Krikor Hovasapian
ECG795 HW#2
3/19/2013

1.

a) Normalized Cross Correlation Detection function

```
function [ list ] = corr_detect( template, timg, threshold )
%LIST Returns a list of image locations that are > than threshold
%   Detailed explanation goes here

%Step 1: Perform normalized correlation
g = normxcorr2(template, timg);

%Step 2: Look for locations that are > threshold and return
list = zeros(size(g,1), size(g,2));
for row=1:size(g,1)
    for col=1:size(g,2)
        if(g(row,col) > threshold)
            list(row, col) = g(row,col);
        end
    end
end

end
```

b) Tried following the VOC documentation for computing the overlap, but had some issues trying to get the ROC plotted.

```
function [ roc ] = problem1( template, threshold, total_imgs, truedatafile )
%ROC Summary of this function goes here
%   Detailed explanation goes here

%Create a total_imgs x 2 table for ROC consisting of TP, FP
outcome_table = zeros(total_imgs, 2);
bounding_box = zeros(1,4);
previous_tp = 0;
previous_fp = 0;
rate = 0;

%Read in and parse the true data set
truetable = cardata_parser(truedatafile);

for img = 0:total_imgs
    %Grab image
    imgnum = int2str(img);
    path = strcat('CarData/TestImages/test-',imgnum,'.pgm');
    f = imread(path);

    %Run correlation and determine bounds
    list = corr_detect(template,f,threshold);
    label = bwlabel(list);
    s = regionprops(label,'Area','BoundingBox');

    %Grab true coordinate for image x
    index = find(truetable(:,3) == img+1, 3);
    truecoord = truetable(index,[1,2]);
```

```
    %Create bounding box (position vector) for true location
    bounding_box(1) = truecoord(1);
    bounding_box(2) = truecoord(2);
    bounding_box(3) = 100;
    bounding_box(4) = 40;


    %Compute ROC components
    if(~isempty(s))
        for i=1:size(s,1)

            area1 = sum(intersect(bounding_box,s(i).BoundingBox));
            area2 = sum(union(bounding_box, s(i).BoundingBox));
            rate = area1 / area2;

            %fishing for TP's :)
            if(rate > 0.5)
                break
            end

        end
    end

    if(rate > 0.5)
        %TP
        outcome_table(img+1,:) = [rate previous_fp];
        previous_tp = rate;
    else
        %FP
        outcome_table(img+1,:) = [previous_tp rate];
        previous_fp = rate;
    end


    %Run NMS on list (part c)


end

 imshow(f)
 hold on
 axis equal
 if(~isempty(s))
    for i=1:size(s,1)
        rectangle('Position', s(i).BoundingBox, 'EdgeColor', 'r');
    end
 end
 rectangle('Position', bounding_box, 'EdgeColor', 'g');
 hold off

roc = outcome_table;
end
```

cardata_parser.m

```
function [ table ] = cardata_parser( datafile )
%TABLE Parses out trueLocations.txt file only and returns a matrix
%   Each line is read from the datafile and parsed out for
%   coordinates. Each coordinate has a tag that denotes which
%   image that coordinate belongs to. Images with multiple
%   coordinates have the same tag on multiple rows. The tag
%   is the third column in the matrix.

%Step 1: Open the file with read access
```

```
fileID = fopen(datafile, 'r');
truetable = [];
row = 1;
while(feof(fileID) ~= 1)

    %Step 2: Grab a line from the datafile
    %         and determine how many coordinates there are
    %         by looking at commas
    str = fgetl(fileID);
    commas = strfind(str,',');

    %Step 3: Pre Tokenize, if only one comma we're done...
    [token, remain] = strtok(str);
    if(size(commas(:),1) ==  1)
        xy = textscan(remain, '(%d16,%d16)');
        xy = horzcat(xy, row);
        truetable = vertcat(truetable,xy);
    else
        %Step 4: Otherwise... keep breaking into tokens
        for i = 1:size(commas(:),1)
            [token, remain] = strtok(remain);
            xy = textscan(token, '(%d16,%d16)');
            xy = horzcat(xy, row);
            truetable = vertcat(truetable,xy);
        end
    end
    row = row + 1;
end

fclose(fileID);

%Step 5: Convert your cell to a matrix.
data = cellfun(@int16,truetable,'UniformOutput',false);
table = cell2mat(data);

%Example: How to find an image coordinates:
% >> index = find(table(:,3) == 21, 3);
%
% Here, we're looking in the third column for image 21,
% which for the trueLocations.txt file returns index 34
% in our new matrix table above.

% >> table(index,[1,2]);
%
% Once you know the index, you can extract the coordinates
% and do what you want with the values.
end
```

Krikor Hovasapian
ECG795 HW#2
3/19/2013

2. Corner Detection

a) Show that the eigenvalues of A are given by:

$$\lambda = \frac{tr(A) \pm \sqrt{tr(A)^2 - 4det(A)}}{2}$$

Given:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \textbf{(2 x 2 matrix)}$$

$$det(\lambda I - A) = 0 \qquad \textbf{(Characteristic Equation)}$$

Show:

$$\lambda = \frac{tr(A) \pm \sqrt{tr(A)^2 - 4det(A)}}{2}$$

We know:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \textbf{(Identity Matrix)}$$

Therefore:

$$det(\lambda I - A) = 0$$
$$det\left(\begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} - \begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = 0$$
$$det\left(\begin{bmatrix} \lambda - a & -b \\ -c & \lambda - d \end{bmatrix}\right) = 0$$

$$(\lambda - a)(\lambda - d) - bc = 0$$
$$\lambda^2 - \lambda(d + a) + (ad - bc) = 0$$

Using Quadratic Equation, we get:

$a = 1$
$b = (d + a) = tr(A)$
$c = |A| = (ad - bc)$

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\lambda = \frac{-tr(A) \pm \sqrt{tr(A)^2 - 4det(A)}}{2}$$

b)

problem2.m

```
function [ f ] = problem2( image )
%F Returns image with keypoints denoting corners found
%   Detailed explanation goes here

%Find corners
ret = corner_detector(image);

%Plot keypoints
imshow(image);       %# Display your image
hold on;             %# Add subsequent plots to the image

for i=1:size(ret,1)
    x = ret(i,1);
    y = ret(i,2);
    plot(y,x,'o');  %# NOTE: x_p and y_p are switched (see note below)!
end
hold off;            %# Any subsequent plotting will overwrite the image!

f = image;
end
```

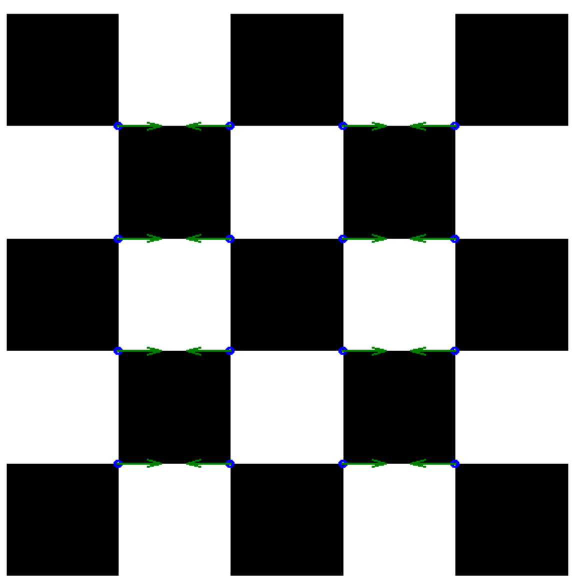
corner_detector.m

```
function [ s ] = corner_detector( image )
%X Finds the minimum eigenvalues greater than a certain threshold
%   Detailed explanation goes here

% Step 1: Read in image
%I = imread(image);
%f = im2double(I(:,:,1));
f = imread(image);
ROWS = size(f,1);
COLS = size(f,2);
lammax = zeros(ROWS,COLS);
lammin = zeros(ROWS,COLS);
angle = zeros(ROWS,COLS);

% Step 2: Compute the Image Gradient of image
[Ix Iy] = imgrad(f);
Ix2 = Ix .* Ix;
Iy2 = Iy .* Iy;
IxIy = Ix .* Iy;

% Step 3: Filter gradients with weight
w = [1 1 1; 1 1 1; 1 1 1];
a = double(imfilter(Ix2, w));
b = double(imfilter(IxIy,w));
c = double(imfilter(Iy2, w));

% Step 4: Compute Autocorrelation Matrix and Lambdas (eig)
for i=1:ROWS
    for j=1:COLS
        A = [a(i,j) b(i,j); b(i,j) c(i,j)];
        lambda = eig(A);
        lammin(i,j) = min(lambda);
        lammax(i,j) = max(lambda);
```

```
        angle(i,j) = 0.5.*atan((2.*b(i,j))/(a(i,j) - c(i,j)));
    end
end

% Step 5: Look for lammin's greater than max(lammin) * 80%
threshold = (max(lammin(:)) .* 0.80);
[rows, cols] = find(lammin > threshold);

% Step 6: Plot keypoints and vector
imshow(image);
hold on;

u = zeros(size(rows,1),1);
v = zeros(size(rows,1),1);
scale = zeros(size(rows, 1),1);
for i=1:size(rows,1)
    x = rows(i);
    y = cols(i);
    u(i) = Ix(x);
    v(i) = Iy(y);
    scale(i) = lammin(x,y);
    plot(y,x,'o');
end

quiver(cols,rows,v,u);



hold off;

% Step 7: Package and return
s = struct('Ix',Ix, 'Iy',
Iy,'LambdaMin',lammin,'LambdaMax',lammax,'Angle',angle,'Threshold',threshold','
KeypointsRows',rows,'KeypointsCols',cols);

end
```

Results:

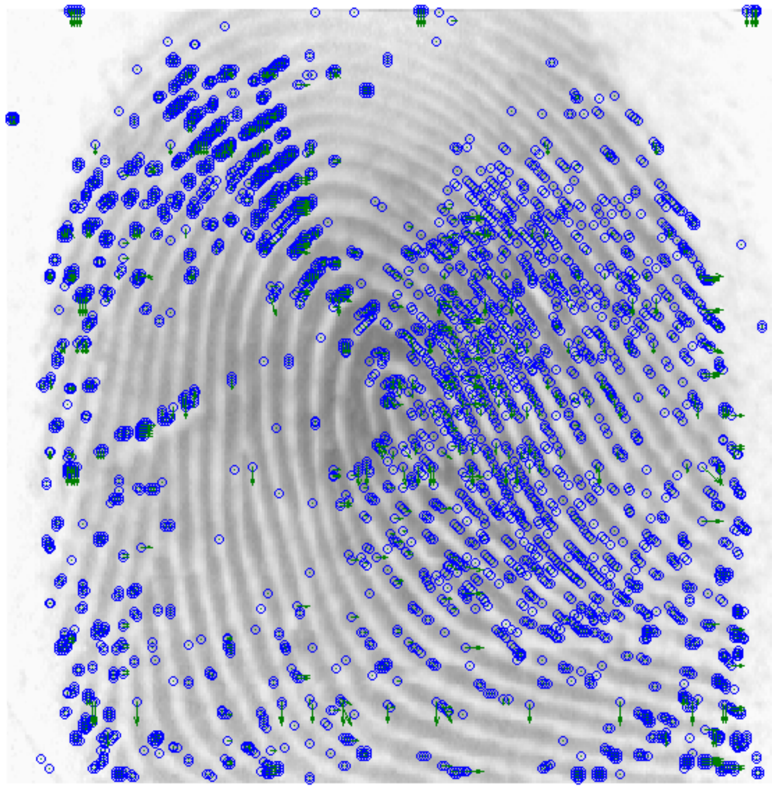
Checkerboard:



Krikor Hovasapian
ECG795 HW#2
3/19/2013

c)

Fingerprint with vectors:



Krikor Hovasapian
ECG795 HW#2
3/19/2013

3.

a) Keypoints in both graffiti images

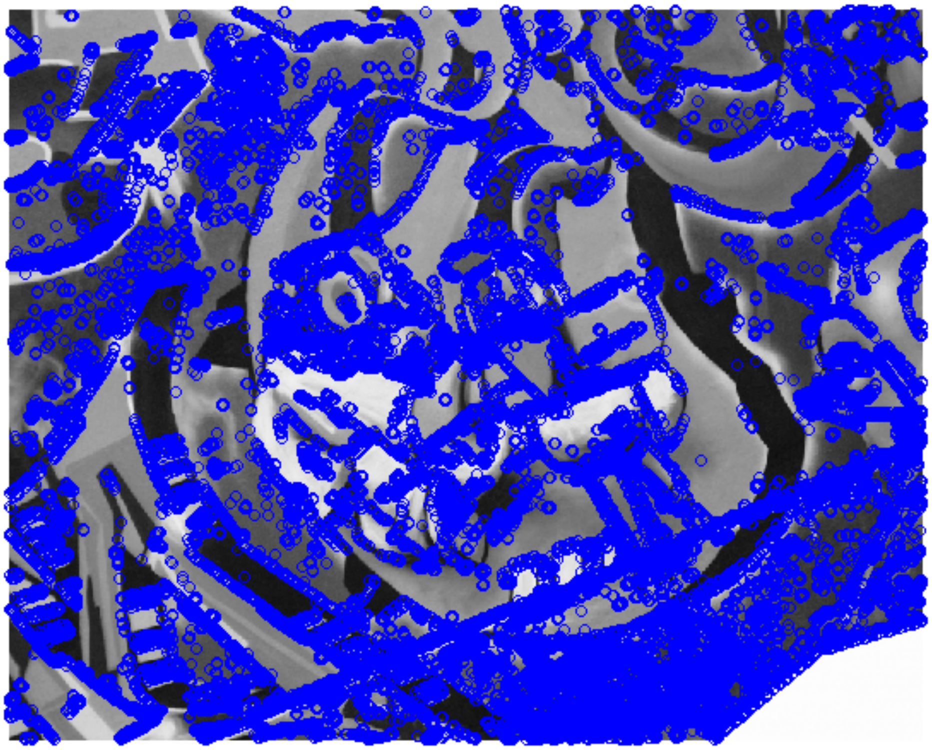


Figure 1 img1.ppm

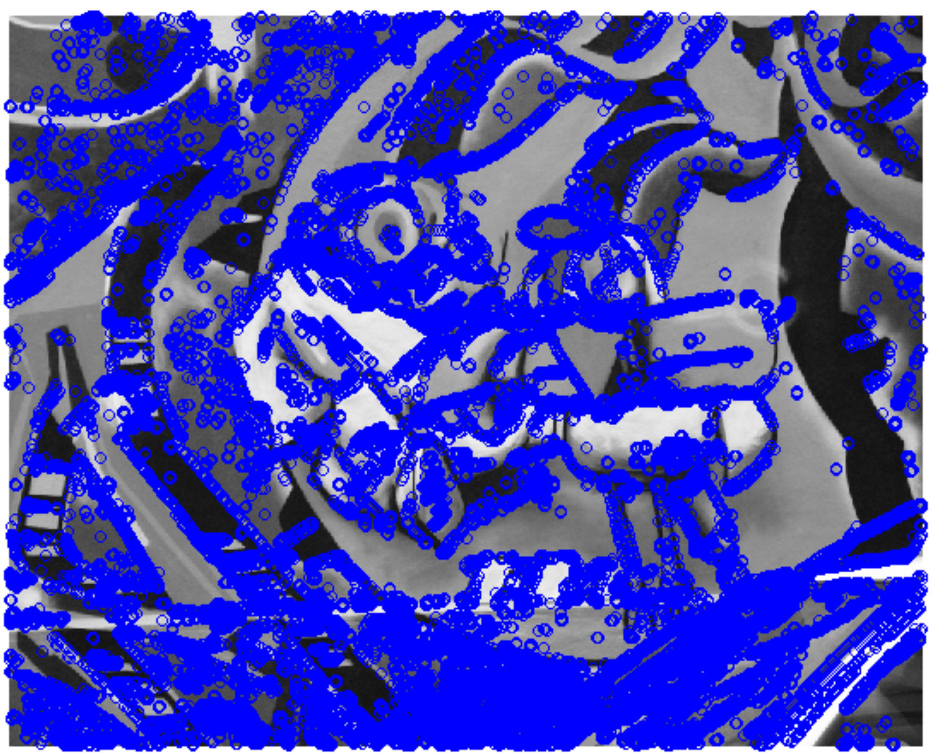


Figure 2 img2.ppm

b) Incomplete… Understood the concept but got lost in the Matlab.

```
function [ D ] = sift_descriptor( image1, image2)
%D Returns 128 Dimension vector of keypoints in both images
%   Detailed explanation goes here


% Step 1: Find keypoints in image 1
s1 = corner_detector(image1);

% Step 2: Find keypoints in image 2
s2 = corner_detector(image2);

% Step 3: Construct 16x16 matrix grid around keypoints (image 1)
D1 = sift_bin8(s1);

% Step 4: Construct 16x16 matrix grid around keypoints (image 2)
D2 = sift_bin8(s2);

% Step 5: Compare vectors

end

function [ D ] = sift_bin8( s1 )

GRID_16 = zeros(16,16);

rows = s1(1).KeypointsRows;
cols = s1(1).KeypointsCols;
angles = s1(1).Angle;

%for k = 1:size(rows(:),1)  %Takes forever to compute... 36,000+ keypoints!
for k = 1:1000              %Just look at first 1000 for time being...
    %Grab next keypoint
    if(k < size(rows(:),1))
        x1 = rows(k) - 8;
        y1 = cols(k) - 8;
    else
        break;
    end

    % Step 4: Run through grid and grab angles
    if(x1 > 0 && y1 > 0)
        for i=1:16
            for j=1:16
                if((x1+i > 0) && (x1+i < size(angles(:,1),1)) && (y1+i > 0) &&
                                 (y1+i < size(angles(1,:),2)))
                    %Still in image bounds...
                    if(angles(x1+i, y1+j) < 0)
                        %Negative radian, add 2pi
                        GRID_16(i,j) = ((2 .* pi) + angles(x1+i,y1+j));
                    else
                        GRID_16(i,j) = angles(x1+i,y1+j);
                    end
                else
                    %Out of bounds!
                    GRID_16(i,j) = 0;
                end
            end
        end
    end
```

```
    % Step 5: Take 4x4 of 16x16 grid and place angles in 8 bins from 0 to 2pi
    %         to get 128D vector.
    a = 0;
    b = 0;
    d128Vector = zeros(128,1);

    while(a < 16)
        while(b < 16)
            for c = 1:4
                for d = 1:4
                   d128Vector(a+c) = GRID_16(a+c,b+d);
                end

            end
            b = b + 4;
        end
        a = a + 4;
        b = 0;
    end
end

D = d128Vector;
end
```

c) Incomplete… Depended on b)

4.

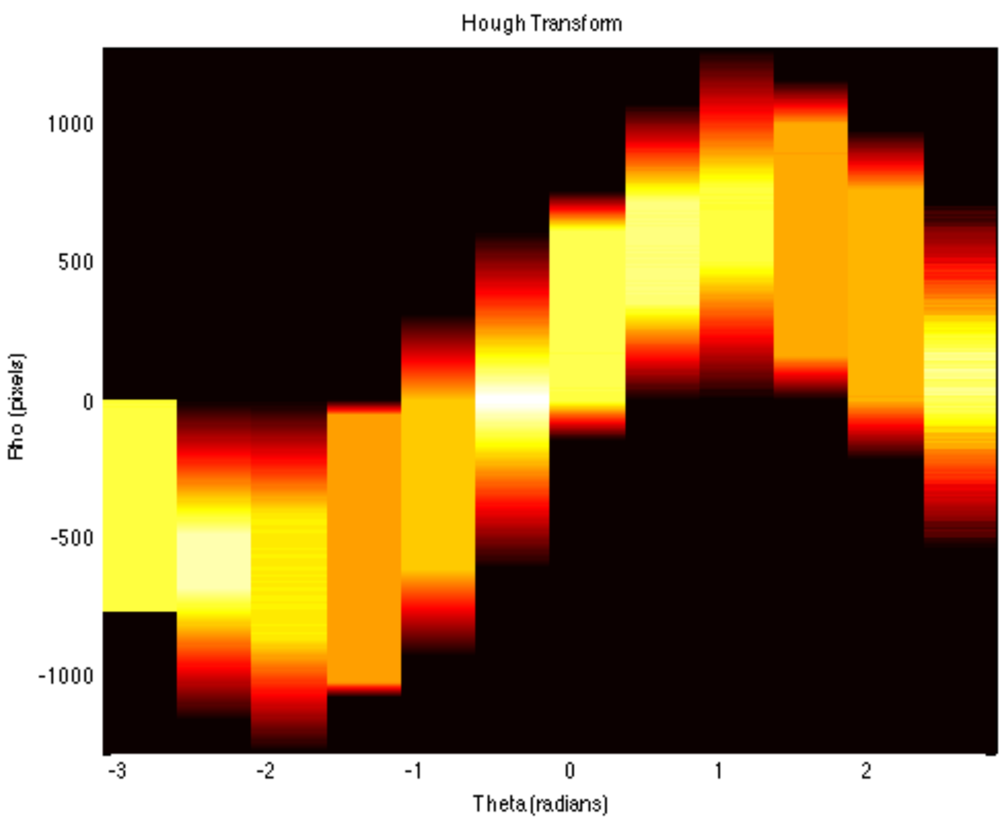
a) Needed some assistance on this code. I've referenced the URL for where I found the basis of the implementation and also tried to follow szeliski.

```
%Used HoughTransform from RosettaCode and Szeliski as guidance/example
%Link: http://rosettacode.org/wiki/Example:Hough_transform/MATLAB
function [ space ] = hough_T( img, theta_freq )
%SPACE Hough Space
%   Detailed explanation goes here

    RGB = imread(img);
    I = rgb2gray(RGB);

    I = flipud(I);
    [w, h] = size(I);

    rhoLimit = norm([w, h]);
    rho = (-rhoLimit:1:rhoLimit);
    theta = (-pi:theta_freq:pi);

    totalThetas = numel(theta);
    space = zeros(numel(rho),totalThetas);

    %Look for the edge pixels
    [x, y] = find(I);

    %Allocate space for accumulator
    totalEdges = numel(x);
    accumulator = zeros(totalEdges, totalThetas);

    %Allocate cosine and sine calculations
    cosine = (0:w - 1)'*cos(theta);
    sine = (0:h - 1)'*sin(theta);

    accumulator((1:totalEdges),:) = cosine(x,:) + sine(y,:);

    %Bin the rhos
    for i = (1:totalThetas)
        space(:,i) = hist(accumulator(:,i),rho);
    end

    pcolor(theta, rho, space);
    shading flat;
    title('Hough Transform');
    xlabel('Theta (radians)');
    ylabel('Rho (pixels)');
    colormap(hot);
end
```

Since I couldn't find the city image, I just found one online of San Francisco.
Command:

```
EDU>> hough_T('San_Francisco_City.jpg', 0.5)
```

Krikor Hovasapian
ECG795 HW#2
3/19/2013



Krikor Hovasapian
ECG795 HW#2
3/19/2013

b)

c) Matlab results:

Command:

```
EDU>> RGB = imread('San_Francisco_City.jpg');
EDU>> I = rgb2gray(RGB);
EDU>> hough(I);
EDU>> BW = edge(I,'canny');
EDU>> [H,T,R] = hough(BW,'RhoResolution',0.5,'Theta',-
90:0.5:89.5);
EDU>> subplot(2,1,1);
EDU>> imshow(RGB);
EDU>> title('Original Image');
EDU>> subplot(2,1,2);
EDU>> imshow(imadjust(mat2gray(H)),'XData',T,'YData',R,...
      'InitialMagnification','fit');
EDU>> title('Hough Transform of San Francisco');
EDU>> xlabel('\theta'), ylabel('\rho');
EDU>> axis on, axis normal, hold on;
EDU>> colormap(hot);
```

Result: