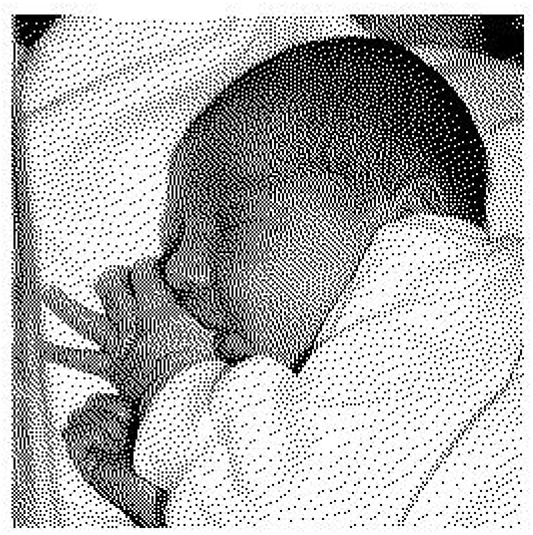
Name: Amirhossein Moghaddas Jafari
Degree: ME
ID: 20804524

Project Report

- Data type and Conversions
- Difference between effective resolution and real resolution
- Quantization and Dithering

  Most images have 256 grayscales, in some cases, like in newspapers, we can show only 2 grayscales level. Using the theorem of **error diffusion** in order to make `fl_stein` function. The image just has 2 grayscales and it is completely usable in newspaper.



- `imsubtract` , `imadd` and `imcomplement`functions to add, substract or complement a number on image pixels. Note that, the image should be the type of double to use Arithmetic Operations.
- Histograms

  we always tend to have a gray scale image which the distribution of the number of times each gray level occurs be wide and equalized.

  - Histogram Stretching: Widen the grayscale distribution without changing the number of occurrence. Function in MATLAB is `imadjust(t, [a,b], [c,d], gamma)`
  - Histogram Equalization: Widen the grayscale distribution, and, meantime, equalized the number of occurrence to have an ideal uniform occurrence. The function is `histpwl`
  - Lookup Tables (LUT): We can do Histogram Stretching with this manner. In my view, since, we can define unlimited section, and arbitrary function to be applied on grayscales, it is better idea than Histogram Equalization.
- Filtering

  Using `Filter2` or `conv2` function in MATLAB.

  - Low and High-pass filter: for make an image blur or show the edge.

    - Gaussian Filters
    - Unsharp Masking : using `average` in `fspecial` and then be applied on image by `filter2`
    - High-Boost Filtering : subtraction of original image and a low pass filter.
    - Nonlinear Filters
- ROI

  It is a binary image the same size as the original image, with 1s for the ROI and 0s elsewhere.
- Region of Interest Filtering

  `roifilt2`function in MATLAB
- Interpolation of Data

  Difference between `nearest`, `bilinear` and `bicubic` method for image resize.
- Rotating
- Anamorphosis
- The Two-Dimensional DFT

  Using `fft2` and `ifft2` function in MTALAB
- Fourier Transforms of Images

  using `fftshow` function to show the transformed image. Note that it is necessary to use `fftshift` function along `fft2` to make the image showable.

  edges in image would be lines in transformed image
- Filtering in the Frequency Domain

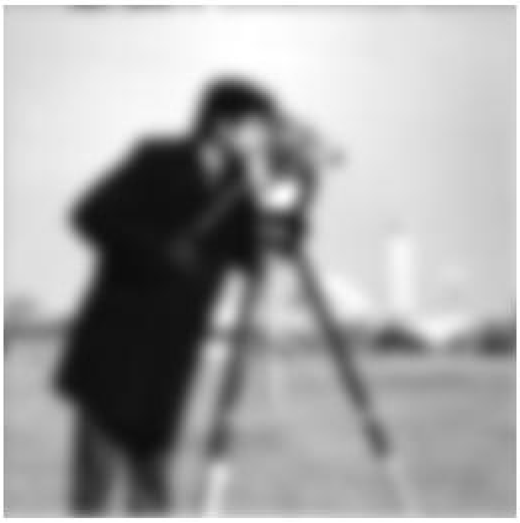
  By using convolution theorem, we can apply mask on images, without convolution, we can easily take `fft2` of the image, and multiplying it into a filter (note that filter should not necessary in frequency domain), then take `ifft2` from the result and show the image by `fftshow` . It is much faster than convolution.

Proof of Convolution theorem with MATLAB functions

```
cm = imread('cameraman.tif');
cf =fftshift(fft2(cm));
figure
fftshow(cf)

g1 = fspecial('gaussian',256,10);
g2 =abs(fftshift(ifft2( g1 ))); %it is the inverse fft of the filter g1, we use
abs to use the value of the elements, we use fftshift
%function to sort it out in the middle


cg1 = cf .* g1;
figure
fftshow(cg1,'log'); %looks like a low-pass filter
fftshow(ifft2(cg1),'abs') %showing the picture obtained from invers fft

%using conv2
```

```
cg2 = conv2(cm,g2,'same'); %it is approximatley the same with  cg2 =
conv2(cm,g1,'same') because fft of a gaussian is another gaussian
%that is why we are interested to GAUSSIAN filters
figure
imshow(cg2/max(cg2(:)))%  the same with above, %we divided it to the max of cg2
to show it precisly
% or fftshow(cg2,'abs')

%note that it is the same with using filter2 function, infact, we use
convolution
%theoriom to make this process much faster, intead of doing a 2D
%convolution which takes a lot time to be processed, we multiply fft of
%image in the filter(gaussian filter here), then, using inverse fft to
%obtain the image
```

Result by filtering in frequency domain (`fft2` and `ifft2`)



Result by filtering with `conv2` or `filter2`

They are the same, the difference is because of zero padding which is automatically applied on `conv2`

- High-Pass Gaussian filters

  ```
  gh = 1-g1
  ```

- Homomorphic Filtering

  Sometimes, We need to manipulate an image in two way. For example manipulating the illumination and reflectance when an image suffer from darkness in some sections and light in others. it is done by `homfilt` function.

- Noise

  - Salt and Pepper Noise: `imnoise(im, 'salt & pepper')`
  - Gaussian Noise : `imnoise(im, 'gaussian')`
  - Speckle Noise: `imnoise(im, 'speckle')`
  - Periodic Noise, it can be made like this

    ```
    s = size(t);
    [x,y] = meshgrid(1:s(1),1:s(2));
    p = sin(x/3 + y/3)+1;
    t_pn = (im2double(t)+p/2)/2;
    imshow(t_pn)
    ```

- Cleaning Salt and Pepper Noise

  The best way is applying median filter. The command is `medfilt2(im,[3 3])` .

- Cleaning Gaussian Noise

  The best way is applying adaptive filtering by `wiener2` command

```
t2 = imnoise(t,'gaussian',0,0.005);
figure
imshow(t2)
t2w = wiener2(t2,[7,7]);
figure
imshow(t2w)
```

- Removal of Periodic Noise

  When we have Periodic Noise, there will be some other spikes regarding to the Periodic Noise which is sine or cosine noise. We aim to remove that Spikes in order to have a noise less image.

  - Notch Filtering : we simply make the rows and columns of the spikes 0
  - Band Reject Filtering : we create a circle filter and then applied to remove the spikes
- Inverse Filtering

  Recovering the original image by dividing into the filter

  - Method one

    ```
    wbf = fftshift(fft2(wba));
    w1 = (wbf ./ b) .* lbutter(w,40,10);
    wla = abs(ifft2(w1));
    imshow(wla/max(wla(:))); %it is not good
    ```

    It is not applicable, go for second method

  - Method two (applicable method)

    ```
    d = 0.01;
    b = lbutter(w,15,2); b(find(b<d)) = 1;
    w1 = fftshift(fft2(wba)) ./ b;
    wla = abs(ifft2(w1));
    imshow(mat2gray(wla));
    ```

- Motion Deblurring

  We would like this to remove blur noises

```
%board image
bc = imread('board.tif');
bg = im2uint8(bc);
b = bg(100:355,50:305);
imshow(b)

%applying motion filter
m = fspecial('motion',7,0);
bm = imfilter(b,m);
imshow(bm) %image with motion noise

%fft of the motion filter
m2 = zeros(256,256);
m2(1,1:7 ) = m;
mf = fftshift(fft2(m2));
fftshow(mf)
```

```
%obtaining the original image (bad method)

bmi = ifft2(fftshift(fft2(bm)) ./ mf );
fftshow(bmi,'abs') %bad quality

%obtaining original image with setting a threshold for fft of the filter
(good method)

d = 0.02
mf = fftshift(fft2(m2)); mf(find(abs(mf)<d)) = 1;
bmi = ifft2(fftshift(fft2(bm)) ./ mf);
imshow(mat2gray(abs(bmi))*2) %good quality image
figure
fftshow(bmi,'abs') %the same with imshow(mat2gray(abs(bmi))*2)
```

- Wiener Filtering

```
%using least square method

w = imread('wombats.bmp');
figure
imshow(w)
b = lbutter(w,15,2); %making bluring filter with butter function
k = 0.01;
wb = wf.* b; %applying filter
wba = ifft2(wb); %getting inverse to show the image
figure
fftshow(wba,'abs')

wbf = fftshift(fft2(wba));
w1 = wbf .* (abs(b) .^ 2 ./ (abs(b) .^ 2 + k)./b);
wla = abs(ifft2(w1));
figure
imshow(mat2gray(wla)) %should be app. simillar to the original image
```

- Change an image into the binary image with setting the best threshold

  First step: Finding the best threshold, this method is called Otsu method.

```
b = imread('bacteria.tif');
tn = graythresh(b); %it is the best threshold for using in im2bw function
```

  Second step: Make binary image with `im2bw` function

```
imshow(im2bw(b,tn)) %tn is the threshold obtained from First step
```

- Adaptive thresholding

  In some cases, both background and object is vary in color in the image, if we use normal process to make a binary image for them, the result would not be satisfying. Then, we need to do different thresholding for each part of image. [256,64] is the

```
fun = inline('im2bw(x,graythresh(x))');
t = blkproc(b,[256,64],fun);
```

- Edge Detection
  - Prewitt edge detection:`edge(ic,'prewitt')` using first derivative
  - Roberts edge detection: `edge(ic,'roberts')`
  - Sobel edge detection: `edge(ic,'sobel')` the smoothing is different with Prewitt method
  - zero crossing: `edge(ic,'zerocross',1)` using second derivatives.
  - Canny edge detection : `edge(ic,'canny')` First smoothing the image with gaussian filter, then, using Laplacian to find the edge in both direction.
- The Hough Transform

  Here is the code to draw the line in images

```
c = imread('cameraman.tif');
cw = edge(c,'canny');
%imshow(cw)

[h,t,r] = hough(cw);

p = houghpeaks(h,6,'threshold',10); %6 ia the number of the peacks
d = r(p(:,1));
theta = t(p(:,2));

x = 1:size(c,2); %setting x direction cordinate

y1 = 1 / sind(theta(1)) * (d(1)-x*cosd(theta(1)));
y2 = 1 / sind(theta(2)) * (d(2)-x*cosd(theta(2)));
y3 = 1 / sind(theta(3)) * (d(3)-x*cosd(theta(3)));

subplot(151); imshow(c); hold on; plot(x, y1,'r', 'linewidth', 3); hold off
subplot(152); imshow(c); hold on; plot(x, y2,'g', 'linewidth', 3); hold off
subplot(153); imshow(c); hold on; plot(x, y3,'b', 'linewidth', 3); hold off
subplot(154); imshow(c); hold on; plot(x, y4,'c', 'linewidth', 3); hold off
```

- Dilation and erosion

```
t = imread('text.tif');
sq =ones(3,3);
td = imdilate(t,sq);
figure
imshow(td) %image will be thicker

%%errosion
ic = imread('circbw.tif');
ce = imerode(ic,sq);
figure
imshow(ce); %image will be thiner
```

- Boundary Detection

  It can be done by the following code on binary images

```
sq =ones(3,3);
rice = imread('rice.tif');
r = rice>110;
re = imerode(r,sq);
 r_int = r &~ re;

figure
imshow(r)
figure
imshow(r_int) %internal boundry

rd = imdilate(r,sq);
r_ext = rd &~ r;
r_grad = rd&~re;
imshow(r_ext); %external boundry
figure
imshow(r_grad) %gradient boundry
```

- Opening and Closing
    - using for noise removal with following commands

    ```
    c = imread('circles.png');
    x = rand(size(c));
    d1 = find(x<0.05);
    d2 = find(x>0.95);
    c(d1) = 0;
    c(d2) = 1;
    figure,imshow(c),title('original');

    se1 = ones(3,3);
    cf1 = imclose(imopen(c,se1),se1);
    figure,imshow(cf1),title('filtered with ones(3,3) se');

    se2 = [0 1 0 ; 1 1 1 ; 0 1 0];
    cf2 = imclose(imopen(c,se2),se2);
    figure,imshow(cf2),title('filtered with cross se');
    ```

- Region filling

`regfill` function would be used for filling a region bounded. Following code can be used.

```
x = imread('nicework.tiff');
figure,imshow(x),title('original image');

se = ones(3,3);
xb = x&~imerode(x,se);
figure,imshow(xb),title('boundary');

xf = regfill(xb,[55,97],se); %74,52      57,97
figure,imshow(xf|xb);
```

- counting the number of white regions (1s region in binary images)

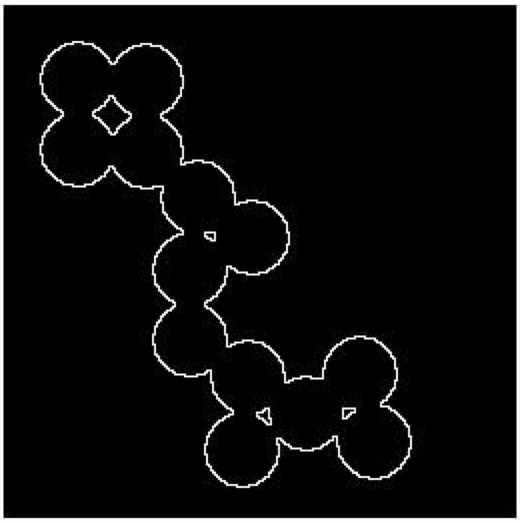
  using `bwlabel` command to determine the number of white region

```
b = imread('bacteria.tif');
figure
imshow(b)
bt = b<100;
figure
imshow(bt)
b1 = bwlabel(bt);
max(b1(:))
```

- Lookup tables (boundary founding)

  It is another method for boundary detection.

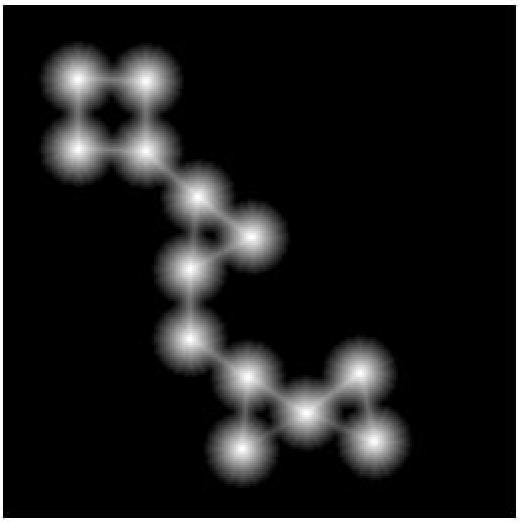
```
f = inline('x(5)&~(x(2)*x(4)*x(6)*x(8))');
lut = makelut(f,3);
c= imread('circles.png');
cw = applylut(c,lut);
imshow(c)
figure
imshow(cw)

%other method
sq =ones(3,3);
re = imerode(c,sq);
c_int = c &~ re;

figure
imshow(c)
figure
imshow(c_int)
```



- Distances and Metric

  Using MATLAB function in order to estimate the distance of zero elements in an image to the nearest non-zero elements. `bwdist` can be used for this aim.

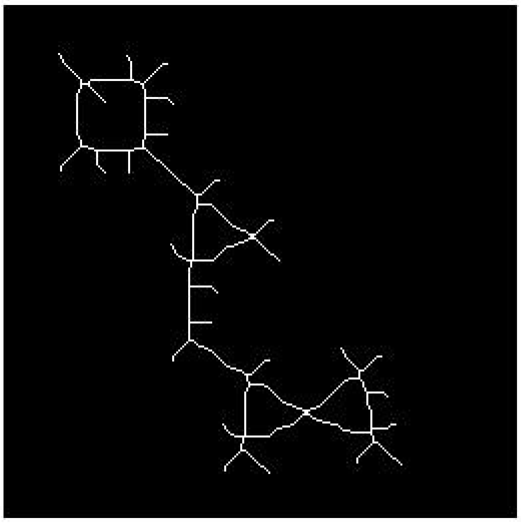
```
c= ~imread('circles.png'); %add ~ to make it more showable
d = bwdist(c);
%D = bwdist(BW) computes the Euclidean distance transform
%of the binary image BW. For each pixel in BW, the distance
%transform assigns a number that is the distance between that
%pixel and the nearest nonzero pixel of BW
imshow(d/max(d(:)))
```



- Skeletonization

  using `bwmorph` function in order to skeletonization.

```
c= imread('circles.png')
BW3 = bwmorph(c,'skel',Inf);
figure
imshow(BW3)
```



There are also some other algorithms for performing skeletonization for an image, but, in my view, MATLAB function do it better. MATLAB uses *Guo-Hall* algorithm.