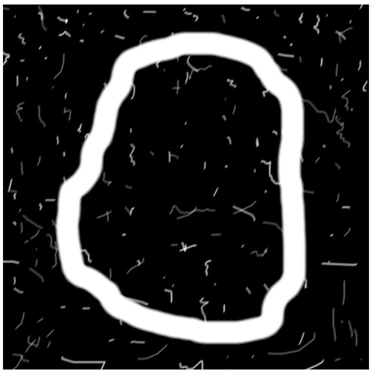
ECE 565 – Image Processing and Computer Vision

Thomas BOOT[1]

[1]IIT, Illinois Institute of Technology, 3300 S Federal St, Chicago, IL 60616

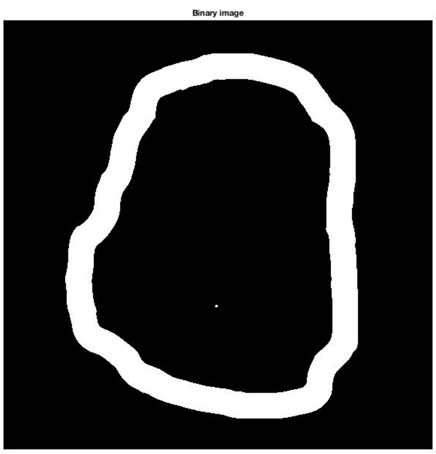
tboot@hawk.iit.edu

1. Chain codes

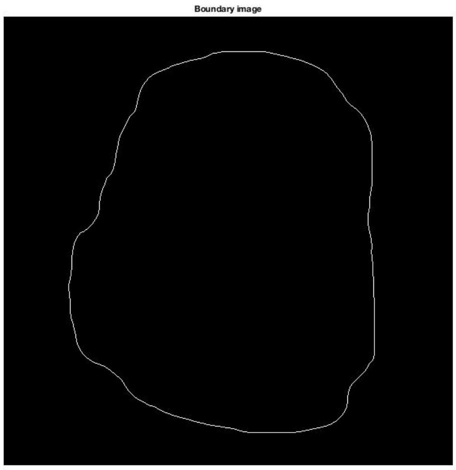


1.1 Generate a smoothed image g using 9x9 averaging filter

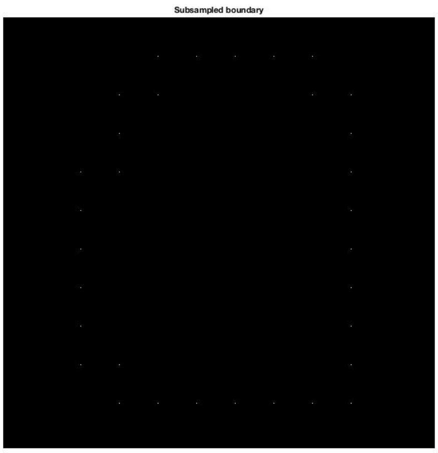
1.2 Generate a binary image gB by thresholding g obtained in part 1.1

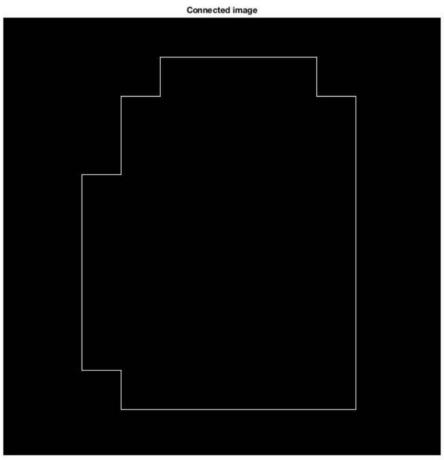


1.3 Extract the outer boundary of gB and display the results as a binary image.



1.4 Subsample the boundary obtained in Part 1.3 onto a grid whose lines are separated by 50 pixels. Connect the subsampled boundary points with straight line segments. Display the resulting points as a binary image.





1.5 Write a program that computes the Freeman chain code c of a boundary b with the code connectivity specified in CONN. The input b is a set of 2-D coordinate pairs for a boundary and CONN can be 8 for an 8-connected chain code or 4 for a 4-connected chain code.

Chain code starting point :
x0y0: [3 9]

Chain code (1x32) : (32 being the number of boundary pixels)
fcc: [3 0 3 0 0 0 0 0 0 1 1 1 1 1 1 1 1 2 1 2 2 2 2 3 2 3 3 2 3 3 3 3]

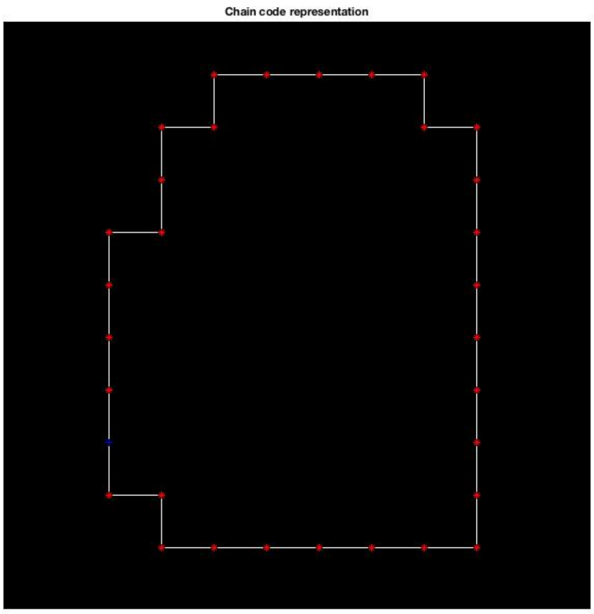
Difference code of fcc (1x32) :
diff: [1 3 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 3 1 0 0 0 1 3 1 0 3 1 0 0 0 0]

Integer of minimum magnitude from the chain code (1x32):
mm: [0 0 0 0 0 0 1 1 1 1 1 1 1 1 2 1 2 2 2 2 3 2 3 3 2 3 3 3 3 3 0 3]

First difference code of the minimum magnitude (1x32) :

diffmm: [0 0 0 0 0 1 0 0 0 0 0 0 0 1 3 1 0 0 0 1 3 1 0 3 1 0 0 0 0 1 3 1]

Here the chain code starts at the blue dotted point in the figure below. The chain code is in anti-clockwise order.



1.6 Appendix Code

```
% Part 1 : CHAIN CODE
%%
clc
clear all
close all

image = imread('Figure1.tif');
[M,N] =  size(image);

% (a) Generate a smoothed image g using 9x9 filter

h = fspecial('average', [9 9]);
g = filter2(h, image);

% (b) Generate a binary image gB

for i=1:M
    for j=1:N
        if g(i,j)<128
            gB(i,j)=0;
        else
            gB(i,j)=1;
        end
    end
end

% (c) Extract outer boundary of gb
```

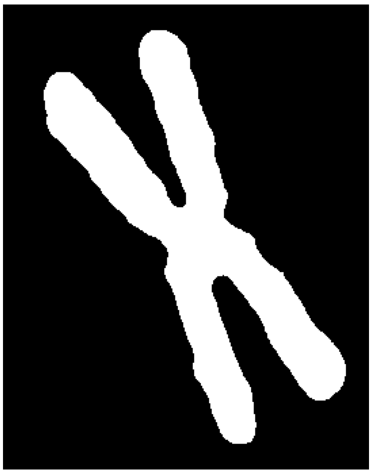
```
B = bwboundaries(gB, 8, 'noholes');
figure, imshow(gB); hold on
k = 1;
boundary = cell2mat(B(k));
plot(boundary(:,2), boundary(:,1), 'r', 'LineWidth', 3);
G = bound2im(boundary, M, N);


%% (d) Subsample the boundary obtained in (c) onto a grid whose lines are
% separated by 50 pixels.

[s, sUnit] = bsubsamp(boundary,50);
sub = bound2im(s,M,N);
cn = connectpoly(s(:,1), s(:,2));
connec = bound2im(cn,M,N);

figure,
set(gcf,'Color','white'),imshow(image, []), title('Original image')
figure,
set(gcf,'Color','white'), imshow(uint8(g)), title('Smoothed image')
figure,
set(gcf,'Color','white'), imshow(gB,[]), title('Binary image')
figure,
set(gcf,'Color','white'), imshow(G), title('Boundary image')
figure,
set(gcf,'Color','white'),imshow(sub,[]), title('Subsampled boundary')
figure,
set(gcf,'Color','white'),imshow(connec, []), title('Connected image')

figure, imshow(connec, [])
%% Freeman chain code
clc
close all
c = fchcode(sUnit, 4)

sF =  flipdim(s,1);
sF =  flipdim(sF,2);
figure, set(gcf,'Color','white'),imshow(connec),hold on, title('Chain code
representation'), plot(sF(1,1), sF(1,2), '+b')
for i = 2: length(sF)
    plot(sF(i,1),sF(i,2), '*r');
end
%%%%%%%%
%%%%%%%%
%%%%%%%%

function [ c ] = fchcode(b, conn)
%       c.fcc = chain code (1 x np) where np is the number of boundary pixels
%       c.diff = First difference of code c.fcc (1 x np)
%       c.mm = Integer of minimum magnitude from c.fcc (1 x np)
%       c.diffmm = First difference of code c.mm (1 x np)
%       c.x0y0 = Coordinates where the code starts (1 x 2)

if nargin == 1
    conn = 4;
elseif nargin == 0
    error('Too few arguments')
elseif nargin > 2
    error('Too many input arguments')
end

[M, N] = size(b);
```

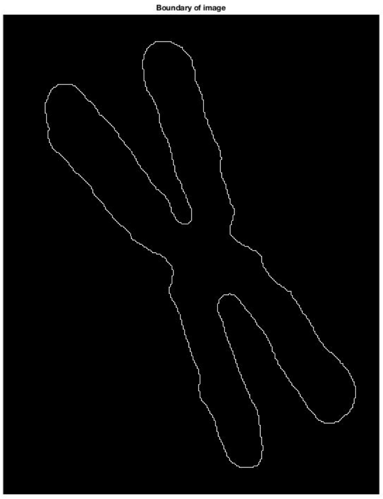
```
if M < N
    error('B must be of size np-by-2.');
end
b =  flip(b,1); %flip the elements in each column to connected subsampled boundary
b =  flip (b,2); % flip the elements in each row match connected subsampled boundary
x0 = b(1,1);
y0 = b(1,2);
c.x0y0 = [x0, y0];

%Compute dx, dy by circular shift on coordinates array by 1 element
sb = circshift(b,[-1, 0]);
delta = sb - b; % contains dx and dy between successive points in b.
delta(:,2)= - delta(:,2);
delta = delta/(max(max(delta))); % Compute the delta in range [-1;1];


%Check if boundary is close
if abs(delta(end,1))>1|| abs(delta(end,2))>1;
    warning('The input curve is broken, we will cut the last end');
    delta = delta(1:(end-1),:);
end

C(11)=1;C(6)=2;C(9)=3;C(14)=0;
z = 4*(delta(:,1)+2)+(delta(:,2)+2);

fcc = C(z);

% 4-connectivity
if conn == 4
    index = find(fcc == 1 | fcc == 3 | fcc == 5 | fcc == 7);
    if isempty(index)
        fcc = fcc./2;
    else
        warning('The 4-connected code cannot be satisfied')
    end
end

% Freeman chain
c.fcc = fcc;

% Diff code
c.diff = codediff(c.fcc, conn);

% Integer of minimum magnitude
c.mm = minmagnitude(c.fcc);

% Diff code c.mm
c.diffmm = codediff(c.mm, conn);

end

%%%%%%%%
%%%%%%%%
%%%%%%%%

function [ d ] = codediff( fcc, conn )
sr = circshift(fcc, [0, -1]);
delta = sr - fcc;
d = delta;
idx = find(delta < 0);

type = conn;
switch type
```

```
    case 4
        d(idx) = d(idx)+4;
    case 8
        d(idx) = d(idx)+8;
end

end

%%%%%%%%
%%%%%%%%
%%%%%%%%


function [ mm ] = minmagnitude( c )

zeroIndex = find(c == min(c));
A = zeros(length(zeroIndex), length(c));

idx = 0;
for k = zeroIndex;
    idx = idx + 1;
    A(idx,:) = circshift(c,[0 -(k-1)]);
end
N = zeros(size(A,1),1);
for i=1:size(A,1)
    r = 0;
    for j=1:size(A,2)
        if A(i,j)==0
            r = r+1;
        else
            break
        end
    end
    N(i)=r;
end
I = find(N ==max(N));
mm = A(I,:);
end
```

2. Fourier descriptors

2.1 Extract the boundary of the chromosome and display the result as a binary image





2.2 Write a program to compute the Fourier descriptors of a boundary s (i.e., z = fourierdescp(s)). The input s is an 2 sequence of ordered coordinates describing a boundary and the output z is a sequence of Fourier descriptors obtained. Compute the Fourier descriptors for the boundary obtained in Part 2.1.

```
function [ z ] = fourierdescp( s )

[M,N] = size(s);
if N ~=2
    error('S must be of size M by 2')
end
if M/2 ~= round(M/2)
    s(end+1,:)=s(end,:);
    M = M+1;
end

% Centering the transform
x=0:(M-1);
m=((-1).^x)';
s(:,1) = m.*s(:,1);
s(:,2) = m.*s(:,2);

s = s(:,1) + 1i*s(:,2);
z = fft(s);

end
```

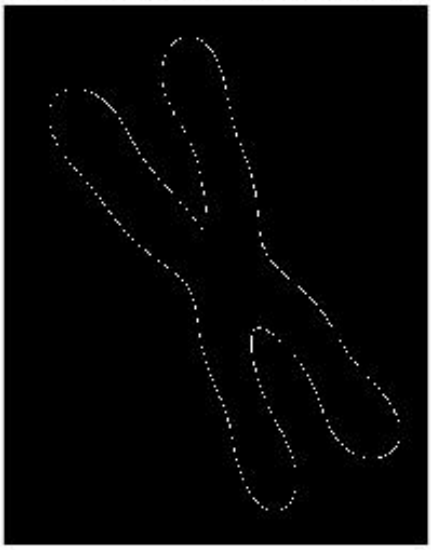
2.3 Write a program to compute the inverse Fourier descriptors (i.e., s = ifourierdescp(z, nd)). The input z is a sequence of Fourier descriptors and nd is the number of descriptors used to compute the inverse. nd must be an even integer no greater than length(z). Reconstruct the boundary using 50% of the total possible descriptors and display the result as a binary image. Then, reconstruct the boundary using 1% of the total possible descriptors and display the result as a binary.

```
function [ s1 ] = ifourierdescp( z, nd )

M = length(z);
if nargin == 1 || nd > M
    nd = M;
end

if rem(nd,2) ~=0
    error('Descriptor must be even');
end

% Centering the transform
x = 0:(M-1);
m = ((-1).^x)';

% Use only nd descriptors
% as it is centered (M-nd)/2 from each end of the sequence are set to 0.
d = round((M - nd)/2);
z(1:d)=0;
z(M - d +1 : M)=0;

% Inverse and convert to coordinates
z2 = ifft(z);
s1(:,1) = real(z2);
```

```
s1(:,2) = imag(z2);

% Undo centering
s1(:,1) = m.*s1(:,1);
s1(:,2) = m.*s1(:,2);


end
```

Fourier Descriptor inverse image 50%

Fourier Descriptor inverse image 1%



```
% Part 2 : FOURIER DESCRIPTORS

clc
clear all
close all

%% (a)
image = imread('Figure2.tif');
[M,N] =  size(image);

B = bwboundaries(image, 'noholes');
figure, imshow(image); hold on
boundary = cell2mat(B(1));
plot(boundary(:,2), boundary(:,1), 'r', 'LineWidth', 3);
G = bound2im(boundary, M, N);

figure, set(gcf,'Color','white'),
imshow(G,[]), title('Boundary of image');

%% (b)
z = fourierdescp(boundary);

%% (c) np must not be longer than length(z)
% Plot at 50% and 1% of total length(z)
fdescptot = length(z);
fdescp50 = length(z)*0.5;
fdescp1 = floor(length(z)*0.01);

stot = ifourierdescp(z,fdescptot);
```

```
s50 = ifourierdescp(z,fdescp50);
s1 = ifourierdescp(z,fdescp1);

simtot = bound2im(stot,M,N);
sim50 = bound2im(s50,M,N);
sim1 = bound2im(s1,M,N);

figure, set(gcf,'Color','white')
subplot 221, imshow(image, []), title('Original image');
subplot 222, imshow(G,[]), title('Boundary of image');
subplot 223, imshow(sim50,[]), title('Fourier Descriptor inverse image 50%');
subplot 224, imshow(sim1,[]), title('Fourier Descriptor inverse image 1%');
```