TEACHETH MAN THAT WHICH HE KNEW NOT

IUB

INDEPENDENT UNIVERSITY

ASSIGNMENT-06

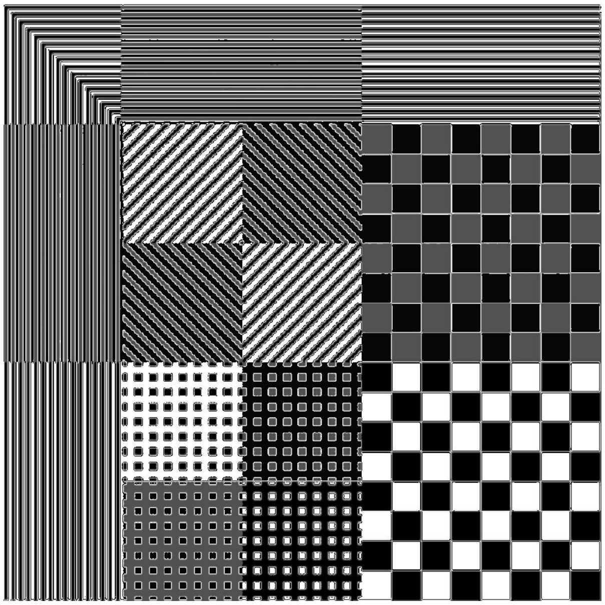
Course Title: Image Processing
Course ID: CSC 420
Submitted By: Abu Bakar Siddik Nayem.
ID: 1510190.
Section: 01.
Date of Submission: 24th November, 2018.
Submitted To: Md. Ashraful Amin, PhD.

1. **Find edge from the given image.**

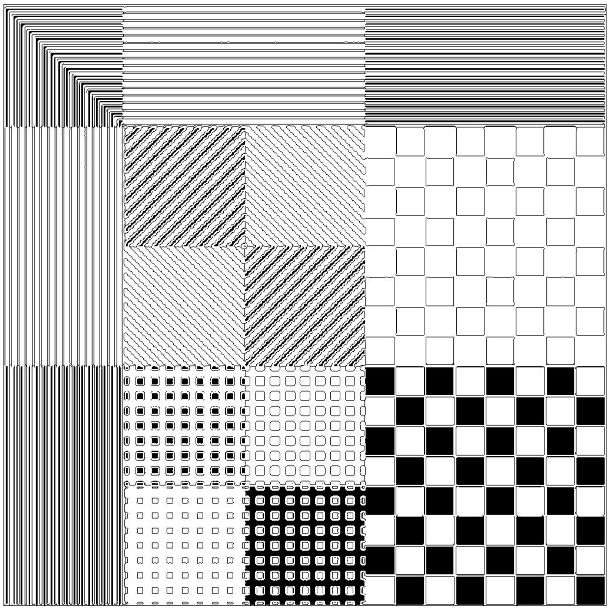
<u>Ans:</u>

There are many edge detection methods available in image processing.

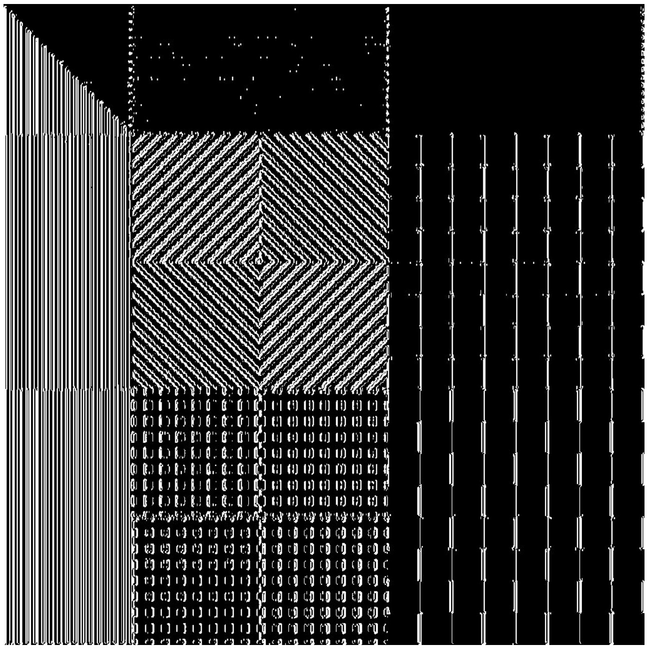
The input image is:



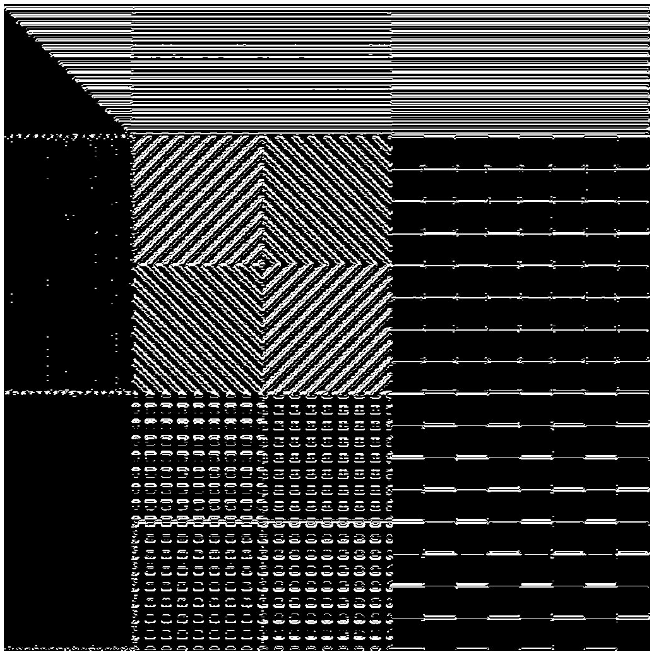
Using the laplacian filter we can get the edges below:



Using right sobel filter:



Using bottom sobel:



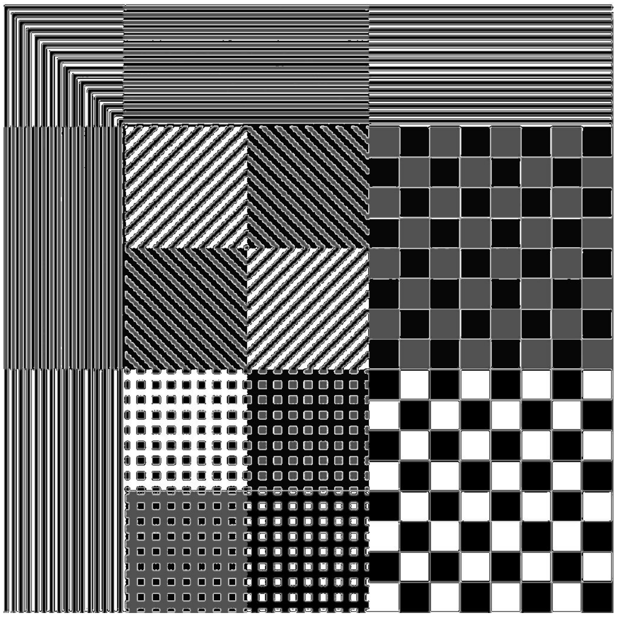
Observation:

The laplacian filter seems to detect the edges better than most other filters.

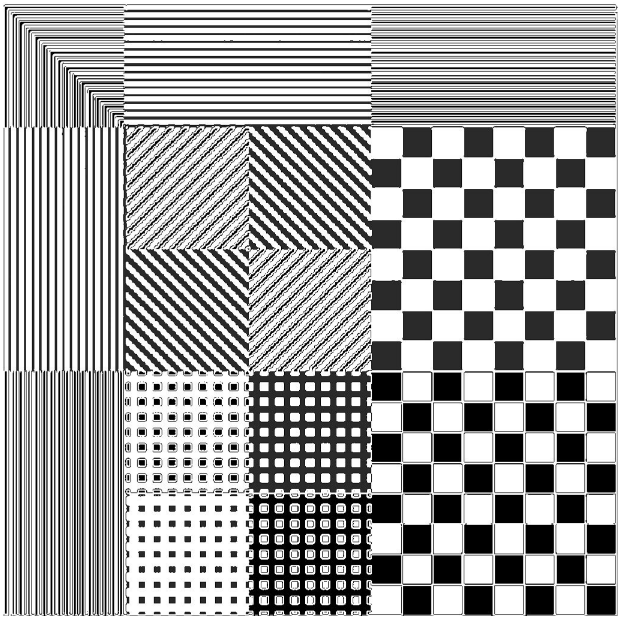
2. Apply edge enhancement techniques to the given image.

Ans:

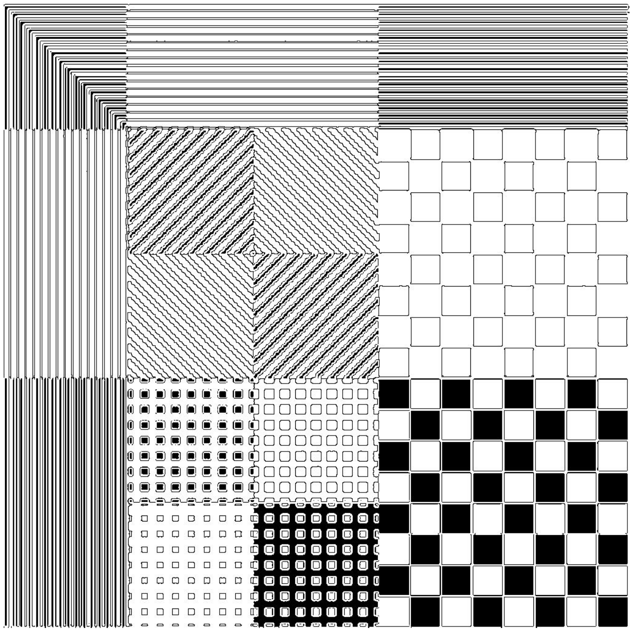
Input image:



Adding the original image with the laplacian filtered edges:



After applying sharpen filter:

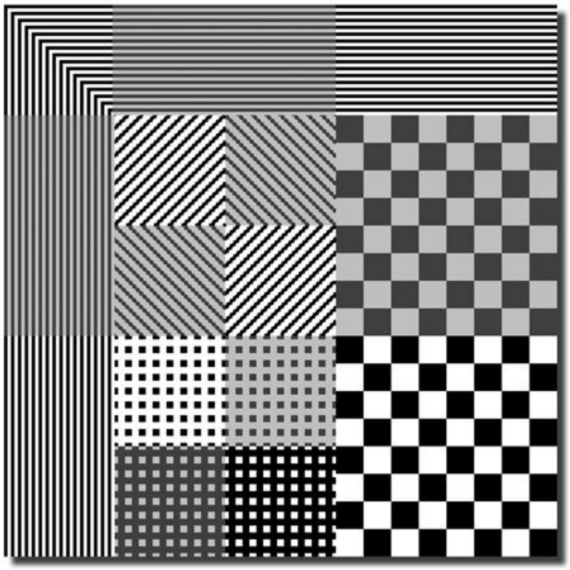


Sharpening an image boosts/enhances the edges.

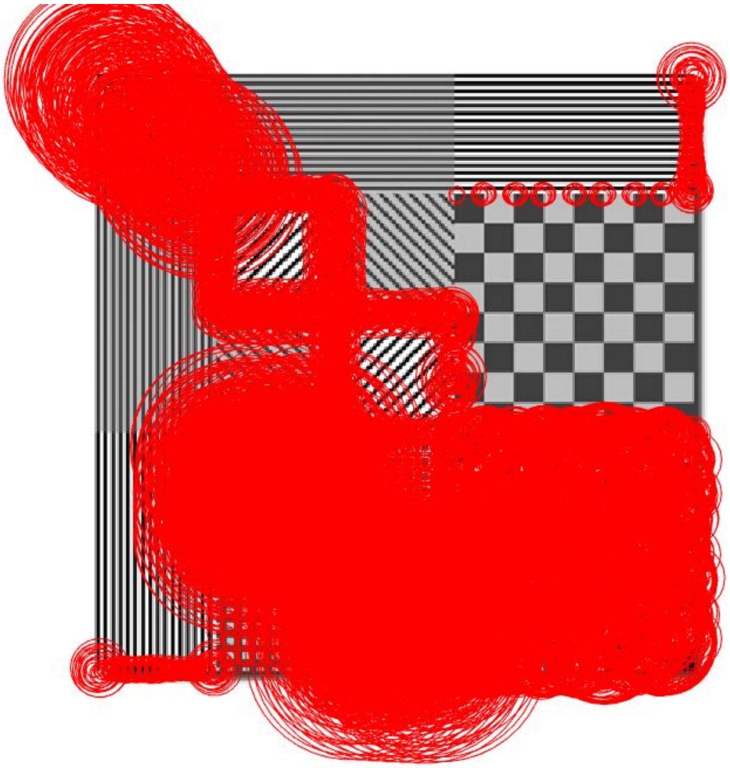
3. Detect corner from the given image.

Ans:

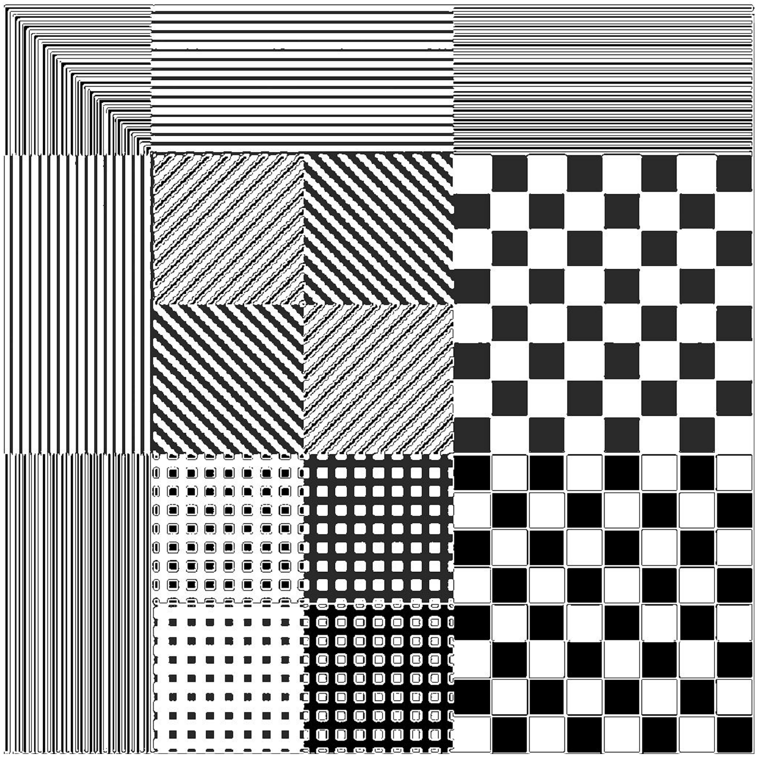
Input image:



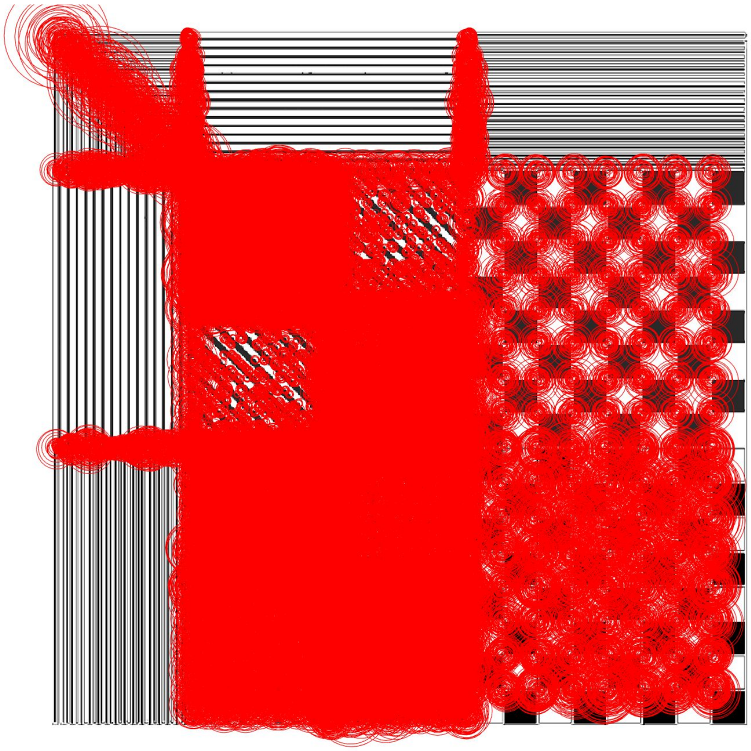
Using harris corner detection:



Using a sharpened(laplacian edge) version of the board image as input:



Then applying harris cornet detection:



Observation:

From the above experiment we can see that the corner detection method works best when the input image has sharp/enhanced edges.

Appendix:

1) laplacian.m

```
clc
clear all
close all
image=imread('Board.bmp');
filter=[0 -1 0 ; -1 9 -1 ; 0 -1 0];
rows = size(image,1);
cols = size(image,2);
outputimage = zeros(rows,cols);

for row = 2:rows-1
   for col = 2:cols-1
        outputimage (row,col)= sum (sum(double(image(row-1: row+1, col-1: col+1 )) .*
filter ));
   end
end
image =uint8(image);
figure,imshow (outputimage);
imwrite(outputimage, 'Edges.png')
```

AppltyBottomSobelFIlterToImage.m

```
function [ outputImage ] = applyBottomSobelFilterToImage( imageFilePath )
   bottomSobelFilter = [-1 -2 -1; 0 0 0; 1 2 1];
   outputImage = applyImageFilter(imread(imageFilePath), bottomSobelFilter);
end
```

ApplyRIghtSObelFIlterToImage:

```
function [ outputImage ] = applyRightSobelFilterToImage( imageFilePath )
   rightSobelFilter = [-1 0 1; -2 0 2; -1 0 1];
   outputImage = applyImageFilter(imread(imageFilePath), rightSobelFilter);
end
```

AddPAdding.m:

```
function [ paddedInputMatrix ] = addPadding( inputMatrix, numOfLayersOfPadding )
% Pads an input matrix in both the vertical and horizontal directions
%   Values of the padded rows/columns are copied from the edges of the input matrix
%   That is, the resulting matrix will look like a stretched version of the input
   paddedInputMatrix = padarray(inputMatrix, [numOfLayersOfPadding
numOfLayersOfPadding], 'replicate');
end
```

ApplyImageFilter:

```
function [ outputImage ] = applyImageFilter( inputImage, filter )
% Applies a filter to a given input image and outputs the resulting filtered image

   numOfPaddingLayers = calcNumOfPaddingLayersGivenFilter(filter);
   paddedInputImage = addPadding(inputImage, numOfPaddingLayers);

   outputImage = zeros(size(inputImage));

   for i=1:size(inputImage, 1)
      for j=1:size(inputImage, 2)
         inputImageSegment = paddedInputImage(i:i + size(filter, 1) - 1, j: j + size(filter, 2) - 1);
         outputImage(i, j) = applyFilterToCentralPixel(inputImageSegment, filter);
      end
   end
end
```

SampleUsage:

```
originalImageFilePath = './images/Board.bmp';

originalImage = imread(originalImageFilePath);
figure(1)
imshow(originalImage, [0 255]);
title('Original');

bottomSobelFilteredImage = applyBottomSobelFilterToImage(originalImageFilePath);
figure(3)
imshow(bottomSobelFilteredImage, [0 255]);
title('Bottom Sobel');
imwrite(bottomSobelFilteredImage,'bottom_sobel.png');
```

```
rightSobelFilteredImage = applyRightSobelFilterToImage(originalImageFilePath);
figure(7)
imshow(rightSobelFilteredImage, [0 255]);
title('Right Sobel');
imwrite(rightSobelFilteredImage,'right_sobel.png');
```

2.

Laplacian.m

```
clc
clear all
close all
image=imread('Board.bmp');
filter=[0 -1 0 ; -1 9 -1 ; 0 -1 0];
rows = size(image,1);
cols = size(image,2);
outputimage = zeros(rows,cols);

for row = 2:rows-1
   for col = 2:cols-1
       outputimage (row,col)= sum (sum(double(image(row-1: row+1, col-1: col+1 )) .* filter ));
   end
end
image =uint8(image);
figure,imshow (outputimage);
imwrite(outputimage, 'Edges.png')
imUint = uint8(outputimage);
imEn = imadd(image,imUint);
figure,imshow (imEn);
imwrite(imEn,'Edge_enhanced.png');
```

ApplySharpenFolterToImage:

```
function [ outputImage ] = applySharpenFilterToImage( imageFilePath )
   sharpenFilter = [0 0 0; 0 2 0; 0 0 0] - (1/9)*ones(3);
   outputImage = applyImageFilter(imread(imageFilePath), sharpenFilter);
end
```

AddPAdding.m:

```
function [ paddedInputMatrix ] = addPadding( inputMatrix, numOfLayersOfPadding )
% Pads an input matrix in both the vertical and horizontal directions
%   Values of the padded rows/columns are copied from the edges of the input matrix
%   That is, the resulting matrix will look like a stretched version of the input
   paddedInputMatrix = padarray(inputMatrix, [numOfLayersOfPadding
numOfLayersOfPadding], 'replicate');
end
```

ApplyImageFilter:

```
function [ outputImage ] = applyImageFilter( inputImage, filter )
% Applies a filter to a given input image and outputs the resulting filtered image

   numOfPaddingLayers = calcNumOfPaddingLayersGivenFilter(filter);
   paddedInputImage = addPadding(inputImage, numOfPaddingLayers);

   outputImage = zeros(size(inputImage));

   for i=1:size(inputImage, 1)
      for j=1:size(inputImage, 2)
         inputImageSegment = paddedInputImage(i:i + size(filter, 1) - 1, j: j + size(filter, 2) - 1);
         outputImage(i, j) = applyFilterToCentralPixel(inputImageSegment, filter);
      end
   end
end
```

SampleUsage:

```
originalImageFilePath = './images/Board.bmp';

sharpenenedImage = applySharpenFilterToImage(originalImageFilePath);
figure(8)
imshow(sharpenenedImage, [0 255]);
title('Sharpen');
imwrite(sharpenenedImage,'sharpened.png');
```

3)

ExtractKeypoints:

```
function [ x, y, scores, Ix, Iy ] = extract_keypoints( image )

% convert to double
G2 = im2double(image);

% create X and Y Sobel filters
horizontal_filter = [1 0 -1; 2 0 -2; 1 0 -1];
vertical_filter = [1 2 1; 0 0 0 ; -1 -2 -1];

% using imfilter to get our gradient in each direction
filtered_x = imfilter(G2, horizontal_filter);
filtered_y = imfilter(G2, vertical_filter);

% store the values in our output variables, for clarity
Ix = filtered_x;
Iy = filtered_y;

f = fspecial('gaussian');
Ix2 = imfilter(Ix.^2, f);
Iy2 = imfilter(Iy.^2, f);
Ixy = imfilter(Ix.*Iy, f);

% set empirical constant between 0.04-0.06
k = 0.04;

num_rows = size(image,1);
num_cols = size(image,2);

% create a matrix to hold the Harris values
H = zeros(num_rows, num_cols);

% % get our matrix M for each pixel
for y = 6:size(image,1)-6
    for x = 6:size(image,2)-6
        Ix2_matrix = Ix2(y-2:y+2,x-2:x+2);
        Ix2_mean = mean(Ix2_matrix(:));
        % Iy2 mean
        Iy2_matrix = Iy2(y-2:y+2,x-2:x+2);
        Iy2_mean = mean(Iy2_matrix(:));

        % Ixy mean
        Ixy_matrix = Ixy(y-2:y+2,x-2:x+2);
        Ixy_mean = mean(Ixy_matrix(:));

        Matrix = [Ix2_mean, Ixy_mean;
                  Ixy_mean, Iy2_mean];
        R1 = det(Matrix) - (k * trace(Matrix)^2);


        % store the R values in our Harris Matrix

        % H(y,x) = R;

        H(y,x) = R1;

    end
end

% set threshold of 'cornerness' to 5 times average R score
avg_r = mean(mean(H))
threshold = abs(5 * avg_r)
[row, col] = find(H > threshold);
scores = [];
%get all the values
for index = 1:size(row,1)
    %see what the values are
    r = row(index);
    c = col(index);
    %store the scores
    %score(index) = H(r,c);
    scores = cat(2, scores,H(r,c));
end
y = row;
x = col;

end
```

Compute_features:

```
function [ features, x, y, scores ] = compute_features( x, y, scores, Ix, Iy )

grad_mag = zeros(size(Ix));
grad_orient = zeros(size(Ix));

% size of score
features = [];

for i = 1:size(Iy, 1)
    for j = 1:size(Ix, 2)
        % fill the magnitude matrix
        grad_mag(i, j) = sqrt(Ix(i,j)^2 + Iy(i,j)^2);

        % find the orient
        orient_raw = atand(Iy(i,j) / Ix(i,j));

        if (isnan(orient_raw))
            assert(grad_mag(i,j) == 0);
            orient_raw = 0; % if no change, we won't count a gradient magnitude
        end

        % get the correct bin
        assert(orient_raw >= -90);
        if (orient_raw <= -67.5)
            grad_orient(i,j) = 1;
        elseif (orient_raw <= -45)
            grad_orient(i,j) = 2;
        elseif (orient_raw <= 22.5)
            grad_orient(i,j) = 3;
        elseif (orient_raw <= 0)
            grad_orient(i,j) = 4;
        elseif (orient_raw <= 22.5)
            grad_orient(i,j) = 5;
        elseif (orient_raw <= 45)
            grad_orient(i,j) = 6;
        elseif (orient_raw <= 67.5)
            grad_orient(i,j) = 7;
        else % orient_raw > 67.5
            grad_orient(i,j) = 8;
        end
    end
end


% find 8-d descriptor for the 11x11 grid aroud each feature
% get values for each feature
for index = 1:size(scores,2)
    % save the descriptor
    descriptor = zeros(8,1);

    % get indices
    row = y(index, 1);
    col = x(index, 1);

    % check if feature is < 5px from top-left/bottom-right, erase if so
     if row <= 5 || col <= 5 || (size(grad_mag,1) - row) <= 5 || (size(grad_mag,2) - col) <= 5
         % then skip, we just don't add the feature
         continue
     end

    % assert that we don't have row or col values that will be -1 or lower
    assert(row-5 >= 1, 'Row is < 1!');
    assert(col-5 >= 1, 'Col is < 1!');

    % iterate through the 11x11 area AROUND the feature
    for y_val = row-5:row+5
        for x_val = col-5:col+5
            mag = grad_mag(y_val,x_val);
            orient = grad_orient(y_val, x_val);

            % store that orient/mag
            descriptor(orient, 1) = descriptor(orient, 1) + mag;
        end
    end

    % cat this onto the features vector
    features = cat(2, features, descriptor);
end

% normalize the vectors
for index = 1:size(features,2)
    f = features(:,index);
    f_normal = f(:)/sum(f);
    features(:, index) = f_normal;
end

% clip values to 0.2
features( features > 0.2 ) = 0.2;
% and normalize the vectors again, with this new value set
for index = 1:size(f_normal,2)
    f = features(:,index);
    f_normal = f(:)/sum(f);
    features(:, index) = f_normal;
end

end
```

Show_Keypoints.m:

```
image = imread('edge_enhanced.png');
image = imresize(image, 0.75);
[ x, y, scores, Ix, Iy ] = extract_keypoints( image );
figure; imshow(image)
hold on
for i = 1:size(scores,2)
    %plot(x(i), y(i), 'ro', 'MarkerSize', scores(i) / 1000000000);
    plot(x(i), y(i), 'ro', 'MarkerSize', scores(i) * 5);

end
saveas(gcf,'assignment_06_corners2.png');
hold off

% get the feature descriptors via SIFT
[ pgh_features, x, y, scores ] = compute_features( x, y, scores, Ix, Iy );
```