Image Processing Matlab / Octave Code

2021 Fall English-taught Program.

1. Basic Instruction

Put a semicolon after one line of code to make the execution silent.

Press Ctrl + C to break execution.

To load required package, use `pkg load image`

We put all the sample images in `c:\im\`

2. Load Image

```
img = imread('c:\im\cameraman.png');
```

3. Show Image

```
Imshow(img);
```

4. Convert Color Image to Gray Image

```
img = imread('c:\im\kookaburra2.jpg');
img = rgb2gray(img);
imshow(img)
```

Original Image

After Processing

5. Reverse

```
img = imread('c:\im\cameraman.png');
img = 255-img;
imshow(img)
```

Original Image

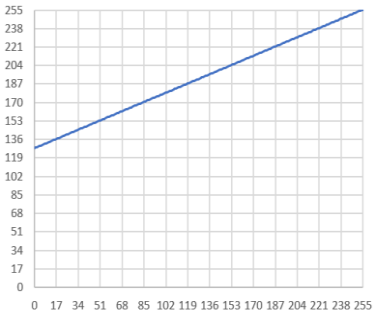
After Processing

6. Adjustment

```
img = imread('c:\im\cameraman.png');
img = img/2+128;
imshow(img)
```

Original Image **After Processing** **Adjustment Curve**

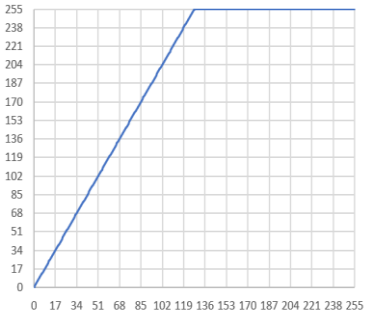





```
img = imread('c:\im\cameraman.png');
img = img*2;
imshow(img)
```

Original Image **After Processing** **Adjustment Curve**

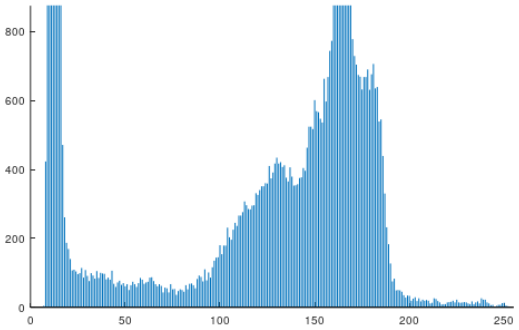




7. Histogram

```
img = imread('c:\im\cameraman.png');
imhist(img)
```

Original Image **Histogram**

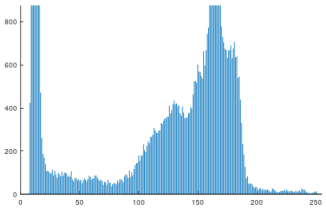
8. Histogram Equalization

```
img = imread('c:\im\cameraman.png');
img = histeq(img);
imshow(img)
```

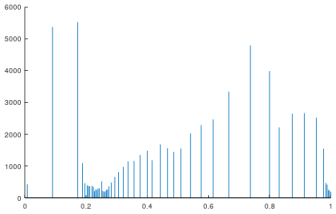
Original Image

After Processing

Original Histogram



Processed Histogram



9. Smooth

```
img = imread('c:\im\cameraman.png');
fl = fspecial('average', 3)
img = imfilter( img, fl );
imshow(img)
```

Original Image

After Processing

10. Unsharp Filter

```
img = imread('c:\im\cameraman.png');
fl = fspecial('unsharp', 0.5);
img = imfilter( img, fl );
imshow(img)
```

Original Image

After Processing

11. High Boost Filter

```
img = imread('c:\im\cameraman.png');
fl = [-1 -1 -1; -1 11 -1; -1 -1 -1]/9;
img = imfilter( img, fl );
imshow(img)
```

Original Image

After Processing

12. Diagonal Line Enhancement

```
img = imread('c:\im\cameraman.png');
fl = [9 -1 -1; -1 9 -1; -1 -1 9]/9;
img = imfilter( img, fl );
imshow(img)
```

Original Image

After Processing

13. Horizontal Line Detection

```
img = imread('c:\im\cameraman.png');
fl = [-1 -1 -1; 2 2 2; -1 -1 -1];
img = imfilter( img, fl );
imshow(img)
```

Original Image

After Processing

14. Vertical Line Detection

```
img = imread('c:\im\cameraman.png');
fl = [-1 2 -1; -1 2 -1; -1 2 -1];
img = imfilter( img, fl );
imshow(img)
```

Original Image

After Processing

15. Diagonal Line Detection

```
img = imread('c:\im\cameraman.png');
fl = [2 -1 -1; -1 2 -1; -1 -1 2];
img = imfilter( img, fl );
imshow(img)
```

Original Image

After Processing

16. Max Filter

```
img = imread('c:\im\cameraman.png');
cmax = ordfilt2( img, 9, ones(3,3) );
imshow(cmax)
```

Original Image

After Processing

17. Min Filter

```
img = imread('c:\im\cameraman.png');
cmin = ordfilt2( img, 1, ones(3,3) );
imshow(cmin)
```

Original Image

After Processing

18. Median Filter

```
img = imread('c:\im\cameraman.png');
cmed = ordfilt2( img, 5, ones(3,3) );
imshow(cmed)
```

Original Image

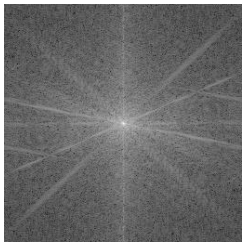
After Processing

19. Fourier Transform

```
pkg load image
img = imread('c:\im\cameraman.png');
cf = fftshift(fft2(img));
imshow(mat2gray(log(1+abs(cf))));
```

Spatial Domain

Frequency Domain

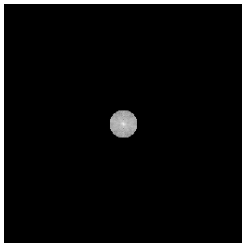


20. Ideal Low Pass & High Pass Filter

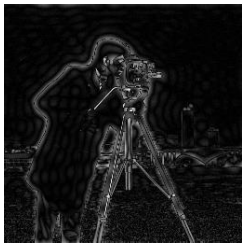
```
pkg load image
img = imread('c:\im\cameraman.png');
cf = fftshift(fft2(img));
[x, y] = meshgrid(-128:127, -128:127);
z = sqrt(x.^2+y.^2);
cfl = cf.*(z<15);
imshow(mat2gray(log(1+abs(cfl))));
cfli = ifft2(cfl);
figure,imshow(mat2gray(abs(cfli)));
cfh = cf.*(z>15);
imshow(mat2gray(log(1+abs(cfh))));
cfhi = ifft2(cfh);
figure,imshow(mat2gray(abs(cfhi)));
```

Low Pass Spatial Domain

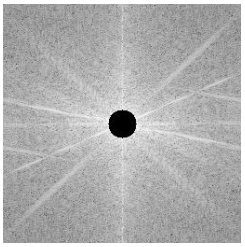
Low Pass Freq Domain



High Pass Spatial Domain



High Pass Freq Domain

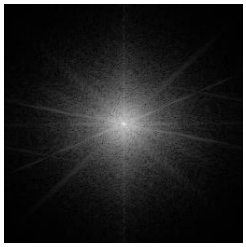


21. Butterworth Low Pass Filter

```
pkg load image
img = imread('c:\im\cameraman.png');
cf = fftshift(fft2(img));
[x, y] = meshgrid(-128:127, -128:127);
bl = 1./(1+((x.^2+y.^2)/15.^2).^2);
cfl = cf.*bl;
imshow(mat2gray(log(1+abs(cfl))));
cfli = ifft2(cfl);
figure,imshow(mat2gray(abs(cfli)));
```

Low Pass Spatial Domain

Low Pass Frequency Domain

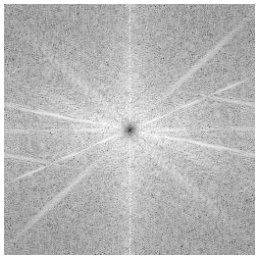


22. Butterworth High Pass Filter

```
pkg load image
img = imread('c:\im\cameraman.png');
cf = fftshift(fft2(img));
[x, y] = meshgrid(-128:127, -128:127);
bh = 1-1./(1+((x.^2+y.^2)/15.^2).^2);
cfh = cf.*bh;
imshow(mat2gray(log(1+abs(cfh))));
cfhi = ifft2(cfh);
figure,imshow(mat2gray(abs(cfhi)));
```

High Pass Spatial Domain

High Pass Frequency Domain



23. Homomorphic Filter

```
img = imread('c:\im\arch.png');
cutoff=128; order=2; lowgain=0.5; highgain=2;
height=size(img,1);
width=size(img,2);
u=im2uint8(img);
u(find(u==0))=1;
l=log(double(u));
ft=fftshift(fft2(l));
[x,y]=meshgrid(-floor(width/2):floor((width-1)/2),-floor(height/2):floor((height-1)/2));
f=lowgain+(highgain-lowgain)*(1-(1./(1+(sqrt(2)-1)*((x.^2+y.^2)/cutoff^2).^order)));
b=f.*ft;
ib=abs(ifft2(b));
res=exp(ib);
imshow(uint8(res*10));
```

Original Image

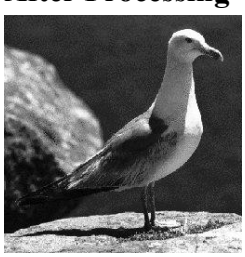
After Processing

24. Bilateral Filter

```
img = imread('c:\im\gull.jpg');
img = imsmooth( img, 'bilateral', sigma_d=2, sigma_r=0.1);
imshow(img)
```

Original Image

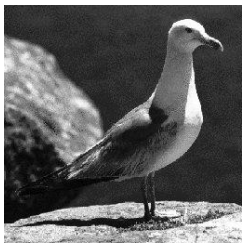
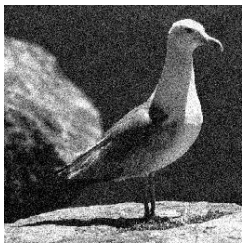
After Processing



25. Add Gaussian Noise and Remove by Average Filter

```
img = imread('c:\im\gull.jpg');
img = imnoise( img, 'gaussian');
imshow(img)
img = imsmooth(img, 'average', 3);
figure,imshow(img)
```

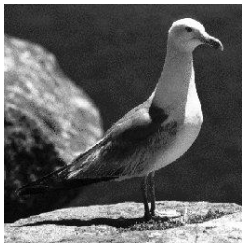
Original Image **Gaussian Noise** **After Denoising**


26. Add Salt & Pepper Noise and Remove by Median Filter

```
img = imread('c:\im\gull.jpg');
img = imnoise( img, 'salt & pepper',0.2);
imshow(img)
img = ordfilt2( img, 5, ones(3,3) );
figure,imshow(img)
```

Original Image **Salt & Pepper Noise** **After Denoising**



27. Remove Salt & Pepper Noise by Outlier Method

```
img = imread('c:\im\cameraman.png');
noise_image = imnoise( img, 'salt & pepper',0.1);
gsp = im2double(noise_image);
av = [1 1 1; 1 0 1; 1 1 1]/8;
gspa = imfilter(gsp,av);
D = 0.175;
r = abs(gsp-gspa)>D;
imshow(r.*gspa+(1-r).*gsp)
```

Noisy Image 1

Noisy Image

After Denoising

28. Remove Motion Blur by Division Constrain

```
pkg load image
img = imread('c:\im\car.png');
blur = fspecial('motion', 7, 0);
blur_image = imfilter(img, blur);
imshow(blur_image);
d=0.09;
m2 = zeros(size(img));
m2(1, 1:7) = blur(floor(7/2)+1,:);
mf = fft2(m2);
mf( find(abs(mf)<d) ) = 1;
ni = ifft2(fft2(blur_image)./mf);
figure,imshow(mat2gray(abs(ni))*2);
```

Original Image

Motion Blur Image

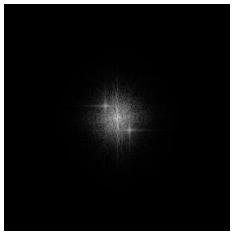
After Deblurring

29. Remove Periodic Noise by Criss-Cross Filtering

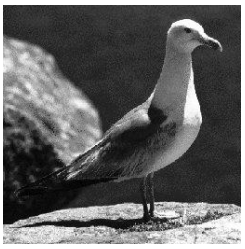
```
pkg load image
img = imread('c:\im\gull.jpg');
[rs, cs] = size(img);
[x, y] = meshgrid(1:rs, 1:cs);
p = sin(x/3+y/3)+1;
noise_image = (2*im2double(img)+p/2)/3;
imshow(noise_image);
cf = fftshift(fft2(noise_image));
[x, y] = meshgrid(-128:127, -128:127);
bl = 1./(1+((x.^2+y.^2)/15.^2).^2);
cfl = cf.*bl;
imshow(mat2gray(log(1+abs(cfl))));
cfl(:,112:118)=0;
cfl(112:118,:)=0;
cfl(:,142:148)=0;
cfl(142:148,:)=0;
figure,imshow(mat2gray(log(1+abs(cfl))));
cfli = ifft2(cfl);
figure,imshow(mat2gray(abs(cfli)));
```

Noisy Image

Frequency Domain

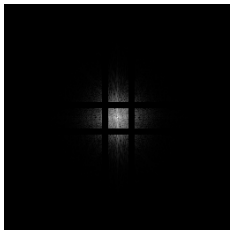


Original Image



After Denoising

Frequency Domain



30. Remove Gaussian Noise by Image Averaging

```
img = imread('c:\im\cameraman.png');
t = zeros( [ size(img), 5 ] );
for i=1:5
    t(:,:,i) = imnoise(img, 'gaussian');
    figure,imshow( uint8( t(:,:,i) ) );
    imwrite( uint8( t(:,:,i)), ['c:\im\result', int2str(i), '.png'] )
end
ta = mean(t,3);
figure,imshow( uint8( ta ) );
imwrite(uint8( ta ), 'c:\im\result.png')
```

Noisy Image 1

Noisy Image 2

Noisy Image 3

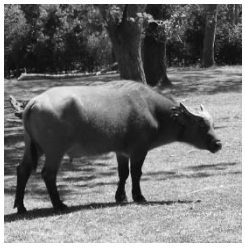
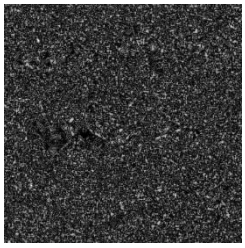
Noisy Image 4

Noisy Image 5

Average Image

31. Inverse from Butterworth Filter

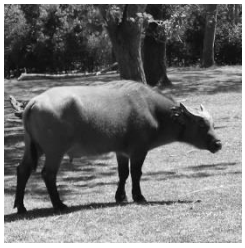
```
pkg load image
b = imread('c:\im\buffalo.png');
bf = fftshift(fft2(b));
[r, c] = size(b);
[x, y] = meshgrid(-c/2:c/2-1, -r/2:r/2-1);
bworth = 1./(1+(sqrt(2)-1)*((x.^2+y.^2)/15^2).^2);
bw = bf.*bworth;
bwa=abs(ifft2(bw));
blur = im2uint8(mat2gray(bwa));
imshow(blur);
blf = fftshift(fft2(blur));
blfw = blf./bworth;
bla = abs(ifft2(blfw));
figure,imshow(mat2gray(bla));
D=40;
bworth2 = 1./(1+(sqrt(2)-1)*((x.^2+y.^2)/D^2).^10);
blfb = blf./bworth.*bworth2;
ba=abs(ifft2(blfb));
figure,imshow(mat2gray(ba));
```

Original Image	**Blurred Image**	**Inverse by Division**	**Constrained Division**
			

32. Weiner Filter

```
pkg load image
b = imread('c:\im\buffalo.png');
bf = fftshift(fft2(b));
[r, c] = size(b);
[x, y] = meshgrid(-c/2:c/2-1, -r/2:r/2-1);
bworth = 1./(1+(sqrt(2)-1)*((x.^2+y.^2)/15^2).^2);
bw = bf.*bworth;
bwa=abs(ifft2(bw));
blur = im2uint8(mat2gray(bwa));
blur = imnoise(blur, 'gaussian');
imshow(blur);
blf = fftshift(fft2(blur));
D=40;
bworth2 = 1./(1+(sqrt(2)-1)*((x.^2+y.^2)/D^2).^10);
b = bworth;
K=0.00001;
blfb = blf.*((abs(b.^2)./(abs(b.^2)+K))./b);
blfb = blfb .*bworth2;
ba=abs(ifft2(blfb));
figure,imshow(mat2gray(ba));
K=0.1;
blfb = blf.*((abs(b.^2)./(abs(b.^2)+K))./b);
blfb = blfb .*bworth2;
ba=abs(ifft2(blfb));
figure,imshow(mat2gray(ba));
```

Original Image **Blur & Noise** **K=0.00001** **K=0.1**

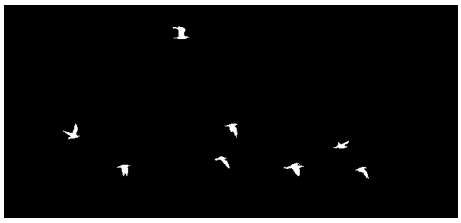



33. Thresholding

```
img = imread('c:\im\flying.png');
binary_image = img<50;
imshow(binary_image);
```

Original Image

After Processing

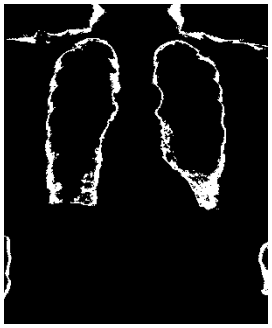


34. Double Thresholding

```
img = imread('c:\im\xray.jpg');
img = rgb2gray(img);
binary_image = img>50 & img<80;
imshow(binary_image);
```

Original Image

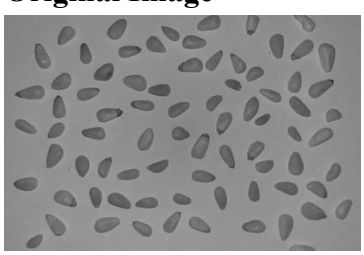
After Processing



35. Otsu's Method

```
img = imread('c:\im\pinenuts.png');
t = graythresh(img)
binary_image = img > t*256;
binary_image = 1 - binary_image;
imshow(binary_image);
```

Original Image



After Processing

36. Sobel Edge Detector

```
img = imread('c:\im\cameraman.png');
fl = [1 0 -1; 1 0 -1; 1 0 -1 ];
img = imfilter( img, fl );
imshow(img)
```

Original Image

After Processing

37. Laplacian Edge Detector

```
img = imread('c:\im\cameraman.png');
fl = fspecial('laplacian')
img = imfilter( img, fl );
imshow(img)
```

Original Image

After Processing

38. Canny Edge Detector

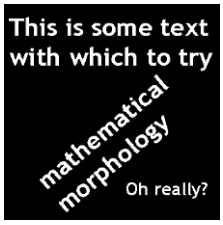
```
img = imread('c:\im\cameraman.png');
img = edge(img, 'canny');
imshow(img)
```

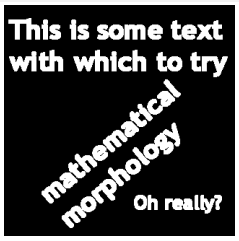
Original Image

After Processing

39. Dilation

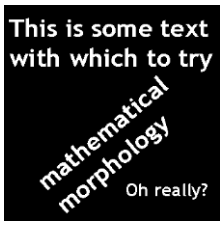
```
img = imread('c:\im\morph_text.png');
img = imdilate( img, ones(3,3) );
imshow(img)
```

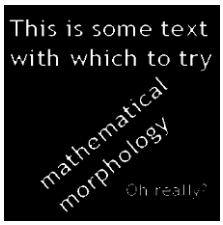




40. Erosion

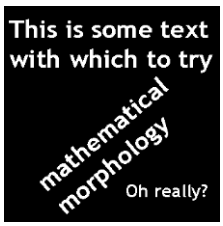
```
img = imdilate( img, ones(3,3) );
imshow(img)
```

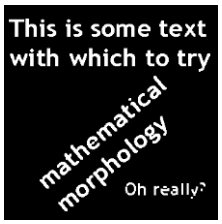




41. Open Operation

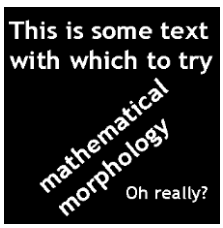
```
img = imopen( img, ones(3,3) );
imshow(img)
```

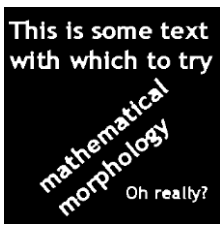




42. Close Operation

```
img = imclose( img, ones(3,3) );
imshow(img)
```



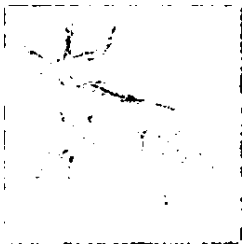


43. Harris Corner Detector

```
img = imread('c:\im\caribou.jpg');
sigma = 1; k = 0.01;
gauss = fspecial('gaussian',max(1,fix(6*sigma)),sigma);
sob = [1 2 1;0 0 0;-1 -2 -1];
imx = filter2(sob,img,'same');
imy = filter2(sob',img,'same');
Axx = filter2(gauss,imx.^2);
Axy = filter2(gauss,imx.*imy);
Ayy = filter2(gauss,imy.^2);
detA = Axx.*Ayy - Axy.^2;
trA = Axx+Ayy;
out = detA - k*trA.^2;
imshow(out);
```

Original Image

After Detection



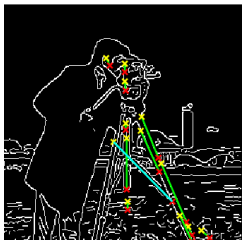
44. Find Hough Lines by Hough Transform

```
img = imread('c:\im\cameraman.png');
BW = edge(img, 'canny');
figure, imshow(BW), hold on
max_len = 0;
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');

    % Plot beginnings and ends of lines
    plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
    plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

    % Determine the endpoints of the longest line
segment
    len = norm(lines(k).point1 - lines(k).point2);
    if ( len > max_len)
        max_len = len;
        xy_long = xy;
    end
end

% highlight the longest line segment
plot(xy_long(:,1),xy_long(:,2),'LineWidth',2,'Color','cyan');
```

Original Image

Canny

Hough Lines



45. Connected Components – 4 Adjacency & 8 Adjacency

```
im = zeros(8,8);
im(2:4, 3:6) = 1;
im(5:7, 2) = 1;
im(6:7, 5:8) = 1;
im(8, 4:5) = 1;
im4 = bwlabel(im, 4);
im4
im8 = bwlabel(im, 8);
im8
```

Input

0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	0	0
0	1	0	0	1	1	1	1
0	1	0	0	1	1	1	1
0	0	0	1	1	0	0	0

4-Adjacency

0	0	0	0	0	0	0	0
0	0	2	2	2	2	0	0
0	0	2	2	2	2	0	0
0	0	2	2	2	2	0	0
0	1	0	0	0	0	0	0
0	1	0	0	3	3	3	3
0	1	0	0	3	3	3	3
0	0	0	3	3	0	0	0

8-Adjacency

0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	0	0
0	1	0	0	2	2	2	2
0	1	0	0	2	2	2	2
0	0	0	2	2	0	0	0

46. 4-Adjacency Chain Code

```
image = zeros(8,7);
image(2:7, 3:5) = 1; image(5:7, 2) = 1; image(3:6, 6) = 1;
n = [0 1;-1 0;0 -1;1 0]; %4 directions
dn = ['0', '1', '2', '3'];
flag = 1; chain_code = []; path = [];
[xs ys] = find(image==1);
x = min(xs);
imx = image(x,:);
y = min(find(imx==1));
first = [x y]; %the starting point
dir = 4-1;
res = zeros(size(image)); res(:)='.';
while flag==1,
    tt = zeros(1,4);
    newdir = mod(dir+3,4);
    for i=0:4-1,
        j = mod(newdir+i,4)+1;
        tt(i+1) = image(x+n(j,1),y+n(j,2));
    end
    d = min(find(tt==1));
    dir = mod(newdir+d-1,4);
    chain_code = [chain_code,dir];
    x = x+n(dir+1,1);y = y+n(dir+1,2);
    path = [path;[x y]];
    res(x, y) = dn(dir+1);
    if x==first(1) && y==first(2)
    flag=0;
    end
end
char(res)
chain_code
```

Input

```
0 0 0 0 0 0 0
0 0 1 1 1 0 0
0 0 1 1 1 1 0
0 0 1 1 1 1 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 1 1 1 1 0 0
0 0 0 0 0 0 0
```

4-Adjacency Path

```
.......
..221..
..3.21.
..3..1.
.23..1.
.3..10.
.3000..
.......
```

Chain Code: 3 3 3 2 3 3 0 0 0 1 0 1 1 1 2 1 2 2

47. 8-Adjacency Chain Code

```
image = zeros(8,7);
image(2:7, 3:5) = 1; image(5:7, 2) = 1; image(3:6, 6) = 1;
n = [0 1;-1 1;-1 0;-1 -1;0 -1;1 -1;1 0;1 1]; %8 directions
dn = ['0', '1', '2', '3','4','5','6','7'];
flag = 1; chain_code = []; path = [];
[xs ys] = find(image==1);
x = min(xs);
imx = image(x,:);
y = min(find(imx==1));
first = [x y]; %the starting point
dir = 8-1;
res = zeros(size(image)); res(:)='.';
while flag==1,
    tt = zeros(1,8);
    newdir = mod(dir+7-mod(dir,2),8);
    for i=0:8-1,
        j = mod(newdir+i,8)+1;
        tt(i+1) = image(x+n(j,1),y+n(j,2));
    end
    d = min(find(tt==1));
    dir = mod(newdir+d-1,8);
    chain_code = [chain_code,dir];
    x = x+n(dir+1,1);y = y+n(dir+1,2);
    path = [path;[x y]];
    res(x, y) = dn(dir+1);
    if x==first(1) && y==first(2)
    flag=0;
    end
end
char(res)
chain_code
```

Input

```
0 0 0 0 0 0 0
0 0 1 1 1 0 0
0 0 1 1 1 1 0
0 0 1 1 1 1 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 1 1 1 1 0 0
0 0 0 0 0 0 0
```

8-Adjacency Path

```
. . . . . . .
. . 4 4 3 . .
. . 6 . . 2 .
. . 6 . . 2 .
. 5 . . . 2 .
. 6 . . . 1 .
. 6 0 0 0 . .
. . . . . . .
```

Chain Code: 6 6 5 6 6 0 0 0 1 2 2 2 3 4 4

48. Color Image Channel Separation

```
pkg load image
img = imread('c:\im\flower.png');
red = img(:,:,1);
green = img(:,:,2);
blue = img(:,:,3);
figure,imshow(red);
figure,imshow(green);
figure,imshow(blue);
```

Red Part

Green Part

Blue Part

49. Color Image Histogram Equalization on Each Channel

```
pkg load image
img = imread('c:\im\flower.png');
red = img(:,:,1);
green = img(:,:,2);
blue = img(:,:,3);
red = histeq(red) * 255;
blue = histeq(blue) * 255;
green = histeq(green) * 255;
red = uint8(red);
green = uint8(green);
blue = uint8(blue);
img(:,:,1) = red;
img(:,:,2) = green;
img(:,:,3) = blue;
figure,imshow(img);
```

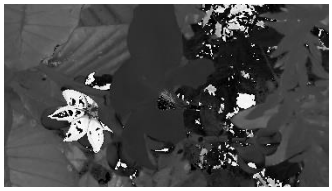
Original Image

Equalized Image

50. RGB to HSV Image

```
pkg load image
img = imread('c:\im\flower.png');
hsv = rgb2hsv(img);
h = hsv(:,:,1);
s = hsv(:,:,2);
v = hsv(:,:,3);
figure,imshow(h);
figure,imshow(s);
figure,imshow(v);
```

H Part



S Part

V Part

51. Color Image Equalization on V Channel Only

```
pkg load image
img = imread('c:\im\flower.png');
hsv = rgb2hsv(img);
h = hsv(:,:,1);
s = hsv(:,:,2);
v = hsv(:,:,3);
v = histeq(v);
hsv(:,:,3) = v;
img = hsv2rgb(hsv);
figure,imshow(img);
```

Original Image

Equalized Image