Image Processing hw2 Report

Andrey Yershov

I. Problem 1

In the first problem, an image filtering was implemented using given mask with dimensions 3x3. The code was written in a way so user could choose the size of the mask he want to apply, between 3x3, 5x5 and 7x7. First I wanted to implement a function in a way so user could define the mask, however, my MATLAB version had a problem reading input in form of a matrix. So as long as the filter is averaging filter, I had to find the sum of all elements of the mask and divide each element of the mask by it, so sum of all elements is equal to 1. Next, zero padding was implemented. based on size of the mask and the fact that it is always square. So the size of padding is 1 for each side for N=3, 2 for N=5 and so forth. Next, going trough each pixel of the original image on the padded image, it is multiplied with the mask pixels and summed up to the point when there are no more mask pixels are left for the current position. Then the central pixel is shifted by one and multiplication continues. Finally, the image is shown with the original one, to see the difference between result and original.

A. Solution

The code is as follows:

```
%choose mask, cases for sizes 3, 5 and 7 are handled
prompt = 'Please type in scale:';
mask_scale = input(prompt);
if mask_scale == 3
        a = [1 2 3;4 5 6;7 8 10];
elseif mask_scale == 5
        a = [1 2 3 4 5;4 5 6 7 8;7 8 10 11 12;1 2 3 4 5; 6 7 8 9 10];
elseif mask_scale == 7
        a = [1 2 3 4 5 6 7;4 5 6 7 8 8 9;7 8 10 11 12 13 14;1 2 3 4 5 6 7; 6 7 8 9 10
            11 12; 1 2 3 4 5 6 7; 8 9 0 1 2 3 4];
end
%create averaging filter
a = a/sum(a, 'all');
[rows_m, cols_m] = size(a);
%find size of padding needed
padding = (mask_scale-1)/2;
%read image
image_or = im2double(imread('cameraman.tif'));
[rows_or, cols_or] = size(image_or);
image_padded = zeros(rows_or+(padding*2), cols_or+(padding*2));
image_filtered = zeros(rows_or, cols_or);
%implement 0 padding
for x = padding+1:rows_or+padding
    for y = padding+1:cols_or+padding
    image_padded(x,y) = image_or(x-padding, y-padding);
    end
end
%convolve padded image with mask
for x = 1 : rows_or
    for y = 1 : cols_or
        for m = 1 : rows_m
            for n = 1 : cols_m
                image_filtered(x, y) = image_filtered(x, y) + (image_padded(m+x-1, n+
                    y-1) * a(m, n));
            end
        end
```

```
35        end
36    end
37
38    %compare input and output
39    imshow(image_or)
40    figure
41    imshow(image_filtered)
```

B. Results

The results for this solution are as follows:

Fig. 1. Original Image

Fig. 2. Filtered with 3x3 mask

Fig. 3. Filtered with 7x7 mask

Clear difference is seen for the filtered images. Fuzzines appeared, as well as vignetting on the borders is clearly seen on the image filtered by 7x7 mask. This is due to bigger size of the filter, which takes more values from black 0 padding.

II. Problem 2

In the second problem the first solution was used with a Sobel edge detector mask. The magnitude of derivative of each pixel was found and the magnitude and direction of edges was found ultimately. There were no particular problems during solution of the first two problems. There were some array exceeding errors during first tries on convolution of the image with the mask, however it was handled with some tuning. The code is as follows:

A. Solution

```
function [magnitude, direction] = sobelf(image, threshold)
mask_scale=3;
a = [1 0 -1;2 0 -2;1 0 -1];
b = [1 2 1;0 0 0;-1 -2 -1];
[rows_m, cols_m] = size(a);
%find size of padding needed
padding = (mask_scale-1)/2;
%read image, created all necessary empty matrices to express results
image_or = im2double(image);
[rows_or, cols_or, ~] = size(image_or);
image_padded = zeros(rows_or+(padding*2), cols_or+(padding*2));
image_filtered = zeros(rows_or, cols_or);
image_filtered_x = zeros(rows_or, cols_or);
image_filtered_y = zeros(rows_or, cols_or);
direction = zeros(rows_or, cols_or);
%create padding
for x = padding+1:rows_or+padding
    for y = padding+1:cols_or+padding
    image_padded(x,y) = image_or(x-padding, y-padding);
    end
end
%convolve padded image with mask
for x = 1 : rows_or
    for y = 1 : cols_or
        for m = 1 : rows_m
            for n = 1 : cols_m
```

```
                    image_filtered_x(x, y) = image_filtered_x(x, y) + (image_padded(m+x
                        -1, n+y-1) * a(m, n));
                    image_filtered_y(x, y) = image_filtered_y(x, y) + (image_padded(m+x
                        -1, n+y-1) * b(m, n));
                end
            end
        %calculate magnitude
        image_filtered(x,y)=sqrt((image_filtered_x(x, y))^2+ (image_filtered_y(x, y))^2);
        %calculate direction
        direction(x,y) = atan2(image_filtered_y(x,y),  image_filtered_x(x,y));
        end
end
%apply thresholding
magnitude = image_filtered-(image_filtered*(threshold*0.01));
imshow(magnitude)
title('Gradient magnitude');
figure
imshow(direction)
title('Gradient direction');
```

So the code was first written as a script and then rearranged as a function for solution of the 4th problem. Also, direction of the edges was imlemented for solution of 4th task and is included in this solution. So as long as given above code is a function, it is tested from separate file.

```
image_or = im2double(imread('cameraman.tif'));
[magnitude, direction]=sobelf(image_or, 80);
```

B. Results

The result is as follows:

Fig. 4. Sobel without thresholding

Fig. 5. Sobel with 20% thresholding

Fig. 6. Sobel with 80% thresholding

So the Sobel edge detector seems to be working fine, especially when proper thresholding is applied.

III. Problem 3

For this problem again solution from the first problem was used with a median filter applied. The problem was solved relatively easy, however, when video was filtering, it took more than 1 hour.

A. Solution

```
function [denoised_image] = median_noise(noisy_image)
mask_scale=3;
a = [1 1 1;1 1 1;1 1 1];
%mask with size 7 was also used as an option
%a = [1 1 1 1 1 1 1;1 1 1 1 1 1 1;1 1 1 1 1 1 1;1 1 1 1 1 1 1;1 1 1 1 1 1 1;1 1 1 1 1 1 1;1 1 1 1 1 1 1;];
[rows_m, cols_m] = size(a);
padding = (mask_scale-1)/2;%find size of padding needed
image_or = noisy_image; %read image, create empty matrices
[rows_or, cols_or, ~] = size(image_or);
image_padded = zeros(rows_or+(padding*2), cols_or+(padding*2));
image_filtered = zeros(rows_or, cols_or);
for x = padding+1:rows_or+padding %0 padding
    for y = padding+1:cols_or+padding
    image_padded(x,y) = image_or(x-padding, y-padding);
    end
end
%convolve padded image with mask
for x = 1 : rows_or
    for y = 1 : cols_or
        %create new empty array for the values for current filter position
        array_median = [];
        for m = 1 : rows_m
            for n = 1 : cols_m
                u = image_padded(m+x-1, n+y-1) * a(m, n);
                %concatenate current result with the array
                array_median = [array_median u];
            end
        end
        %find median of the array for current position and write it to resultant image
        image_filtered(x, y) = median(array_median);
    end
end
denoised_image=uint8(image_filtered);
end
```

So this function was first tested on image, the result is as follows:

Fig. 7. Original image with salt and pepper noise

Fig. 8. Median denoiser with N=3 applied

As it could be seen above, the filter works pretty well for the salt and pepper noise. Next. code was written for opening video, extracting frames, filtering them and creating video back. Even though grayscale video frames were used, the processing time was extremely long for the 7x7 mask.

```
noisy = VideoReader('noise.mp4');
denoised = VideoWriter('denoised');
open(denoised);
        while hasFrame(noisy)
            vidFrame = readFrame(noisy);
            grayFrame = rgb2gray(vidFrame);
            denoisedFrame = median_noise(grayFrame);
            writeVideo(denoised,denoisedFrame);
        end
close(denoised);
```

So the results are as follows, all the resultant videos are submitted with the zip file.

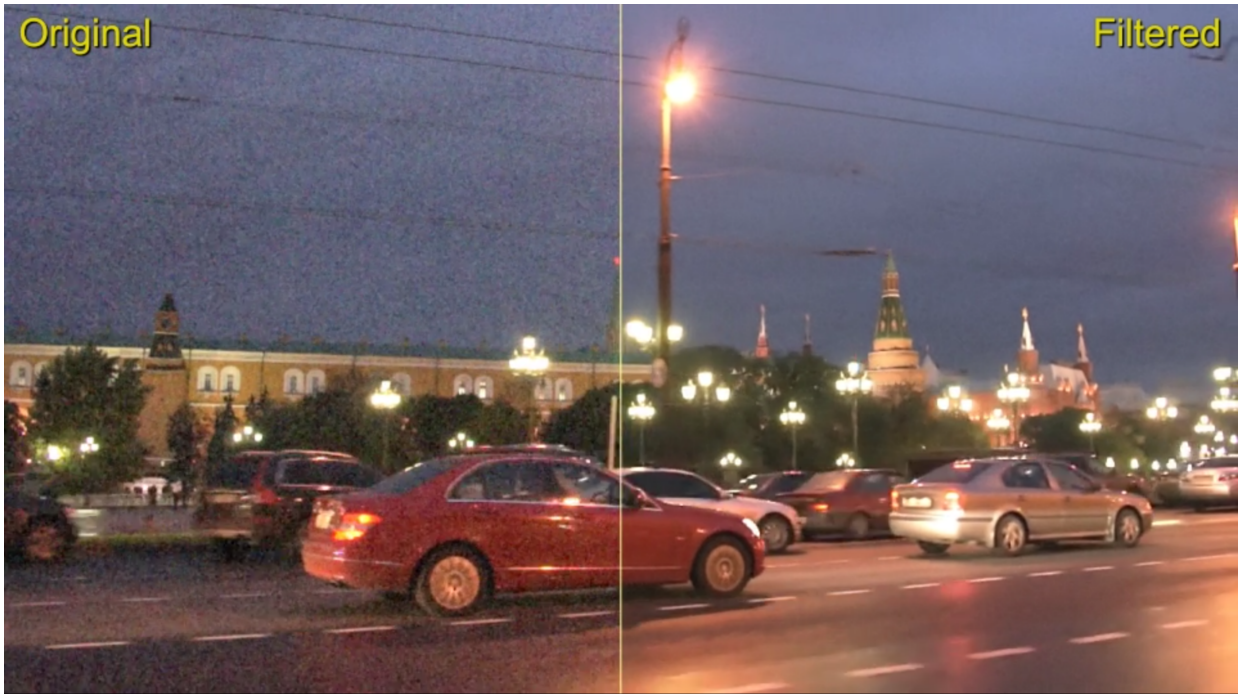

Fig. 9. Original frame

Fig. 10. Frame filtered with N=3 median filter

Fig. 11. Frame filtered with N=7 median filter

As it could be seen from results, N=3 mask has not created sufficiently smooth result. So N=7 mask was applied, which reduced noise considerably. However, the image also became fuzzy because of the large filter. Probably for this size of noise N=5 mask would have worked the best.

IV. Problem 4

This was the hardest problem to solve out of the 4. I modified the function from problem 2 so that it returns the matrices of size of original image with Sobel magnitudes and edge directions. Then given formula of Hough transfrom was applied,

calculating votes for the given image. Changing the value for calculating votes did not alter the results, so it was left as in suggested modified Hough transform.

A. Solution

So the code for the solution is as follows:

```
image_or = im2double(imread('cameraman.tif'));
%call for sobel function
[magnitude, direction]=sobelf(image_or, 0);
[rows_or,cols_or, z]=size(image_or);
%max possible distance from the origin in hough space
distance = round(sqrt(rows_or^2 + cols_or^2));
%Initialize H so that no boundaries will be exceeded
H = zeros(362, distance*2); %181 element from -pi to pi
for j = 1 : rows_or
    for k = 1 : cols_or
        if ~isnan(direction(j,k)) %check if there is valid value in corresponding
            cell
            theta = direction(j,k)*(180/pi)+180; %moved +pi towards infinity to not
                to exceed boundaries
            rho = j*cosd(theta)+k*sind(theta);
            rho = abs(round(rho)); %take absolute rounded value to use as array
                coordinate
            if rho==0 %deal with exception for some images
                rho=rho+1;
            end
            H(round(theta),rho) = H(round(theta),rho)+1;
        end
    end
end
c=[0 10];
imagesc(H,c) %use imagesc() to use differentiate votes by colours
```

B. Results

For the test, three different images were used. Two intersecting lines to show that the code is working (because we know what result to expect - two dots), then standard cameraman.tif and a photo of house with vertical lines present. So the result of this solution is as follows:

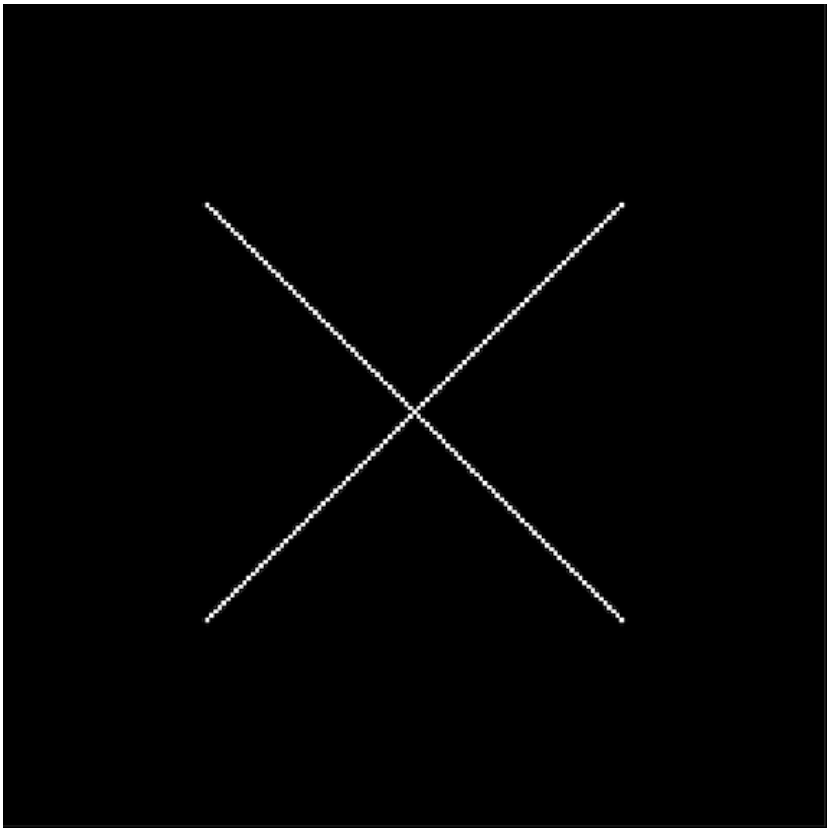

Fig. 12. Original image

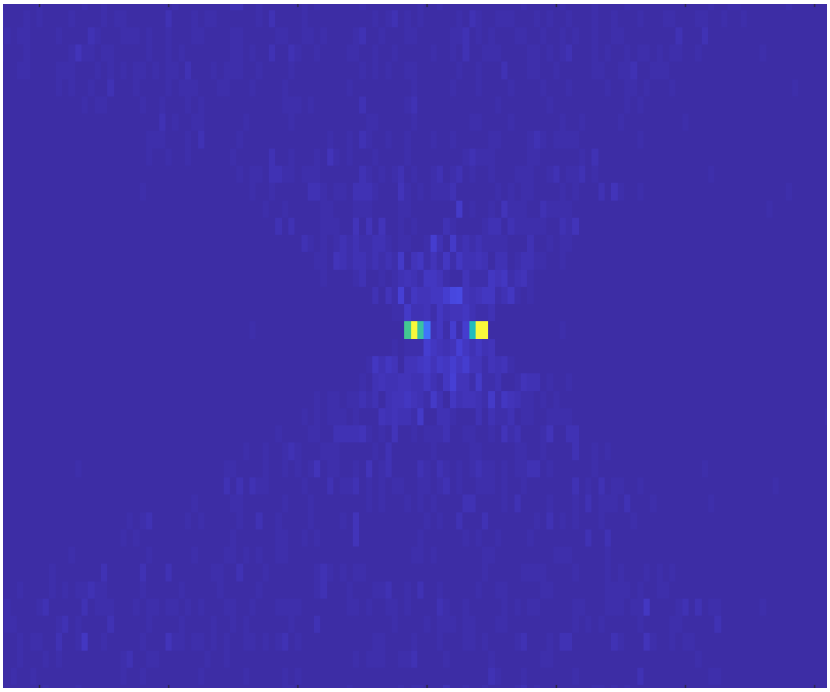

Fig. 13. Hough transform

Fig. 14. Original image

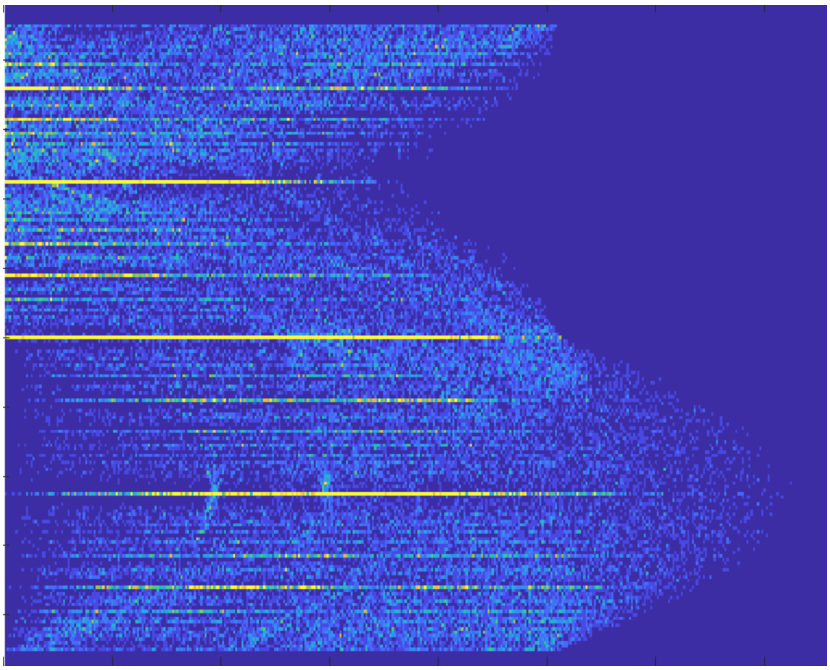

Fig. 15. Hough transform

Fig. 16. Original image

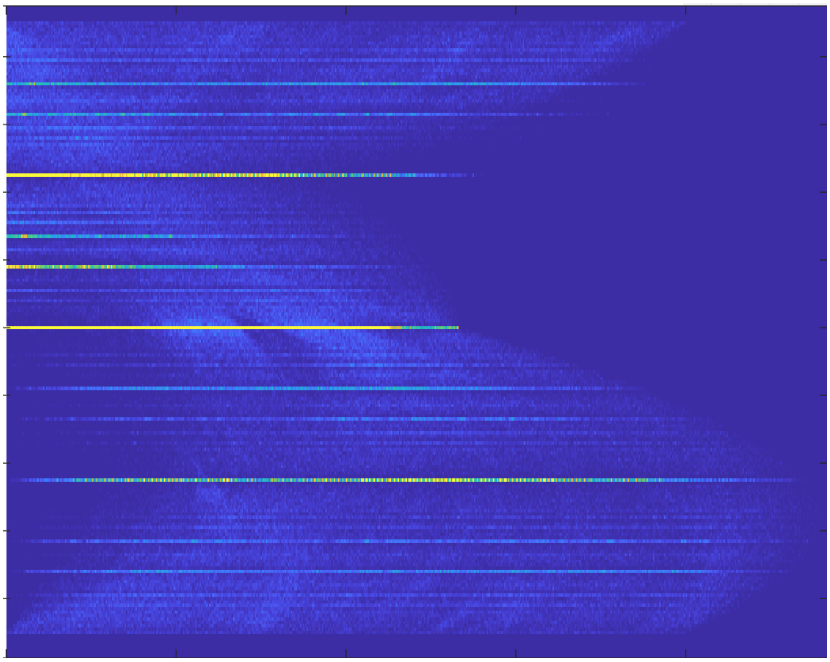

Fig. 17. Hough transform

As it could be seen from results, the algorithm is working, however not perfectly in every case. There are some artifacts in form of lines in more complicated images, which are probably due to rounding the results in the code and taking absolute values. For the first example, the result is as expected - two points, however with some fuzziness due to the line being not 1 pixel wide. On other images points of vote concentration are clearly seen.