Thresholding:

```
clc;
clear all;
close all;
img = imread('cameraman.tif');
% img = rgb2gray(img);
[m, n] = size(img);
avg = mean(img(:));
% size(img(:)) is [m*n, 1]
% median(img(:))
% mode(img(:))
nwimg = ones(m, n);
for i = 1: m
for j= 1: n
if img(i,j) >= avg
nwimg(i,j) = 255;
else
nwimg(i,j) = 0;
end
end
end
imshow(nwimg);
whos;
fprintf("%d\n", 2);
enter = input("Press Enter to exit\n");
```

Swap RGB:

```
img = imread('rgb.jpg');
[m, n] = size(img);
size(img)
r = img(:,:,1);
g = img(:,:,2);
b = img(:,:,3);
% Swap RGB channels
nwimg(:,:,1) = b;
nwimg(:,:,2) = r;
nwimg(:,:,3) = g;
% Plot
figure(1);
subplot(2,1,1)
imshow(img);
subplot(2,1,2)
imshow(nwimg);
figure(2);
imshow(nwimg);
figure(3);
histogram(b(:), 10); % 10 bins
% Alternative to imshow(img) is imtool(img);
imtool(nwimg);
if (1>0)
disp([1, 2, 3, 4]);
end
```

Gamma Transform:

```
c = input("Type the value of c ");
gamma = input("Type the value of gamma ");
img = imread('spinal_cord_1.png');
img = double(img);
[m,n]=size(img);
nwimg = zeros(m, n);
for i = 1: m
for j = 1: n
nwimg(i,j)= c*(img(i,j)^gamma);
end
end

figure(1);
% Only for visual purpose, o.w. scale values using Normalisation.
imshow(nwimg,[]);

% Approximate method of scaling.
imwrite(uint8(nwimg),'spinal_cord_1_v1.bmp')

% Scaling reqd. before storing the image using imwrite().
minima = min(nwimg(:));
maxima = max(nwimg(:));
diff = maxima-minima;
nwimg = 255*(nwimg-minima)/diff;
nwimg = uint8(nwimg);
figure(2);
imshow(nwimg);
imwrite(nwimg,'spinal_cord_1_v2.bmp')
```

Bit Plane Slicing:

```
a=imread('rgb.jpg');
a=rgb2gray(a);
imshow(a);
[x, y]=size(a);
a=double(a);
for k=7:-1:0
for i=1:1:x
for j=1:1:y
if bitand(2^k,a(i,j))==0
% We have bitor(234, 123)=251, bitcmp(uint8(0))=255,
bitxor(234, 123)=145.
b(i,j)=0;
else
b(i,j)=255;
end
end
end
b=uint8(b);
figure();
imshow(b);
end
```

Histogram Equalisation:

```
a=imread('164054396-beach-wallpapers.jpg');
a=rgb2gray(a);
figure;
imshow(a);
a=double(a);
[m, n]=size(a);
histo=zeros(1,256);
for i=1:1:m
for j=1:1:n
curpix=a(i,j);
histo(curpix+1)=histo(curpix+1)+1;
end
end
histo=histo/(m*n);
s=zeros(1,256);
s(1)=histo(1);
for i=2:1:256
s(i)=s(i-1)+histo(i);
end
%for i=1:1:256
%for j=1:1:i
%s(i)=s(i)+hist(j);
%end
%end
s=s*255;
b=zeros(m,n);
for i=1:1:m
for j=1:1:n
b(i,j)=s(a(i,j)+1);
end
end
b=uint8(b);
figure();
imshow(b);
```

Median Filtering To Remove Salt Pepper Noise:

```
a=imread('salt_pepper_noise_2.jpg');
a=rgb2gray(a);
imshow(a);
a=double(a);
b=a;
[x, y]=size(a);
for i=2:1:x-1
for j=2:1:y-1
img = a(i-1:i+1,j-1:j+1);
b(i,j)=median(img(:));
end
end
b=uint8(b);
```

Normal Averaging/ Blurring:

```
a=imread('blur.png');
a=rgb2gray(a);
imshow(a);
a=double(a);
[x, y] = size(a);
% b = zeros(x, y); Zero Padding Case.
b = a;
masksize = 15;
mask=ones(masksize, masksize);
for i=1+floor(masksize/2):1:x-floor(masksize/2)
for j=1+floor(masksize/2):1:y-floor(masksize/2)
for k=1:1:masksize
for l=1:1:masksize
b(i,j)=b(i,j)+mask(k,l)*a(i+k-floor(masksize/2)-1,j+l-floor(masksize/2)-1);
end
end
b(i,j)=b(i,j)/(masksize*masksize);
end
end
b=uint8(b);
figure();
imshow(b);
```

Median Filtering To Remove Salt Pepper Noise:

```
a=imread('salt_pepper_noise_2.jpg');
a=rgb2gray(a);
imshow(a);
a=double(a);
b=a;
[x, y]=size(a);
for i=2:1:x-1
for j=2:1:y-1
count=1;
list=zeros(1,9);
for k=1:1:3
for l=1:1:3
list(count)=a(i+k-2,j+l-2);
count=count+1;
end
end
list=sort(list);
b(i,j)=list(5);
end
end
b=uint8(b);
figure;
imshow(b);
```

Laplacian:

```
a=imread('laplace.png');
a=rgb2gray(a);
imshow(a);
a=double(a);
mask=[-1,-1,-1;-1,8,-1;-1,-1,-1];
[x, y]=size(a);
b=zeros(x,y);
for i=2:1:x-1
for j=2:1:y-1
for k=1:1:3
for l=1:1:3
b(i,j)=b(i,j)+mask(k,l)*a(i+k-2,j+l-2);
end
end
end
end
b=uint8(b);
figure;
imshow(b);
```

Weighted Averaging:

```
a=imread('blur.png');
a=rgb2gray(a);
imshow(a);
a = double(a);
[m, n] = size(a);
mask = (1/20)*[1, 2, 1; 2, 8, 2 ; 1, 2, 1];
% mask = (1/56)*[5, 7, 5; 7, 8, 7; 5, 7, 5];
msk=3 % dimension of square mask
% b = zeros(m, n);
b = a;
mm = (msk-1)/2;
for i=1+(msk-1)/2: m-(msk-1)/2
for j=1+(msk-1)/2: n-(msk-1)/2
% b(i, j) = sum(sum(mask.*a(i-mm:i+mm,
j-mm:j+mm)));
subimg = mask.*a(i-mm:i+mm,
j-mm:j+mm);
b(i, j) = sum(subimg(:));
end
end
subplot(2,1,1); imshow(uint8(a));
subplot(2,1,2); imshow(uint8(b), []);
```

Assignment 1 Thresholding:

```
img = imread('For_asign1.jpg');
figure();
imshow(img);
[m, n] = size(img);
disp(m)
disp(n)
nwimg = img;
for i = 1: m
for j = 1: n
if img(i, j) >= 75 && img(i, j) <= 150
nwimg(i, j) = 255;
end
end
end
disp(size(nwimg))
figure();
imshow(nwimg);
```

Dilation:

```
% Create Image of 65x65 square
img = zeros(256,256);
for i=64:128,
for j=64:128,
img(i, j) = 255;
end
end
figure();
imshow(img);
% Symmetric Square Strel
strelSize = 31;
center = [15, 15];
h = (strelSize-1)/2;
[m, n] = size(img);
% Below 2 lines implement padding
IMG = [zeros(m, h), img, zeros(m, h)];
IMG = [zeros(h, n+(2*h)); IMG; zeros(h, n+(2*h))];
result = zeros(m, n);
for i=1+h:m+h
for j=1+h:n+h
subimg = IMG(i-h:i+h, j-h:j+h);
% if (subimg(center(1),center(2)) == 255)
% result(i-h, j-h) = 255;
% continue;
% end
vals = subimg(:);
[p, q] = size(vals); % q will always be 1
for k= 1:p
if vals(k) == 255
result(i-h, j-h) = 255;
break;
end
end
end
end
figure();
imshow(result);
```

Erosion:

```
% Create Image of 65x65 square
img = zeros(256,256);
for i=64:128,
for j=64:128,
img(i, j) = 255;
end
end
figure();
imshow(img);
% Symmetric Square Strel
strelSize = 31;
center = [15, 15];
h = (strelSize-1)/2;
[m, n] = size(img);
% Below 2 lines implement padding
IMG = [zeros(m, h), img, zeros(m, h)];
IMG = [zeros(h, n+(2*h)); IMG; zeros(h, n+(2*h))];
result = ones(m, n);
for i=1+h:m+h
for j=1+h:n+h
subimg = IMG(i-h:i+h, j-h:j+h);
% if (subimg(center(1),center(2)) == 0)
% result(i-h, j-h) = 0;
% continue;
% end
vals = subimg(:);
[p, q] = size(vals); % q will always be 1
for k= 1:p
if vals(k) == 0
result(i-h, j-h) = 0;
break;
end
end
end
end
result = result*255;
figure();
imshow(uint8(result));
```

Hit-Miss Transformation:

```
% disp(func(2, 3));
% Create Image of 65x65 square
img = zeros(256,256);
for i=64:128
for j=64:128
img(i, j) = 255;
end
end
imshow(img);
[m, n] = size(img);
B1 = [2 255 2; 0 255 255; 0 0 2];
B2 = [2 255 2; 255 255 0; 2 0 0];
B3 = [2 0 0; 255 255 0; 2 255 2];
B4 = [0 0 2; 0 255 255; 2 255 2];
B = [B1 B2 B3 B4];
% Padding:
img = [zeros(m, 1), img, zeros(m, 1)];
img = [zeros(1, n+2); img; zeros(1, n+2)];
result = zeros(m, n);
for i=2:1:m+1
for j=2:1:n+1
subimg = img(i-1:i+1, j-1:j+1);
for k=1:3:10
mask = B(:,k:k+2);
flag = 0;
for x=1:1:3
for y=1:1:3
if mask(x, y) == 2
continue;
end
if mask(x, y) ~= subimg(x, y)
flag = 1;
break;
end
end
if flag == 1
break;
end
end
if flag == 0
result(i-1,j-1) = 255;
break;
end
end
end
end
imshow(result);
```

Boundary Extraction Using Erosion:

```
img=imread('boundary_extraction.png');
img=rgb2gray(img);
[m, n] = size(img);
% Thresholding Original Image to make it Binary Image:
avg = mean(img(:));
for i=1:m
for j=1:n
if img(i,j) >= avg
img(i,j) = 255;
else
img(i,j) = 0;
end
end
end
figure();
imshow(img);
% Symmetric Square Strel
strelSize = 3;
center = [1, 1];
h = (strelSize-1)/2;
% Below 2 lines implement padding
IMG = [zeros(m, h), img, zeros(m, h)];
IMG = [zeros(h, n+(2*h)); IMG; zeros(h, n+(2*h))];
result = ones(m, n);
for i=1+h:m+h
for j=1+h:n+h
subimg = IMG(i-h:i+h, j-h:j+h);
vals = subimg(:);
[p, q] = size(vals); % q will always be 1
for k= 1:p
if vals(k) == 0
result(i-h, j-h) = 0;
break;
end
end
end
end
result = result*255;
boundary_img = zeros(m, n);
for i=1:m
for j=1:n
boundary_img(i,j) = img(i,j)-result(i,j);
end
end
figure();
imshow(uint8(boundary_img));
```

End Point Detection:

```
% img = imresize(img, 4);
% Create Image
img = zeros(256, 256);
for i=30:64
img(i, 20) = 255;
end
for j=30:64
img(20, j) = 255;
end
for i=30:64
img(i, i+3) = 255;
end
for i=30:1:45
img(i, 33) = 255;
end
[m, n] = size(img);
figure();
imshow(img);
B1 = [255 255 2; 2 2 2; 2 2 2];
B2 = [2 255 255; 2 2 2; 2 2 2];
B3 = [2 2 255; 2 2 255; 2 2 2];
B4 = [2 2 2; 2 2 255; 2 2 255];
B5 = [2 2 2; 2 2 2; 2 255 255];
B6 = [2 2 2; 2 2 2; 255 255 2];
B7 = [2 2 2; 255 2 2; 255 2 2];
B8 = [255 2 2; 255 2 2; 2 2 2];
B = [B1 B2 B3 B4 B5 B6 B7 B8];
% Padding:
img = [zeros(m, 1), img, zeros(m, 1)];
img = [zeros(1, n+2); img; zeros(1, n+2)];
result = zeros(m, n);
for i=2:1:m+1
for j=2:1:n+1
if img(i,j) ~= 255
continue;
end
subimg = img(i-1:i+1, j-1:j+1);
if sum(subimg(:)) == 2*255
result(i-1,j-1) = 255;
continue;
elseif sum(subimg(:)) > 3*255
continue;
end
for k=1:3:22
mask = B(:,k:k+2);
flag = 0;
for x=1:1:3
for y=1:1:3
if mask(x, y) == 2
continue;
end
if mask(x, y) ~= subimg(x, y)
flag = 1;
break;
end
end
if flag == 1
break;
end
end
if flag == 0
result(i-1,j-1) = 255;
break;
end
end
end
end
figure();
imshow(result);
```

Automatic Global Thresholding:

```
img = imread("For_asign1.jpg");
figure();
imshow(img);
img=double(img);
[m, n] = size(img);
epsilon = 0.1;
N1 = 0.0;
D1 = 0;
N2 = 0.0;
D2 = 0;
T = mean(img(:)); % Initial value of T
for i=1:m
for j=1:n
if img(i,j) <= T
D1 = D1+1;
N1 = N1+img(i, j);
else
D2 = D2+1;
N2 = N2+img(i, j);
end
end
end
newT = ((N1/D1)+(N2/D2))/2;
fprintf("T:%f , newT:%f\n", T, newT);
while abs(newT-T) > epsilon
T = newT;
N1 = 0.0;
D1 = 0;
N2 = 0.0;
D2 = 0;
for i=1:m
for j=1:n
if img(i,j) <= T
D1 = D1+1;
N1 = N1+img(i, j);
else
D2 = D2+1;
N2 = N2+img(i, j);
end
end
end
newT = ((N1/D1)+(N2/D2))/2;
fprintf("T:%f , newT:%f\n", T, newT);
end
result = img;
for i=1:m
for j=1:n
if img(i,j) <= newT
result(i,j)=0;
else
result(i,j)=255;
end
end
end
figure();
imshow(result);
```

Connected Components:

```
img=imread('cc2.png');
img=rgb2gray(img);
img=imresize(img,1/4);
[m,n]=size(img);
fprintf("m:%d n:%d", m, n);
threshold=mean(img(:));
for i=1:m
for j=1:n
if img(i,j) <= threshold
img(i,j)=0;
else
img(i,j)=255;
end
end
end
figure();
imshow(img);
img=double(img);
mask=zeros(m,n);
count=1;
for i=1:m
for j=1:n
if img(i,j)==0
continue;
end
if mask(i,j)~=0
continue;
end
seed=zeros(m,n);
seed(i,j)=255;
seed = conditioned_dilation(img,seed);
for x=1:m
for y=1:n
if seed(x,y)==255
mask(x,y)=count;
end
end
end
fprintf("Count: %d\n", count);
count=count+1;
end
end
for k=1:count-1
result=zeros(m,n);
for x=1:m
for y=1:n
if mask(x,y)==k
result(x,y)=255;
end
end
end
figure();
imshow(result);
end
```

conditioned_dilation.m:

```
function f=conditioned_dilation(img,seed)
iteration=0;
newseed=dilation_intersection(img,seed);
while sum(seed(:))~=sum(newseed(:))
if mod(iteration,100)==0
fprintf("%d ",iteration);
end
iteration = iteration+1;
seed=newseed;
newseed=dilation_intersection(img,seed);
end
fprintf("\n");
f=newseed;
End
```

dilation_intersection.m:

```
function f=dilation_intersection(img,newseed)
[m, n] = size(newseed);
strelSize = 5;
h = (strelSize-1)/2;
newseed = [zeros(m, h), newseed, zeros(m, h)];
newseed = [zeros(h, n+(2*h)); newseed; zeros(h, n+(2*h))];
result = zeros(m, n);
for i=1+h:m+h
for j=1+h:n+h
subimg = newseed(i-h:i+h, j-h:j+h);
vals = subimg(:);
[p, q] = size(vals); % q will always be 1
for k= 1:p
if vals(k) == 255,
result(i-h, j-h) = 255;
break;
end
end
end
End
% Intersection:
for i=1:m
for j=1:n
if result(i,j)==255 && img(i,j)~=255
result(i,j)=0;
end
end
end
f=result;
end
```

Gradient:

```
a=imread('gradient.png');
a=rgb2gray(a);
figure();
imshow(a);
a=double(a);
mask1=[-1,0,1;-1,0,1;-1,0,1];
mask2=[-1,-1,-1;0,0,0;1,1,1];
mask3=[-1 -1 0; -1 0 1; 0 1 1];
mask4=[0 1 1; -1 0 1; -1 -1 0];
[x, y]=size(a);
b=zeros(x,y);
for i=2:1:x-1
for j=2:1:y-1
for k=1:1:3
for l=1:1:3
b(i,j)=b(i,j)+mask1(k,l)*a(i+k-2,j+l-2);
end
end
end
end
b=uint8(b);
c=zeros(x,y);
for i=2:1:x-1
for j=2:1:y-1
for k=1:1:3
for l=1:1:3
c(i,j)=c(i,j)+mask2(k,l)*a(i+k-2,j+l-2);
end
end
end
end
c=uint8(c);
e=zeros(x,y);
for i=2:1:x-1
for j=2:1:y-1
for k=1:1:3
for l=1:1:3
e(i,j)=e(i,j)+mask3(k,l)*a(i+k-2,j+l-2);
end
end
end
end
e=uint8(e);
f=zeros(x,y);
for i=2:1:x-1
for j=2:1:y-1
for k=1:1:3
for l=1:1:3
f(i,j)=f(i,j)+mask4(k,l)*a(i+k-2,j+l-2);
end
end
end
end
f=uint8(f);
d=zeros(x,y);
for i=2:1:x-1
for j=2:1:y-1
d(i,j)=abs(b(i,j))+abs(c(i,j))+abs(e(i,j))+abs(f(i,
j));
end
end
figure();
imshow(d, []);
```