)+u(a-1,b+1)+u(a-1,b))*(u(a,b-1)+u(a+1,b-1)+u(a+1,b))+(u(a-1,b)+
                    u(a,b)=0;
                end
            end
        end
    end
end
v=~u;
w=bwmorph(v,'thin',Inf);%对图像进行细化
for x=2:m-1
    for y=2:n-1
        if w(x,y)==1
            if (w(x-1,y)==1&&w(x,y-1)==1)||(w(x-1,y)==1&&w(x,y+1)==1)||(w(x,y-1)==1&&w(x+1,
                w(x,y)=0;
            end
        end
    end
end
y=w;
```

细化算法2效果：



4.中心点提取

提取指纹的中心点有很多作用，比如可以把指纹扇形化后变成特征图，用于指纹匹配。这里求中心点只是为了去除指纹边缘的伪特征点。

中心点定义为指纹的脊曲线曲率最大的点，在基于点模式的匹配算法中，将中心点作为匹配的参考点具有一致性比较强且效率高的特点。因此中心点的确定对后面的指纹识别过程十分重要。

本次研究中对指纹图像中心点的确定是在指纹图像二值化以后进行的，这更有利于对中心点确定的准确性。它的具体算法过程如下：

(1)求取指纹图像的块方向 O(i, j)；

(2)在邻域内将 O(i, j)进行平滑(低通滤波)，得到平 滑后的块方向图 O(i, j)。

首先将 O(i, j)表示为 X 方向和 Y 方向的连续向量，也就是计算 O(i, j)在 X 坐标方向和 Y 坐标方向的分量，具体见公式(3-3)和公式(3-4)：

$$\Phi x(i,j)=\cos(2O(i,j)) \quad \text{式(3-3)}$$

$$\Phi y(i,j)=\sin(2O(i,j)) \quad \text{式(3-4)}$$

然后分别对 Φx 和 Φy ，进行低通滤波，滤波方式见公式(3-5)和(3-6)：

$$\Phi'x(i,j)=\sum[-W_\Phi/2\le u\le W_\Phi/2]\sum[-W_\Phi/2\le v\le W_\Phi/2]W(u,v)\cdot\Phi x(i-uw,j-vw) \quad \text{式(3-5)}$$

$$\Phi'y(i,j)=\sum[-W_\Phi/2\le u\le W_\Phi/2]\sum[-W_\Phi/2\le v\le W_\Phi/2]W(u,v)\cdot\Phi y(i-uw,j-vw) \quad \text{式(3-6)}$$

其中 W(u, v)代表二维的低通滤波器，其大小为 $W_\omega \times W_\omega$。研究中去的是 8×8 的这样，平滑后的块方向图表示为 O’ (i, j)，其参考公式如公式(3-7)

$$O'(i,j)=1/2*\arctan(\Phi'y(i,j)/\Phi'x(i,j)) \quad \text{式(3-7)}$$

(3) 计算 Φ'(i,j)的 sine 分量 ε(i,j), 即公式(3-8)：

$$\varepsilon(i,j)=\sin(O'(i,j)) \quad \text{式(3-8)}$$

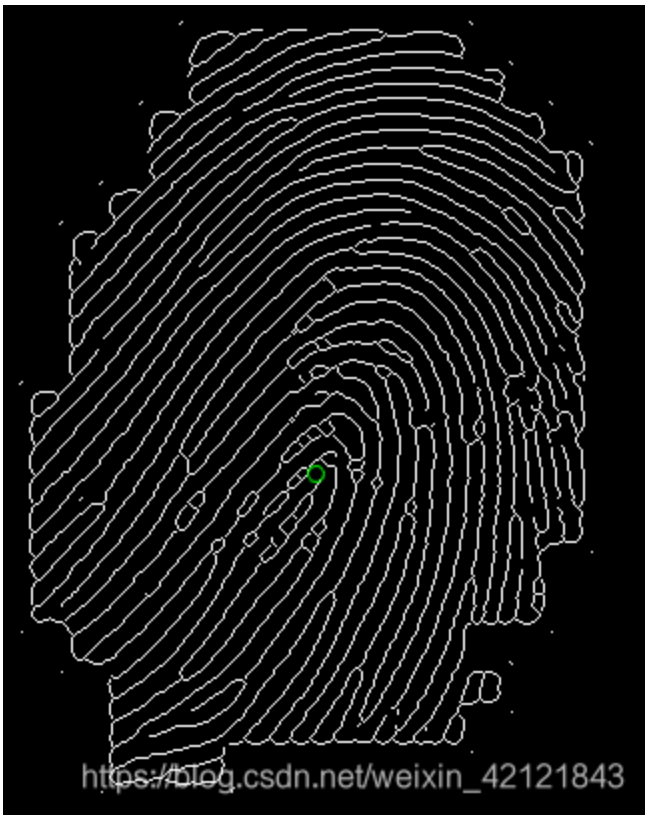
代码：

```
function [XofCenter,YofCenter]=centralizing(fingerprint)
imgN=size(fingerprint,1);
imgM=size(fingerprint,2);
image=wiener2(fingerprint,[3 3]);
[Gx,Gy]=gradient(image);
orientnum=wiener2(2.*Gx.*Gy,[3 3]);
orientden=wiener2((Gx.^2)-(Gy.^2),[3 3]);
W=8;
l1=9;
orient=zeros(imgN/W,imgM/W);
points=(imgN/W)* (imgM/W);
for i=1:1:points
    x=floor((i-1)/(imgM/W))* W+1;
    y=mod(i-1,(imgN/W))* W+1;
    numblock=orientnum(y:y+W-1,x:x+W-1);
    denblock=orientden(y:y+W-1,x:x+W-1);
    somma_num=sum(sum(numblock));
    somma_denom=sum(sum(denblock));
    if somma_denom~=0
        inside = somma_num/somma_denom;
        angle=0.5*atan(inside);
    else
        angle=pi/2;
    end
 %each block
 if angle<0
     if somma_num<0
         angle=angle+pi/2;
     else
         angle=angle+pi;
     end
 else
     if somma_num>0
         angle=angle+pi/2;
     end
 end
 orient(1 +(y-1)/W,1+(x-1)/W)=angle;
```

```
end
binarize=(orient<pi/2);
[bi,bj]=find(binarize);
xdir=zeros(W,W);
ydir=zeros(W,W);
for k=1:1:size(bj,1)
    i=bj(k);
    j=bi(k);
    if orient(j,i)<pi/2
        x=fix(l1*cos(orient(j,i)-pi/2)/(W/2));
        y=fix(l1*sin(orient(j,i)-pi/2)/(W/2));
        xdir(j,i)=i-x;
        ydir(j,i)=j-y;
    end
end
binarize2=zeros(imgN/W,imgM/W);
for i=1:1:size(bj,1)
    x=bj(i);
    y=bi(i);
    if~(xdir(y,x)<1||ydir(y,x)<1||xdir(y,x)>imgM/W||ydir(y,x)>imgN/W)
        while binarize(ydir(y,x),xdir(y,x))>0
            xtemp=xdir(y,x);
            ytemp=ydir(y,x);
            if xtemp<1||ytemp<1||xtemp>imgM/W||ytemp>imgN/W
                break;
            end
            x=xtemp;
            y=ytemp;
            if xdir(y,x)<1||ydir(y,x)<1||xdir(y,x)>imgM/W||ydir(y,x)>imgN/W
                if x-1>0
                    while binarize(y,x-1)>0
                        x=x-1;
                        if x-1<1
                            break;
                        end
                    end
                end
                break;
            end
        end
    end
    binarize2(y,x)=binarize2(y,x)+1;
end
[temp,y]=max(binarize2(1:end-7,:));
[temp2,x]=max(temp);
angle=orient(y(x),x)-pi/2;
```

```
XofCenter=round(x*W-(W/2)-(l1/2)*cos(angle));
YofCenter=round(y(x)*W-(W/2)-(l1/2)*sin(angle));
%Outputprint=binarize2;
end
```

求中心点效果：



5.求特征点

这里我们用最常用的两类特征点：端点和分叉点
找点方法是模板匹配法。

终结点：一条指纹线路在终结点终结，如图 2-9 所示。

图 2-9 终结点

分叉点：一条指纹线路在此分开成两条或更多的指纹线路，如图 2-10 所示。

图 2-10 分叉点

(a)分叉点　　(b)终结点

断点和分叉点（如图 3.13）是指纹细化图象的主要特征，本文采用这两种主要特征构造指纹特征向量，它的提取方法是模板匹配法。模板匹配法有运算量小、速度快的优点[29-36]。

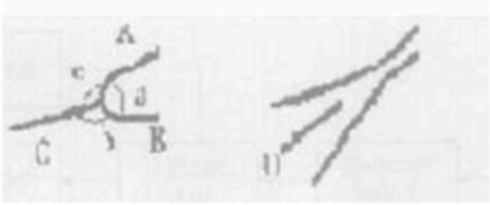


图 3.13　断点和分叉点

在八领域的所有状态中，满足断点特征条件的有 8 种，满足分叉点特征条件的 9 种分别如图 3.14 和图 3.15 所示：

图 3.14　断点模板

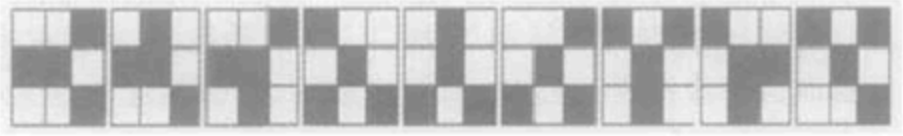


图 3.15　分叉点模板

若细化不完全，会有很多点会误识别为特征点

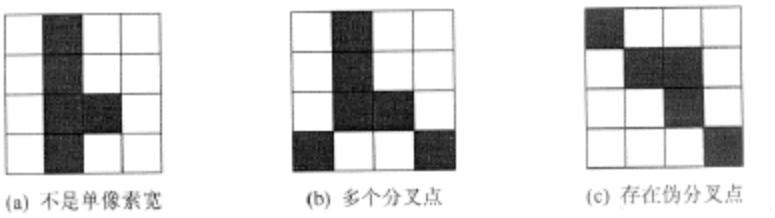
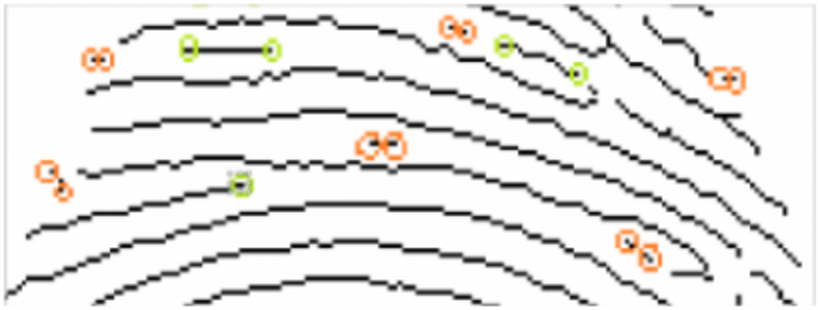


图 3.9　分叉点处细化不彻底的 3 种情况

去除伪特征点：

因为采集、预处理过程中的空因素和提取算法的原因，提取到的特征点中会存在很多为特征点。 为了不影响后续的识别， 必须进行剔除。 本程序中剔除的伪特征点主要有两种：指纹边缘点，断点，并根据这三种伪特征点的特征进行剔除。两类伪特征点如下（红色为伪特征点）

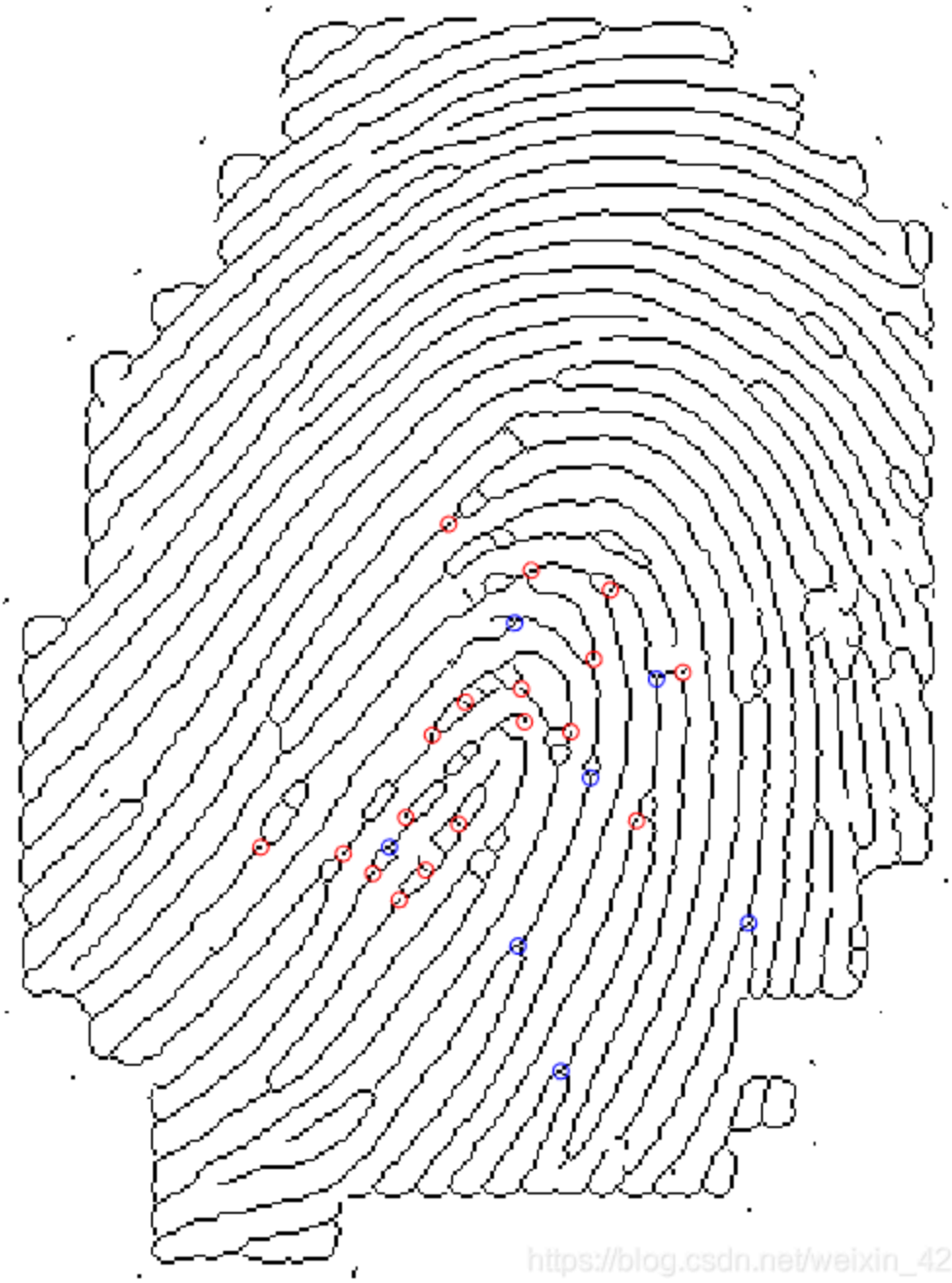


代码

```
function [X1,Y1,X2,Y2]=find_feature2(I,x0,y0,radius)
[M,N]=size(I);
t=0;
k=0;
for i=2:M-1
    for j=2:N-1
        if I(i,j)==0
            n=I(i-1,j-1)+I(i-1,j)+I(i-1,j+1)+I(i,j-1)+I(i,j+1)+I(i+1,j-1)+I(i+1,j)+I(i+1,j+
            if (n==5)%分叉点
                t=t+1;
                x1(t)=j;
                y1(t)=i;
```

```
            end
            if (n==7)%端点
                k=k+1;
                x2(k)=j;
                y2(k)=i;
            end
        end
    end
end
%去除距离较近的特征点
for i=1:t-1
    for j=i+1:t
        d=sqrt((x1(i)-x1(j))^2+(y1(i)-y1(j))^2);
        if d<20
            x1(i)=-1;y1(i)=-1;x1(j)=-1;y1(j)=-1;
        end
    end
end
for i=1:k-1
    for j=i+1:k
        d=sqrt((x2(i)-x2(j))^2+(y2(i)-y2(j))^2);
        if d<10
            x2(i)=-1;y2(i)=-1;x2(j)=-1;y2(j)=-1;
        end
    end
end
%保留中心点半径之内的点
c=1;
for i=1:t
    d(i)=sqrt((x1(i)-x0)^2+(y1(i)-y0)^2);
    if(d(i)<radius)
    X1(j)=x1(i);
    Y1(j)=y1(i);
    j=j+1;
    end
end
j=1;
for i=1:k
    d(i)=sqrt((x2(i)-x0)^2+(y2(i)-y0)^2);
    if(d(i)<radius)
    X2(j)=x2(i);
    Y2(j)=y2(i);
    j=j+1;
    end
end
end
```

求特征点效果：（红色为端点，蓝色为分叉点）



7.匹配（未完待续）