EEE 178 Homework 6

Curtis Muntz

image preprocessing...

```
clear all, clc, close all;
addpath ../commonFunctions;
%code for my custom functions can be found on
%https://github.com/curtismuntz/machine_vision/tree/master/commonFunctions
%Problem 1 photo
P1original = getIMG('Tux2.png');
%Problem 2 photo
P2original = getIMG('mvHW62A.jpg');
%Problem 3 photo
P3original = getIMG('mvHW61.jpg');
rmpath ../commonFunctions
P1 = imresize(P1original, [401 401]);
```

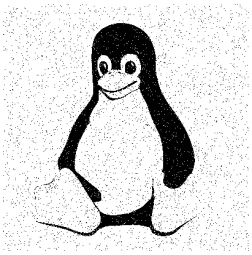
I. Problem 1: Majority Filtering

The majority filter produces pretty excellent results on black and white images with impulse noise. My median filter code was modified to this specific case to produce the majority filter function. For the most part, this function works very well until it hits the image borders. This is because its implementation ignores the borders of the image.

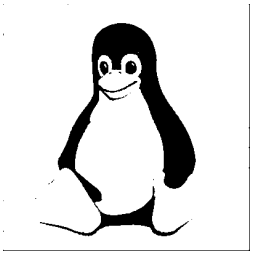
```
figure('name','Majority filtering')
Inoiz = imnoise(P1 ,'salt & pepper', 0.05);
Inoiz = im2bw(Inoiz);
subplot(121), imshow(Inoiz), title('Noisy Image');

addpath ../commonFunctions
Ifilt = majorityfilter(Inoiz);
rmpath ../commonFunctions

subplot(122), imshow(Ifilt), title('Majority Filtered Image');
```

```
function [MAJ] = majorityfilter(image)

    [M,N] = size(image);
    center=2;
    z=1;
    values=zeros(1,9);
    MAJ=zeros(size(image));
    MAJ=im2bw(MAJ);
    for i=1:M
      for j=1:N
        for k=1:3
          for l=1:3
            if (((i+(k-center)) > 0) && ((j+(l-center)) > 0) && ...
                                (i+(k-center) < M) && (j+(l-center) < N))
              values(z) = image((i+(k-center)),(j+(l-center)));
              z = z + 1;
            end
          end
        end
        if (sum(values) >= 5)
         MAJ(i,j) = 1;
        else
            MAJ(i,j) = 0;
      end
        clear values;
        z = 1;
    end
end
```

Noisy Image



Majority Filtered Image

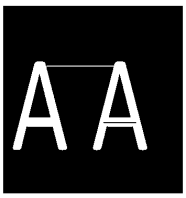


II. Problem 2: Removing Unwanted Portions of Objects

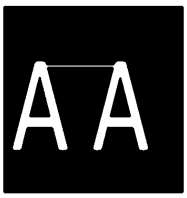
My overall strategy was to first do a closing operation to fill in holes and gaps, and remove some noise. This operation removed the dark line from the rightmost *A*. Next I opened the image, to remove objects. This operation removed the white line connecting the two *A*'s. Finally I dilated A to make the final image closer to the original image size. For simplicity, I used the same structuring element throughout each process.

```
figure('name','Removing Unwanted Portions of Objects');
A = im2bw(imresize(P2original, [401 401]));
B = strel('disk', 2);
closing = imclose(A,B);
reopening = imopen(closing,B);
final = imerode(reopening, B);
final = imdilate(final, B);
subplot(221), imshow(A), title('Original Image');
subplot(222), imshow(closing), title('Result After Closing Operation');
subplot(223), imshow(reopening), title('Result After Reopeing Closed');
subplot(224), imshow(final), title('Result After Opening dilate B')
```

Original Image

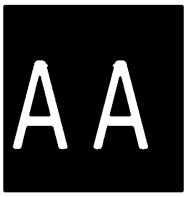


Result After Closing Operation



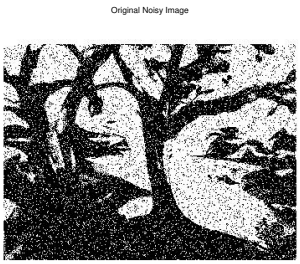
Result After Reopeing Closed

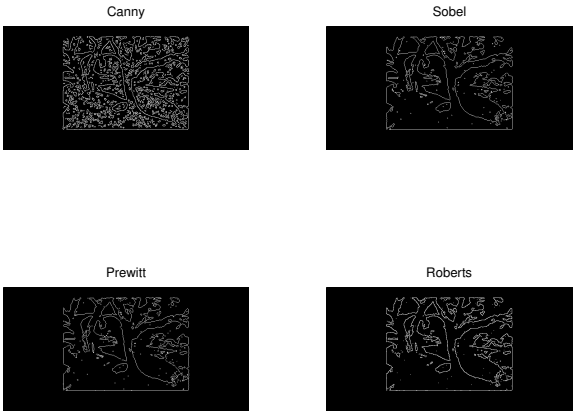
Result After Opening dilate B



III. Problem 3: Edge detection on a noisy image

```
P3 = rgb2gray(P3original);
figure('name', 'Edge Detection on a Noisy Binary Image')
imshow(P3), title('Original Noisy Image');
```

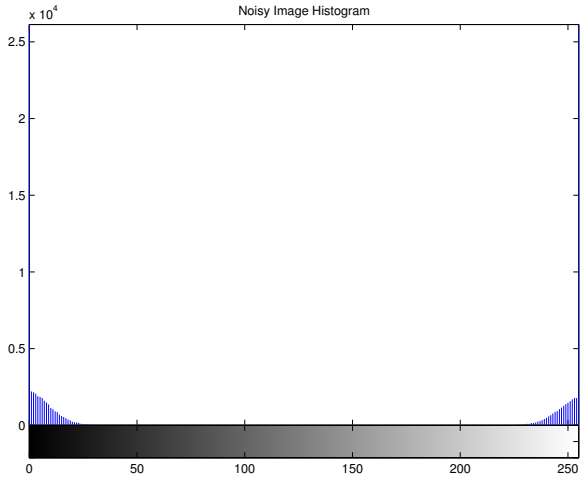




A. Part 1: Classical Edge Detectors

Because the image is so noisy, I wanted to filter the image before I attempted to do edge detection. Because I retrived the noisy image from the internet, I am unsure of the nature of the noise. Although I could estimate the noise type, I decided to be scientific instead and inspected the histogram of the image.

```
figure('name', 'Noisy Image Histogram');
imhist(P3), title('Noisy Image Histogram');
```



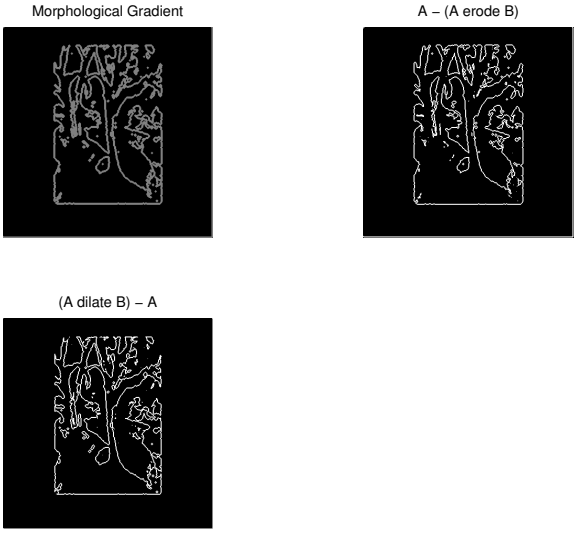
Because it seems that most of the histogram data is on the minimum and maximum gray values, I reasoned that the noise resembled impulse noise. Because of this, I first sent the image through my median filter. As shown in the following plots, the classical edge detection methods were fairly decent. It seems that the Roberts filter is the best of these classical edge detectors.

```
figure('name', 'Classical Edge Detection Methods')

addpath ../commonFunctions
P3f=medianfilter(P3);
rmpath ../commonFunctions

subplot(221), imshow(edge(P3f, 'canny')), title('Canny');
subplot(222), imshow(edge(P3f, 'sobel')), title('Sobel');
subplot(223), imshow(edge(P3f, 'prewitt')), title('Prewitt');
subplot(224), imshow(edge(P3f, 'roberts')), title('Roberts');
```

B. Part 2-4: Thresholding and using Morphological Edge Detectors

Again, the image is plagued with some serious noise. In order to try to get reasonable results from the edge detection methods, I first restored the image through the use of my majority filter. It can be assumed that the rgb impulse noise would translate into binary impulse noise, so I did not need to inspect the image histogram.

```
P3t = imresize(im2bw(P3, 0.4), [401 401]);

addpath ../commonFunctions
A = majorityfilter(P3t);
rmpath ../commonFunctions

B = strel('disk',2);
grad = (imdilate(A, B) - imerode(A, B))/2;

%part3
joe1=A-imerode(A, B);

%part4
joe2=imdilate(A, B)-A;

figure('name', 'Morphological Edge Detectors');
subplot(221), imshow(grad), title('Morphological Gradient');
subplot(222), imshow(joe1), title('A - (A erode B)');
subplot(223), imshow(joe2), title('(A dilate B) - A');
```



All of the morphological edge detectors were effective, but the gradient seemed to be most effective at detecting the edges.