# EECS 442 Homework #4

Yi Yang

11/25/2016

## Problem 1 Harris Corner Detection

### (a)

Steps for Harris Corner Detection:

- Compute partial derivatives at each pixel
- Compute second moment matrix M in a Gaussian window around each pixel
- Compute corner response function R
- Threshold R
- Find local maxima of response function (nonmaximum suppression)

The size we chose for the Gaussian window is $w = 3$. Note that in the codes we use Sobel filter as the derivative filter as the approximation of gradient at each pixel, other derivative filters include Prewitt filter or Robert filter. By applying derivative filter, we can easily calculate the second moment matrix at each pixel. The methods using ellipses and corner response function are both coded and Matlab function $ELLIPSE()$ is used to plot ellipses in the original figures.

```
%% EECS 442 - HW 04 - Q1 Harris Corner Detection

%  Date: 11/25/2016
%  by Yi Yang

%  used helper function
%  --------------------
%  ellipse(ra,rb,ang,x0,y0,C,Nb)

%  Instructions
%  ------------
%  Implement the Harris corner detector by computing the second moment
%  matrix M at each pixel location. Compute the eigenvalues of M at every
%  pixel locations. Compute the ellipses associated to such eigenvalues.
%  Use the corner response R to assess whether a corner has been detected
%  or not. Superimpose corners with R > T on the two images.

%% Initialization
clear; close all; clc
%% ======================== Part 0: Load data ============================
img1 = imread('homework4/I1.png');
img2 = imread('homework4/I3.png');

% img = img1;
img = img2;
```

```
w = 3;

%% ========== Part 1: compute M,R and ellipse parameter ==================

% applying sobel edge detector

img_gray = img;
if length(size(img))>2
    img_gray = rgb2gray(img);
end

% compute the image size
[height,width] = size(img_gray);
% initialize corner position
result = zeros(height, width);
% initialize R score
R = zeros(height, width);

%%applying sobel edge detector in the horizontal direction
fx = [-1 0 1;-1 0 1;-1 0 1];
Ix = filter2(fx,img_gray);
% applying sobel edge detector in the vertical direction
fy = [1 1 1;0 0 0;-1 -1 -1];
Iy = filter2(fy,img_gray);

% compute terms for M
Ix2 = Ix.^2;
Iy2 = Iy.^2;
Ixy = Ix.*Iy;
clear Ix;
clear Iy;

% perform gaussian filtering for eliminating noises
h= fspecial('gaussian',[w w],2); % predefined gaussian filter
Ix2 = filter2(h,Ix2);
Iy2 = filter2(h,Iy2);
Ixy = filter2(h,Ixy);

ellipsePerem = [];
Rmax = 0;
figure;
imshow(img);
hold on;
for i = 1:height
    for j = 1:width
        % compute M
        M = [
            Ix2(i,j), Ixy(i,j);
            Ixy(i,j) Iy2(i,j)
            ];
        % compute corner reponse function, alpha = 0.04
        R(i,j) = det(M)-0.04*(trace(M))^2;
        if R(i,j) > Rmax
           Rmax = R(i,j);
        end;

    end;
end;
cnt = 0;
Rmax
% determine corner by comparing with threshold and all neighbor points
for i = 2:height-1
    for j = 2:width-1
        M = [
                 Ix2(i,j), Ixy(i,j);
```

```
                 Ixy(i,j) Iy2(i,j)
            ];
        % compute M's eigenvalue
        [vector,lambda] = eig(M);
        lambda = diag(lambda);
        % compute radius associate to the ellipse
        [lambda1,index] = max(lambda);
        lambda2 = min(lambda);
        a = lambda1^(-0.5);
        b = lambda2^(-0.5);
        vector = vector(:,index);
        angle = atan(vector(2)/vector(1));
        if mod(i,4) + mod(j,4) == 0
            ellipsePerem = [ellipsePerem;a b angle j i];
        end
        if R(i,j) > 0.1*Rmax &&...
            R(i,j) > R(i-1,j-1) ...
            && R(i,j) > R(i-1,j) ...
            && R(i,j) > R(i-1,j+1) ...
            && R(i,j) > R(i,j-1) ...
            && R(i,j) > R(i,j+1) ...
            && R(i,j) > R(i+1,j-1) ...
            && R(i,j) > R(i+1,j) ...
            && R(i,j) > R(i+1,j+1)

            result(i,j) = 1;
            cnt = cnt+1;
           % if (lambda1 - lambda2 <0.1)
            %end
        end;
    end;
end;
[posc, posr] = find(result == 1);
cnt ;
%% ========== Part 2: Plot ellipse and corner on image ====================

% ellipse(ellipsePerem(:,1), ellipsePerem(:,2), ellipsePerem(:,3), ...
%     ellipsePerem(:,4),ellipsePerem(:,5));
plot(posr,posc,'r.');
```

```
function h=ellipse(ra,rb,ang,x0,y0,C,Nb)
% Ellipse adds ellipses to the current plot
%
% ELLIPSE(ra,rb,ang,x0,y0) adds an ellipse with semimajor axis of ra,
% a semimajor axis of radius rb, a semimajor axis of ang, centered at
% the point x0,y0.
%
% The length of ra, rb, and ang should be the same.
% If ra is a vector of length L and x0,y0 scalars, L ellipses
% are added at point x0,y0.
% If ra is a scalar and x0,y0 vectors of length M, M ellipse are with the same
% radii are added at the points x0,y0.
% If ra, x0, y0 are vectors of the same length L=M, M ellipses are added.
% If ra is a vector of length L and x0, y0 are  vectors of length
% M≠L, L*M ellipses are added, at each point x0,y0, L ellipses of radius ra.
%
% ELLIPSE(ra,rb,ang,x0,y0,C)
% adds ellipses of color C. C may be a string ('r','b',...) or the RGB value.
% If no color is specified, it makes automatic use of the colors specified by
% the axes ColorOrder property. For several circles C may be a vector.
%
% ELLIPSE(ra,rb,ang,x0,y0,C,Nb), Nb specifies the number of points
% used to draw the ellipse. The default value is 300. Nb may be used
```

```
% for each ellipse individually.
%
% h=ELLIPSE(...) returns the handles to the ellipses.
%
% as a sample of how ellipse works, the following produces a red ellipse
% tipped up at a 45 deg axis from the x axis
% ellipse(1,2,pi/8,1,1,'r')
%
% note that if ra=rb, ELLIPSE plots a circle
%

% written by D.G. Long, Brigham Young University, based on the
% CIRCLES.m original
% written by Peter Blattner, Institute of Microtechnology, University of
% Neuchatel, Switzerland, blattner@imt.unine.ch


% Check the number of input arguments

if nargin<1,
  ra=[];
end;
if nargin<2,
  rb=[];
end;
if nargin<3,
  ang=[];
end;

%if nargin==1,
%  error('Not enough arguments');
%end;

if nargin<5,
  x0=[];
  y0=[];
end;

if nargin<6,
  C=[];
end

if nargin<7,
  Nb=[];
end

% set up the default values

if isempty(ra),ra=1;end;
if isempty(rb),rb=1;end;
if isempty(ang),ang=0;end;
if isempty(x0),x0=0;end;
if isempty(y0),y0=0;end;
if isempty(Nb),Nb=300;end;
if isempty(C),C=get(gca,'colororder');end;

% work on the variable sizes

x0=x0(:);
y0=y0(:);
ra=ra(:);
rb=rb(:);
ang=ang(:);
Nb=Nb(:);

```

```
if isstr(C),C=C(:);end;

if length(ra)≠length(rb),
  error('length(ra)≠length(rb)');
end;
if length(x0)≠length(y0),
  error('length(x0)≠length(y0)');
end;

% how many inscribed elllipses are plotted

if length(ra)≠length(x0)
  maxk=length(ra)*length(x0);
else
  maxk=length(ra);
end;

% drawing loop

for k=1:maxk

  if length(x0)==1
    xpos=x0;
    ypos=y0;
    radm=ra(k);
    radn=rb(k);
    if length(ang)==1
      an=ang;
    else
      an=ang(k);
    end;
  elseif length(ra)==1
    xpos=x0(k);
    ypos=y0(k);
    radm=ra;
    radn=rb;
    an=ang;
  elseif length(x0)==length(ra)
    xpos=x0(k);
    ypos=y0(k);
    radm=ra(k);
    radn=rb(k);
    an=ang(k);
  else
    rada=ra(fix((k-1)/size(x0,1))+1);
    radb=rb(fix((k-1)/size(x0,1))+1);
    an=ang(fix((k-1)/size(x0,1))+1);
    xpos=x0(rem(k-1,size(x0,1))+1);
    ypos=y0(rem(k-1,size(y0,1))+1);
  end;

  co=cos(an);
  si=sin(an);
  the=linspace(0,2*pi,Nb(rem(k-1,size(Nb,1))+1,:)+1);
%  x=radm*cos(the)*co-si*radn*sin(the)+xpos;
%  y=radm*cos(the)*si+co*radn*sin(the)+ypos;
  h(k)=line(radm*cos(the)*co-si*radn*sin(the)+xpos,radm*cos(the)*si+co*radn*sin(the)+ypos);

end;
```

## (b)

The Gaussian window size I chose are $w = 3$, $w = 4$, $w = 7$ and $w = 10$, the results for image $I1$ and $I3$ are listed below. From results shown in Fig. 1$\sim$ 4 and Fig. 5$\sim$ 8, we find that the ellipses are easily distinguished and the corner detection is highly effective. However, as we increase the window size, the ellipses gradually become hard to be distinguished from its environment, the effects of detector become worse as window size increases.

Figure 1: Figure I1 with window size $w = 3$

Figure 2: Figure I1 with window size $w = 4$

Figure 3: Figure I1 with window size $w = 7$

Figure 4: Figure I1 with window size $w = 10$

Figure 5: Figure I3 with window size $w = 3$

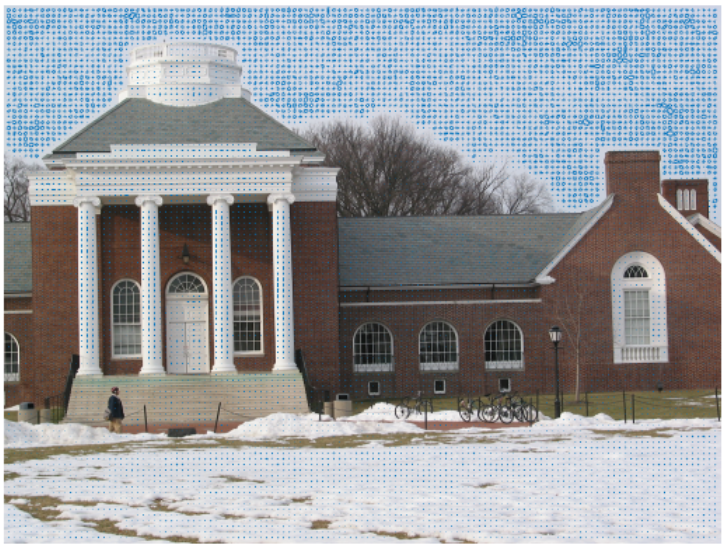

Figure 6: Figure I3 with window size $w = 4$

Figure 7: Figure I3 with window size $w = 7$

Figure 8: Figure I3 with window size $w = 10$

## (c)

The thresholds are chosen as $T = tR_{max}$ , where $R_{max}$ is defined as the maxinum corner response and takes value of $9.4191 \times 10^9$ for $I1.png$, and $1.0751 \times 10^{10}$ for $I3.png$. Take $t = 0.001$, 0.01, 0.05 and 0.1. We show results for different t in Fig. 9 $\sim$ 16. From the figure, we can see that while $T = 0.01R_{max}$ captures a reasonable result, $T = 0.001R_{max}$ results including a lot of edge points besides corner points, indicating this threshold is too low. Also, $T = 0.1R_{max}$ and $T = 0.05R_{max}$ tends to lose some corner points, which is more apparently in the former case, indicating the threshold is too high.

Figure 9: Harris Corner Detection with threshold $t = 0.001$

Figure 10: Harris Corner Detection with threshold $t = 0.01$

Figure 11: Harris Corner Detection with threshold $t = 0.05$

Figure 12: Harris Corner Detection with threshold $t = 0.1$

Figure 13: Harris Corner Detection with threshold $t = 0.001$

Figure 14: Harris Corner Detection with threshold $t = 0.01$

Figure 15: Harris Corner Detection with threshold $t = 0.05$

Figure 16: Harris Corner Detection with threshold $t = 0.1$

# Problem 2 Blob Detection

## (a)

The threshold is defined as $T = t \cdot R_{max}$, $R_{max}$ is the maximum value of scale space representation and $R_{max} = 1.1669 \times 10^4$. When we take values of $t$ as 0.1, 0.2, 0.3 and 0.4, we will get results shown in the following figures. From Fig. 17∼20, it can be deduced that as we increase the threshold for Blob detection, the number of blob features we can detect will decrease. This is reasonable since a high threshold will neglect some less important features in the figure. The Matlab codes and detailed steps are attached behind.

Figure 17: Blob Detection with thresholds $t = 0.1$

Figure 18: Blob Detection with thresholds $t = 0.2$

Figure 19: Blob Detection with thresholds $t = 0.3$

Figure 20: Blob Detection with thresholds $t = 0.4$

## (b)

The histogram is shown in Fig. 21.

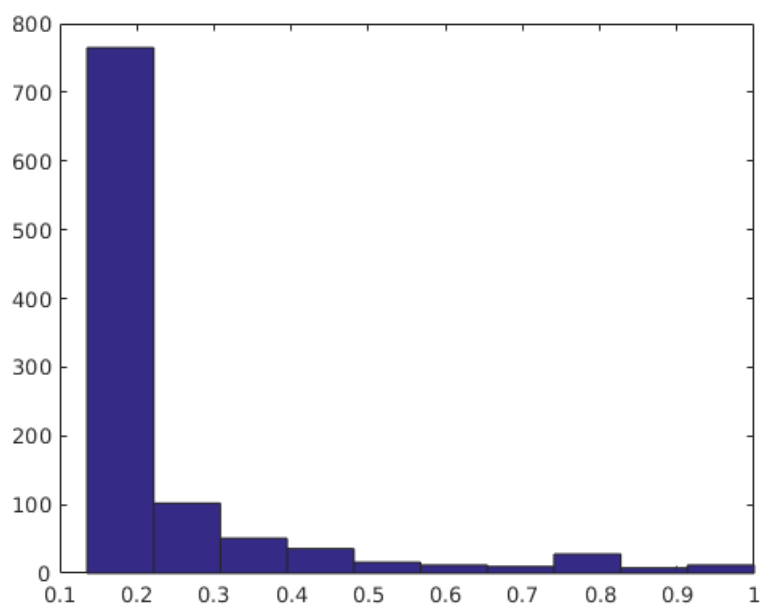


Figure 21: Relations between number of blobs and normalized blobs' radii in Blob Detection with thresholds $T = 1500$

```matlab
%% EECS 442 - HW 04 - Q2 Blob Detection
%  Date: 11/26/2016
%  by Yi Yang

%  used helper function
%  --------------------
%  show_all_circles(I, cx, cy, rad, color, ln_wid)

%  Instructions
%  ------------
%  Implement a blob detector based on the Laplacian operator and
%  characteristic scale concept introduced by Lindeberg. Compute the radius
%  of each blob (in pixels) and report the histogram (dis- tribution)
%  of radii computed for the whole image.

%% Initialization
clc; clear; close all;
%% =============== Part 0: Parameter selection ============================
t = 1:0.1:3;
sigma_scale = exp(t);
sigma_scale = sigma_scale';
%levels = 10;
levels = length(sigma_scale);
%% ======================== Part 1: Load data ============================
img = imread('homework4/3.bmp');
img = rgb2gray(img);
img = img;

dimg = double(img);

% Scale space creation:
[height width] = size(img);
scale_space = zeros(height,width,levels);

%% ================ Part 2: Perform detection ============================

for i=1:levels

% Filter creation:
filter_size = 2*ceil(3*sigma_scale(i))+1;
kernel = fspecial('log', filter_size, sigma_scale(i));

nLoG = sigma_scale(i)^2 * kernel;
imFilt = imfilter(dimg, nLoG, 'same', 'replicate');
imFilt = imFilt .^ 2;
scale_space(:,:,i) = imFilt;
end

% Perform non-maximum suppression in the 3D scale space
% used vl_localmax in vlfeat for detection
% thres = 0.4 * max(max(max(scale_space)));
thres = 1500;
max(max(max(scale_space)))

scale_space_filtered = zeros(height,width,levels);

for i=1:levels
    scale_space_filtered(:,:,i) = ordfilt2(scale_space(:,:,i),9,ones(3));
end
%non maximum supression between scales

for i = 1:levels
    scale_space_filtered(:,:,i) = ...
        max(scale_space_filtered(:,:,max(i-1,1):min(i+1,levels)),[],3);
```

```
end
scale_space_filtered...
    = scale_space_filtered .* (scale_space_filtered == scale_space);


r = [];
c = [];
rad = [];
for i=1:levels
    [rows,cols] = find(scale_space_filtered(:,:,i)≥thres);
    num = length(rows);
    radii = sigma_scale(i)*sqrt(2);
    radii = repmat(radii,num,1);
    r = [r;rows];
    c = [c;cols];
    rad = [rad;radii];
end
show_all_circles(img,c,r,rad);
title(sprintf('%d circles, threshold %f', size(c,1), thres));


% recompute 3d-indices of maxima
s = [height width levels];
%% ================ Part 3: draw blobs on image ==========================
figure
rad = rad / max(rad);

hist(rad,10);
```