数字图像处理作业 3
边缘检测与边缘追踪
实验报告

姓名 刘扬
学号 171850524
邮箱 541446436@qq.com
联系方式 188 3289 9861
(南京大学 计算机科学与技术系 南京 210093)

1．实验思路

实验主要通过实现 Canny 边缘检测器来进行边缘检测， 之后使用广度优先搜索遍历边缘检测的结果，对连通的边缘赋予同一种颜色，实现边缘追踪．

1.1 边缘检测

Canny 边缘检测主要包括五个步骤：高斯函数平滑图像，计算梯度，非最大抑制，滞后阈值，连通性分析．将后面两个步骤合并，共实现四个方法．

(1)高斯函数平滑输入图像

包括两部分：构造高斯滤波的空间模板，对输入图像进行平滑．

根据参数σ确定空间模板的大小为 2σ+1，以(σ+1,σ+1)作为中心点坐标，通过函数

$$G(x,y)=e^{\frac{(x-(\sigma+1))^2+(y-(\sigma+1))^2}{2\sigma^2}}$$

填充空间滤波模板． 若σ非整数， 需要控制模板大小是不小于(2σ+1)的奇数， 再确定中心位置．填充好模板之后，对模板中数据进行归一化，得到最终的滤波模板．

之后遍历图像的每一点进行平滑，得到平滑后的图像．

```
function img_out = GaussianFilter(img_in, sigma)
% 1.construct a Gaussian template
n = ceil(2*sigma+1);
if mod(n,2) == 0
    n = n+1;
end
center = (n+1)/2;
gaussianFilter = zeros(n,n);
denominator1 = 2*pi*sigma*sigma;
denominator2 = 2*sigma*sigma;
weightSum = 0;
for i = 1:n
    for j = 1:n
        numerator = (i-center)^2+(j-center)^2;
        gaussianFilter(i,j)=exp((-1)*numerator/denominator2)/denomi
        weightSum = weightSum + gaussianFilter(i,j);
    end
end
% 2.convolution
gaussianFilter = gaussianFilter / weightSum;
[w,h] = size(img_in);
img_expand = zeros(w+n-1, h+n-1);
img_expand(center:center+w-1, center:center+h-1)=img_in(1:w,1:h);
img_out = zeros(w,h);
for i = 1:w
   for j = 1:h
       img_out(i,j) = sum(sum(img_expand(i:i+n-1,j:j+n-1).*gaussiar
   end
end
end
```

(2)计算梯度

梯度计算较为简单， 对于每一点， 分别计算两个方向的梯度的平方和再开方就可以得到梯度的大小，梯度方向的计算通过 atan2(dy,dx)得到，以弧度制表示.

```
function [gradients, angle, dx, dy] = Gradients(img_in)
[w,h] = size(img_in);
gradients = zeros(w,h);
angle = zeros(w,h);
dx = zeros(w,h);
dy = zeros(w,h);
for i = 1:w-1
    for j = 1:h-1
        dx(i,j) = img_in(i+1,j)-img_in(i,j);
        dy(i,j) = img_in(i,j+1)-img_in(i,j);
        gradients(i,j) = sqrt(dx(i,j)^2+dy(i,j)^2);
        angle(i,j) = atan2(dy(i,j),dx(i,j));
    end
end
end
```

(3)非最大抑制

根据梯度的方向，把梯度大致划分在四个方向上：水平，竖直，正 45°，负 45°.

沿梯度方向，若当前点的梯度值不小于两个邻居点，则保留梯度值，否则置为 0.

简单的把梯度划分在四个方向会带来一定的误差， 一种改进的方式是沿梯度方向使用临界点的梯度值进行插值， 得到两个插值后的梯度， 与当前点梯度值进行比较， 确定取舍. 实际效果会发现改进的方法会导致边缘略微变粗.

```
for i = 2:w-1
   for j = 2:h-1
       t = angle(i,j);
       if (t<=pi/8&&t>=-pi/8) || t>=pi*7/8 || t<=-pi*7/8
           if g(i,j)>=g(i-1,j) && g(i,j)>=g(i+1,j)
               newGradients(i,j)=g(i,j);
           else
               newGradients(i,j)=0;
           end
       elseif (t>=pi/8&&t<=pi*3/8) || (t>=-pi*7/8&&t<=-pi*5/8)
           if g(i,j)>=g(i-1,j-1) && g(i,j)>=g(i+1,j+1)
               newGradients(i,j)=g(i,j);
           else
               newGradients(i,j)=0;
           end
       elseif (t>=pi*3/8&&t<=i*5/8)||(t>=-pi*5/8&&t<=-pi*3/8)
           if g(i,j)>=g(i,j-1) && g(i,j)>=g(i,j+1)
               newGradients(i,j)=g(i,j);
           else
               newGradients(i,j)=0;
           end
       else
           if g(i,j)>=g(i-1,j+1) && g(i,j)>=g(i+1,j-1)
               newGradients(i,j)=g(i,j);
           else
               newGradients(i,j)=0;
           end
       end
   end
end
```

(4)滞后阈值与连通性分析

设置两个阈值之比为 2.5:1，低阈值需要人工指定.

梯度值小于低阈值，则梯度值置为 0.

梯度值大于高阈值，则梯度值置为 1.

介于两者之间， 若当前点的 8 邻域内有梯度值大于高阈值的点， 则梯度值置为 1， 否则置为 0.

```
function res_edge = ConnectivityAnalyse(edge,theta)
TL = theta*max(max(edge));
TH = 2.5*TL;
[w,h] = size(edge);
res_edge = zeros(w,h);
tmp_edge = zeros(w+2,h+2);
tmp_edge(2:w+1, 2:h+1) = edge(1:w,1:h);
for i = 2:w+1
   for j = 2:h+1
       if tmp_edge(i,j)<=TL
           res_edge(i-1,j-1)=0;
       elseif tmp_edge(i,j)>=TH
           res_edge(i-1,j-1)=1;
       else
           if tmp_edge(i-1,j-1)>=TH||tmp_edge(i-1,j)>=TH||tmp_
               res_edge(i-1,j-1)=1;
           end
       end
   end
end
```

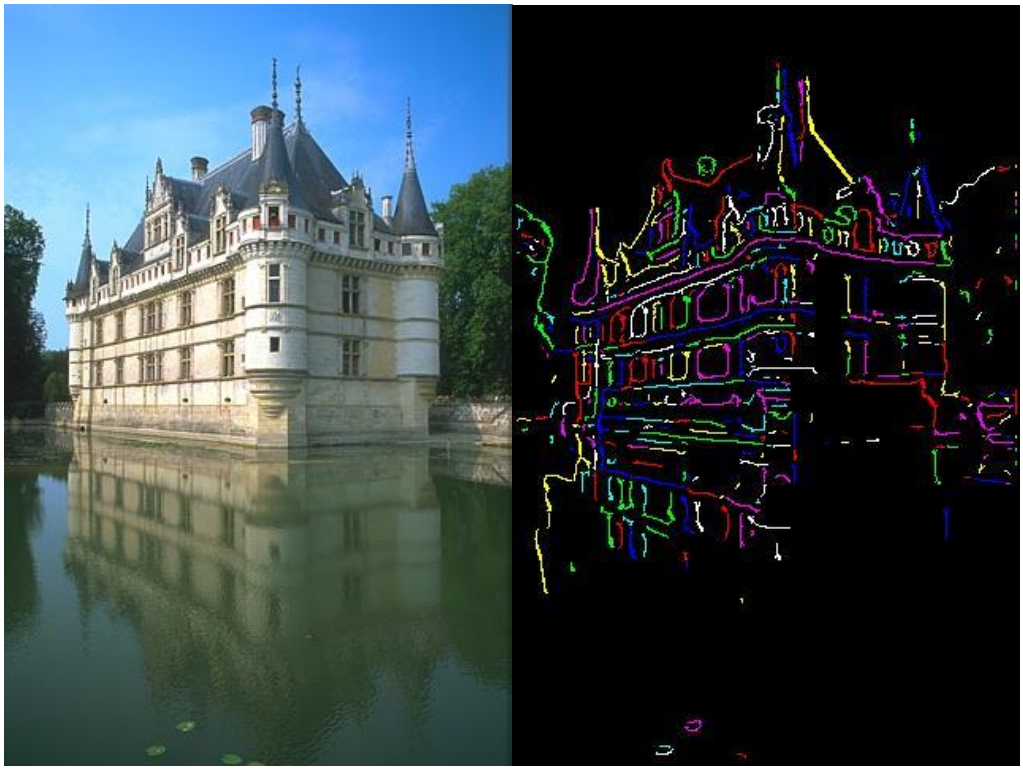
1.2 边缘追踪

事先定义 8 种颜色，那么有 7 种颜色可供边缘使用，遍历整个二值图像，对于遇到的边缘依次使用一种颜色进行染色， 7 种颜色可循环重复使用． 边缘追踪使用广度优先搜索完成．

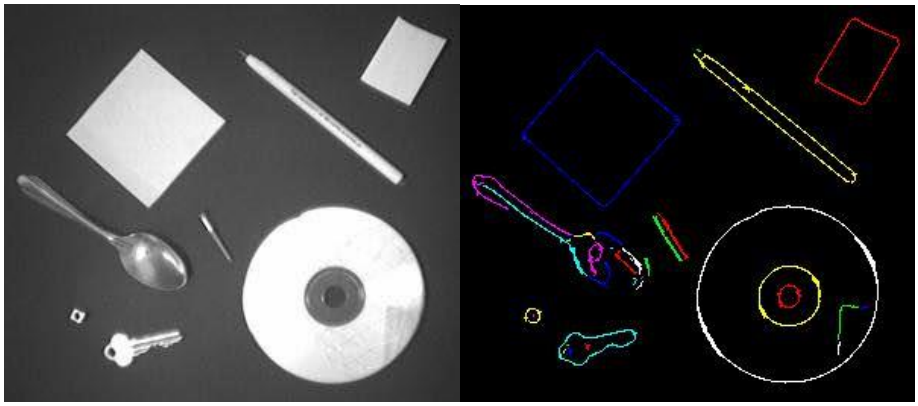
```
for i = 2 : w-1
   for j = 2 : h-1
       if edge(i,j) == 1 && color(i,j) == 0
          queue(1,1)=i; queue(1,2)=j;
          color(i,j) = 1;
          p = 1; q = 2;
          while p < q
              % queue pop
              x = queue(p,1);
              y = queue(p,2);
              p = p+1;
              color(x,y) = 2;
              newEdge(x,y)=cnt+1;
              % check 8-neighbor
              c=[x-1,y-1;x-1,y;x-1,y+1; x,y-1;x,y+1; x+1,y-1;x+1,
              for z = 1:8
                  c1 = c(z,1); c2 = c(z,2);
                  if edge(c1,c2)==1 && color(c1,c2)==0
                      % queue push
                     queue(q,1)=c1; queue(q,2)=c2;
                     q=q+1; color(c1,c2)=1;
                  end
              end
          end
          cnt = mod(cnt+1, num);
       end
   end
end
newEdge = ind2rgb(uint8(newEdge), map);
```

2．实验结果

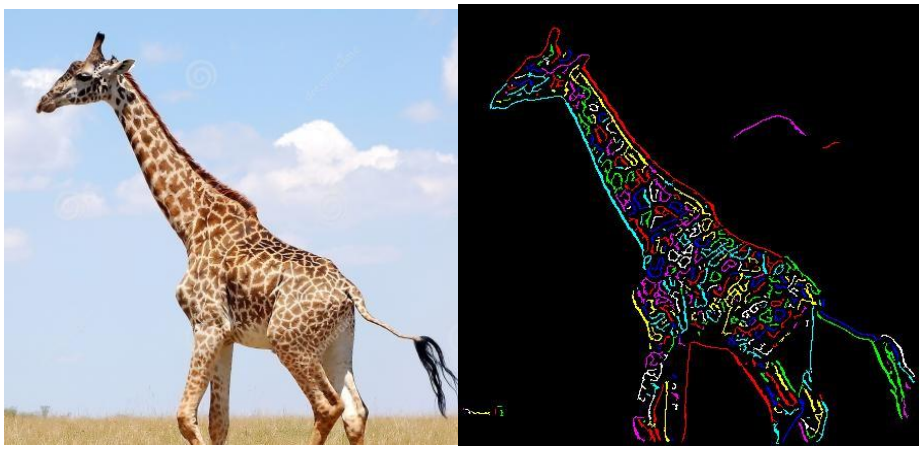
(1) castle



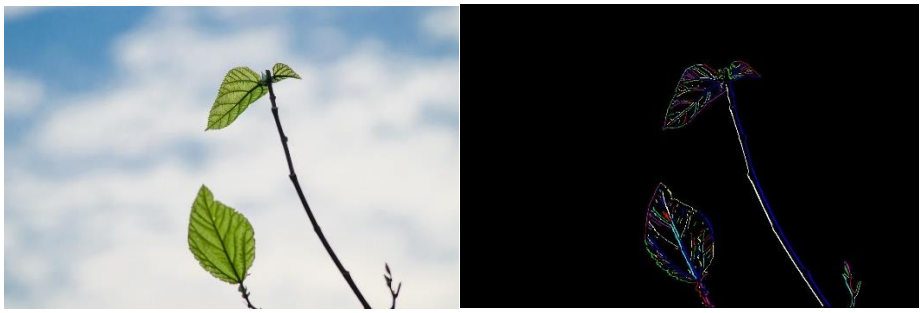
(2) disk



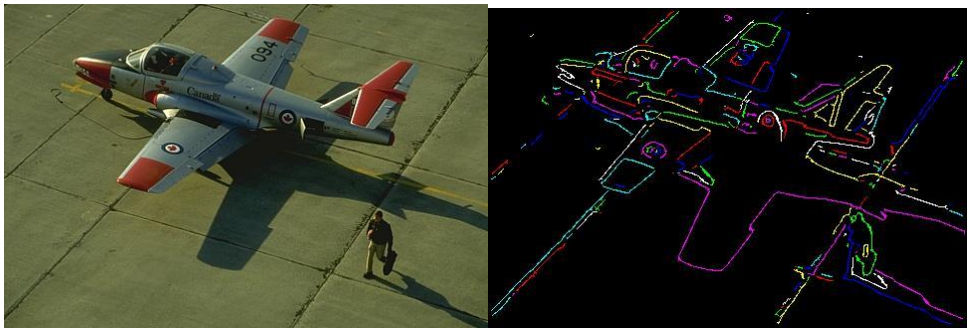
(3) giraffe



(4) leaf



(5) plane



(6) rubberband_cap