作业一 直方图均衡化

吴政亿 151220129 nju_wzy@163.com

(南京大学 计算机科学与技术系, 南京 210093)

边缘检测

在边缘检测中，我使用了`sobel`，`roberts`，`prewitt`，`marr`作为算子，对每一种算子调参，并进行了比较。

主体框架(marr单独介绍)

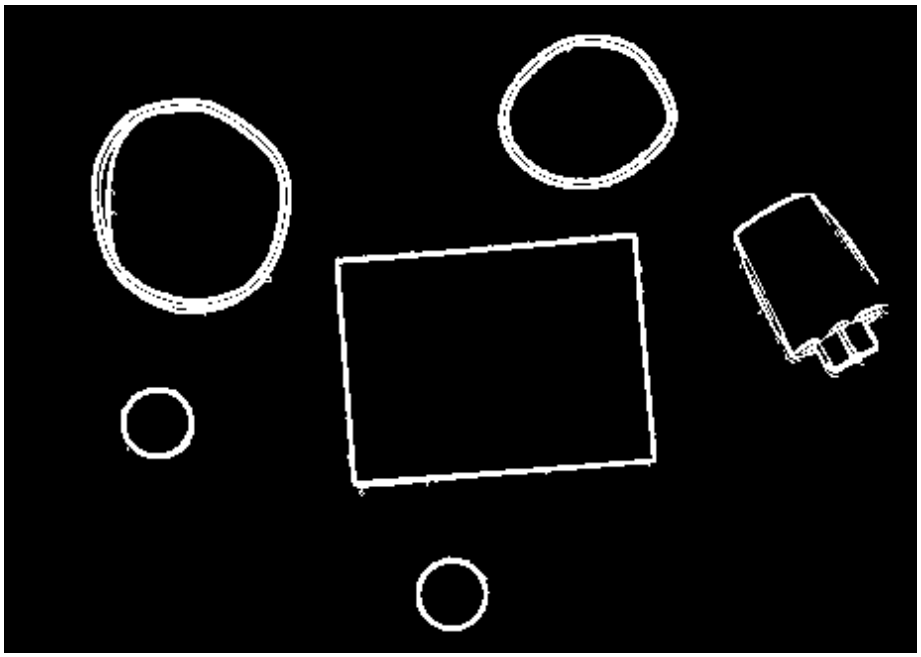
在`my_edge.m`中13~24行分别对应三个算子的high与low参数，大于high的被识别为边界，小于low的则被识别为非边界，在low~high之间的则根据邻域是否有边界来决定是否是边界。这样可以排除一部分的孤立点与噪声。

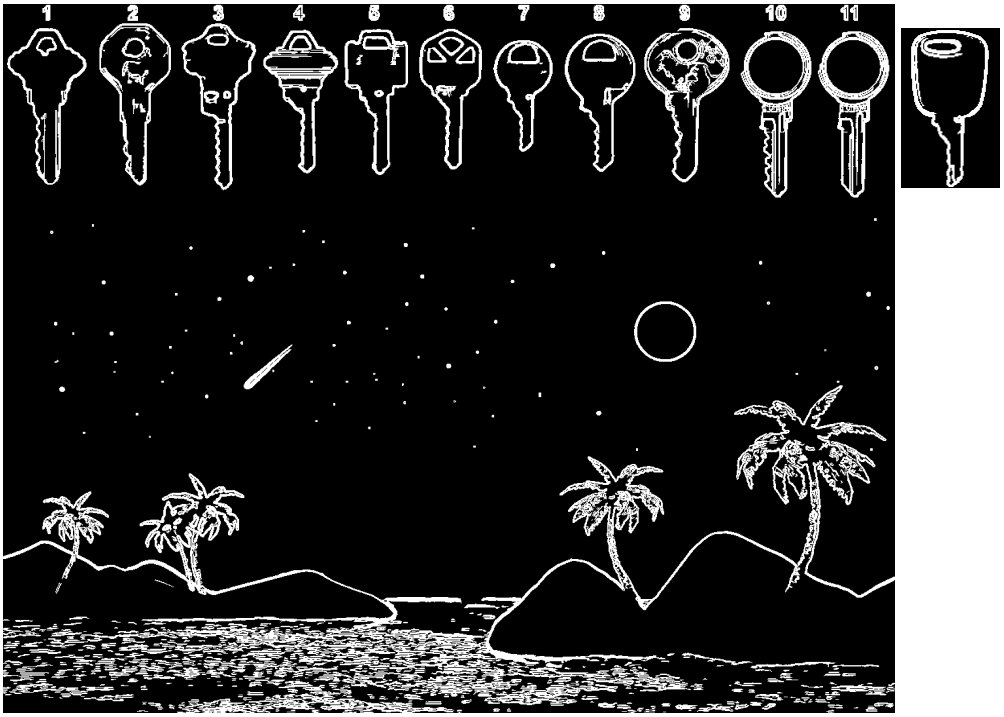
```
function output = my_edge(input_image)
    input_image=im2double(input_image);
    [M,N]=size(input_image);
    SX = zeros(M,N);
    SY = zeros(M,N);
%     %sobel
%     high = 0.3;   % moon:0.3  others:1.5
%     low = 0.2;    % moon:0.2  others:0.5

%     %roberts
%     high = 0.09;  % moon:0.09 key:0.4     cap:0.5
%     low = 0.05;   % moon:0.05 others:0.2

    %prewitt
    high = 0.25;    % moon:0.25 others:0.8
    low = 0.1;      % moon:0.1  others:0.5

    gray_image = zeros(M-2,N-2);
    for i=2:M-1
        for j = 2:N-1
%             gray_image(i-1,j-1) = sobel(input_image(i-1,j-1),input_image
(i-1,j),input_image(i-1,j+1),input_image(i,j-1),input_image(i,j),input_image
(i,j+1),input_image(i+1,j-1),input_image(i+1,j),input_image(i+1,j+1));
%             gray_image(i-1,j-1) = roberts(input_image(i-1,j-1),input_image
(i-1,j),input_image(i-1,j+1),input_image(i,j-1),input_image(i,j),input_image
(i,j+1),input_image(i+1,j-1),input_image(i+1,j),input_image(i+1,j+1));
            gray_image(i-1,j-1) = prewitt(input_image(i-1,j-1),input_image
(i-1,j),input_image(i-1,j+1),input_image(i,j-1),input_image(i,j),input_image
(i,j+1),input_image(i+1,j-1),input_image(i+1,j),input_image(i+1,j+1));
        end
    end

    edge_image = zeros(M-2,N-2);
```

```
        for j = 1:N-2
            if gray_image(i,j) > high
                edge_image(i,j) = 1;
            elseif gray_image(i,j) < low
                edge_image(i,j) = 0;
            elseif i==1 || j ==1 || i == M-2 || j == N-2
                edge_image(i,j) = 0;
            else
                edge_image(i,j) = -1;
            end
        end
    end

    for i = 2:M-3
        for j = 2:N-3
            if edge_image(i,j) == -1
                if edge_image(i-1,j-1) == 1 ||edge_image(i-1,j) == 1 ||edge_image
(i-1,j+1) == 1 ||edge_image(i,j-1) == 1 ||edge_image(i,j+1) == 1 ||edge_image
(i+1,j-1) == 1 ||edge_image(i+1,j) == 1 ||edge_image(i+1,j+1) == 1
                    edge_image(i,j) = 1;
                else
                    edge_image(i,j) = 0;
                end
            end
        end
    end
output = logical(edge_image);
```

sobel算子

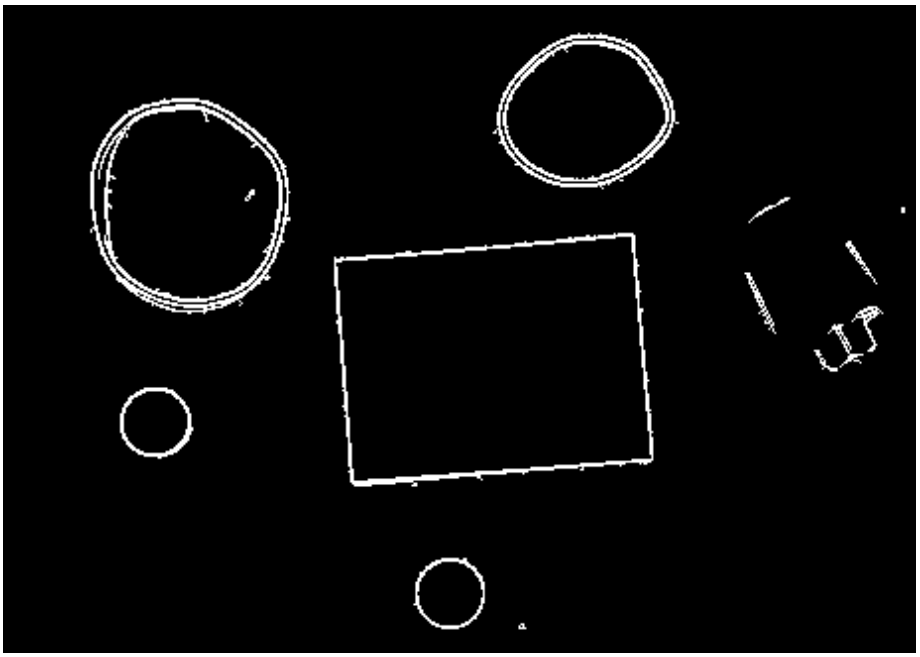
```
function sobel_value = sobel(v1,v2,v3,v4,v5,v6,v7,v8,v9)
    SX= -v1+v3-2*v4+2*v6-v7+v9;
    SY= v1+2*v2+v3-v7-2*v8-v9;
    sobel_value = sqrt(SX^2 + SY^2);
```

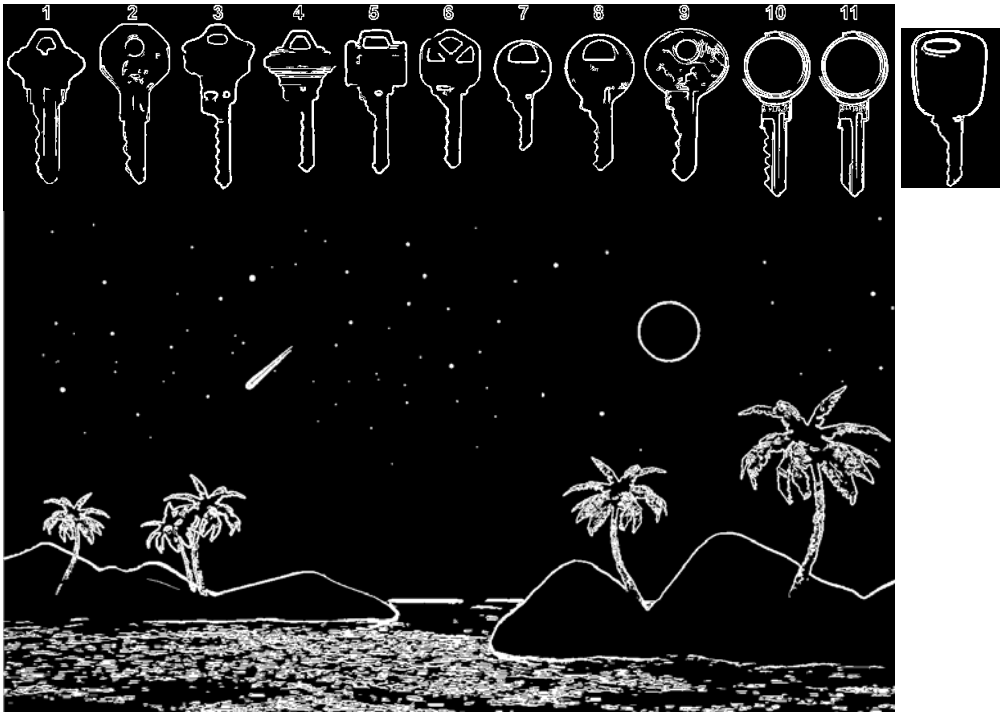




roberts算子

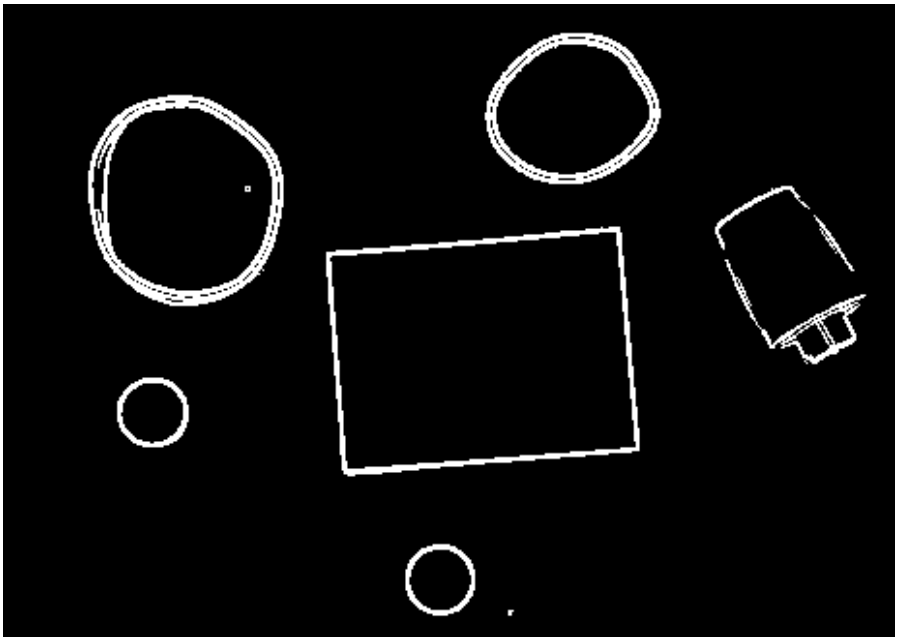
```
function roberts_value = roberts(v1,v2,v3,v4,v5,v6,v7,v8,v9)
    SX= abs(v9-v5);
    SY= abs(v6-v8);
    roberts_value = SX + SY;
```

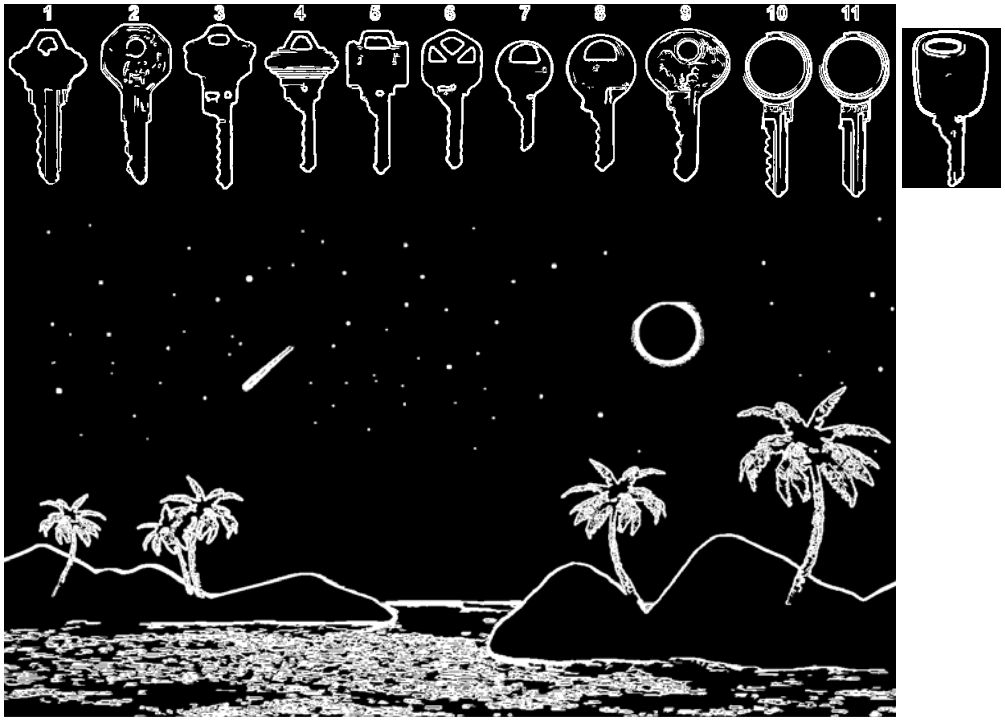




prewitt算子

```
function prewitt_value = prewitt(v1,v2,v3,v4,v5,v6,v7,v8,v9)
    SX= -v1+v3-v4+v6-v7+v9;
    SY= v1+v2+v3-v7-v8-v9;
    prewitt_value = sqrt(SX^2 + SY^2);
```



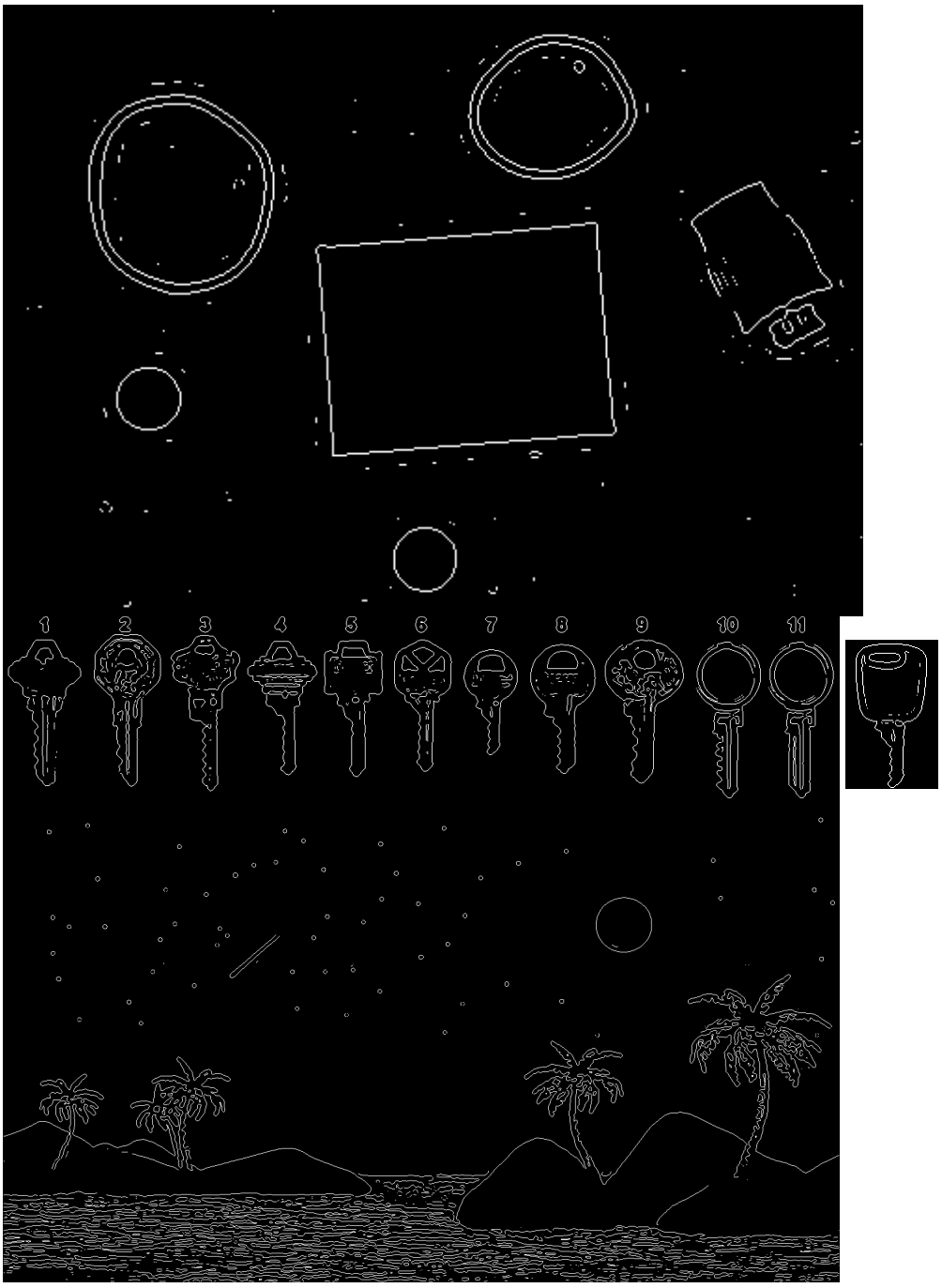


Marr算子

先对图像进行复制外边界的值来扩展的卷积，然后通过零交叉点判断边缘，最后用线性插值的方法估计边缘位置。

```
function output = my_edge_marr(input_image)
% input:gray image output:image edge logical
% marr
[m,n] = size(input_image);
e = false(m,n);
sigma = 2.5;
row = 2:m-1;
col=2:n-1;
op = fspecial('log',ceil(sigma*3) * 2 + 1,sigma);

op = op - sum(op(:))/numel(op); % make the op to sum to zero
b = imfilter(input_image,op,'replicate');

thresh = 0.75 * sum(abs(b(:)),'double') / numel(b);

% Look for the zero crossings
[r,c] = find(( b(row,col) < 0 & b(row,col+1) > 0 & abs( b(row,col)-b(row,col+1) )
> thresh )...
    |(b(row,col-1) > 0 & b(row,col) < 0 & abs( b(row,col-1)-b(row,col) ) >
thresh)...
    |(b(row,col) < 0 & b(row+1,col) > 0 & abs( b(row,col)-b(row+1,col) ) >
thresh)...
    |(b(row-1,col) > 0 & b(row,col) < 0 & abs( b(row-1,col)-b(row,col) ) >
thresh));
e((r+1) + c*m) = 1;
```

```
[rz,cz] = find( b(row,col)==0 );
if ~isempty(rz)
    zero = (rz+1) + cz*m;   % Linear index for zero points
    zz = ((b(zero-1) < 0 & b(zero+1) > 0 & abs( b(zero-1)-b(zero+1) ) >
2*thresh)...
        |(b(zero-1) > 0 & b(zero+1) < 0 & abs( b(zero-1)-b(zero+1) ) >
2*thresh)...
        |(b(zero-m) < 0 & b(zero+m) > 0 & abs( b(zero-m)-b(zero+m) ) >
2*thresh)...
        |(b(zero-m) > 0 & b(zero+m) < 0 & abs( b(zero-m)-b(zero+m) ) > 2*thresh));

    e(zero(zz)) = 1;
end
output = e;
end
```



四者对比

`Roberts`算子定位比较精确，但由于不包括平滑，所以对于噪声比较敏感。`Prewitt`算子和`Sobel`算子都是一阶的微分算子，而前者是平均滤波，后者是加权平均滤波且检测的图像边缘可能大于2个像素。这两者对灰度渐变低噪声的图像有较好的检测效果，但是对于混合多复杂噪声的图像，处理效果就不理想了。`Prewitt`和`Sobel`算子比`Roberts`效果要好一些，因为前两者参考了周围8个像素的灰度值，得到的信息更广。

`Marr`算子相比与上面三种非常突出，因为他考虑了梯度等因素，但是相比之下他被噪声影响的更为明显。

边缘链接

由于边缘链接的传入参数是一个二值图像，我应用一个队列将初始点入列，然后应用四邻域与八邻域两种情况分别对其作边缘链接，并返回所有连通分支上的点。
定义一个neighbour用来运算pix，即他的八/四邻域所有点坐标。

```
function output = my_edgelinking(binary_image, row, col)
    [M,N]=size(binary_image);

    tmp=[];
    queue_head=1;
    queue_tail=1;
    neighbour=[-1 -1;-1 0;-1 1;0 -1;0 1;1 -1;1 0;1 1];  %8-neighbourhood
%     neighbour=[-1 0;1 0;0 1;0 -1];     %4-neighbourhood
    q{queue_tail}=[row col];
    queue_tail=queue_tail+1;
    [ser1 , ~]=size(neighbour);
    num = 1;
    while queue_head~=queue_tail
        pix=q{queue_head};
        tmp(num,1)=pix(1,1);
        tmp(num,2)=pix(1,2);
        num = num + 1;
        for i=1:ser1
            pix1=pix+neighbour(i,:);
            if pix1(1)>=1 && pix1(2)>=1 &&pix1(1)<=M && pix1(2)<=N
                if binary_image(pix1(1),pix1(2)) == true
                    binary_image(pix1(1),pix1(2)) = false;
                    q{queue_tail}=[pix1(1) pix1(2)];
                    queue_tail=queue_tail+1;

                end
            end
        end
        queue_head=queue_head+1;
    end
    output = tmp;
```