Computer Vision Laboratories

Project 4

EE/CPE 428 - 03

Computer Vision

Winter 2023

Group 6

Students: Jay Sisodia, Jeffrey Wexler, Nathan Jaggers

Project Due: 02/23/23

Instructor: Dr. Zhang

Part 1 - Feature Detection

Assignment

Write a program to implement Harris corner detection. You can not use MATLAB built-in functions other than fspecial() and imfilter(). Test your program on the checkerboard image and Go board image below, and overlay the detected corners on the image using red squares. Describe your effort in tweaking the parameters (such as Gaussian filter size, threshold value for validating a corner).

Procedure

The implementation of the Harris Corner Detection algorithm was quite straightforward, and was based on the example shown to us in class. After reading the grayscale image, a gaussian filter is applied first to clean up noise. Next the gradients of the image are calculated. After calculating our x and y gradient, we can then calculate our second moment matrices. By then summing our second moment matrices over a gaussian window, we can find our Harris Corner response. Finally, the local maxima are then used to determine where the corners are located. The threshold for the local maxima depends on the image. Figure 1 required a higher threshold than figure 2 because of the difference in noise.

Results

When conducting Harris corner detection there are several parameters we can adjust to influence our results. We can adjust the sigma in the initial gaussian filter for smoothing or the gaussian window summing the derivatives from the moment matrix as well as the alpha parameter in the Harris Corner Response, the threshold for R, and/or the disk size determining how many pixels we observe at a time. When testing changes for each parameter individually, the ones with the highest impact to the result were determined to be the sigma of the gaussian filter and the disk size. Below are results for corner detection at varied values. If not specified, the default values are as follows:

sigma 1	sigma 2	alpha	R threshold	disk size
1	2	0.6	2	13

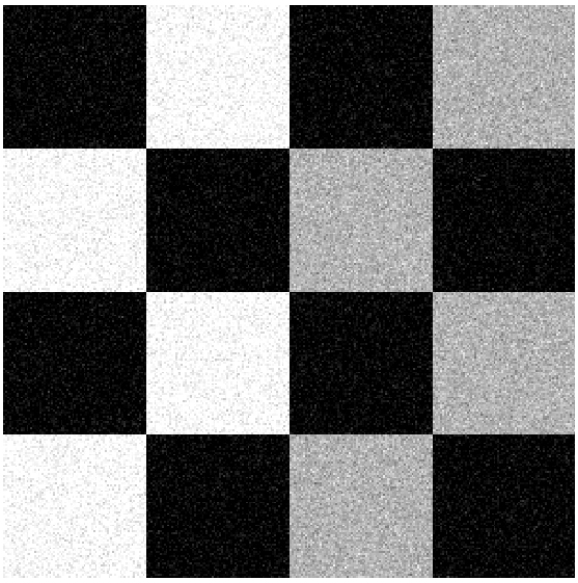

Figure 1 - Original Checkerboard Image

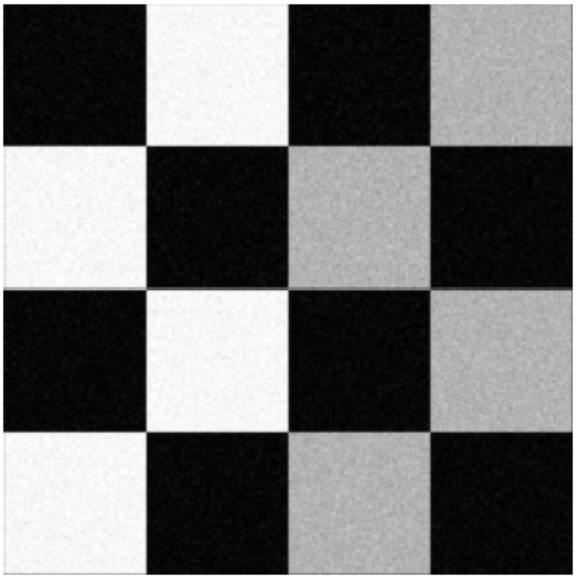

Figure 2 - Smoothed Checkerboard Image, sigma1 = 1

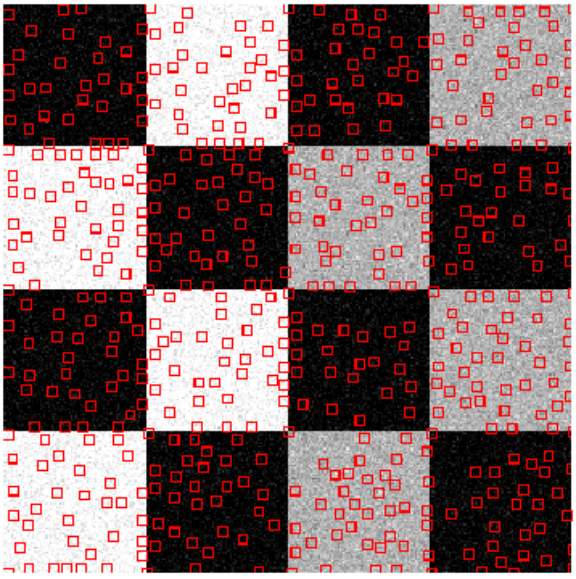

Figure 3 - Checkerboard Image, with applied Harris Corner Dection with default values

Above is a poor job of corner detection. To improve the results, values for the parameters were tweaked to see how they would impact the final results. After playing with the values a bit, we find some successful adjustments.

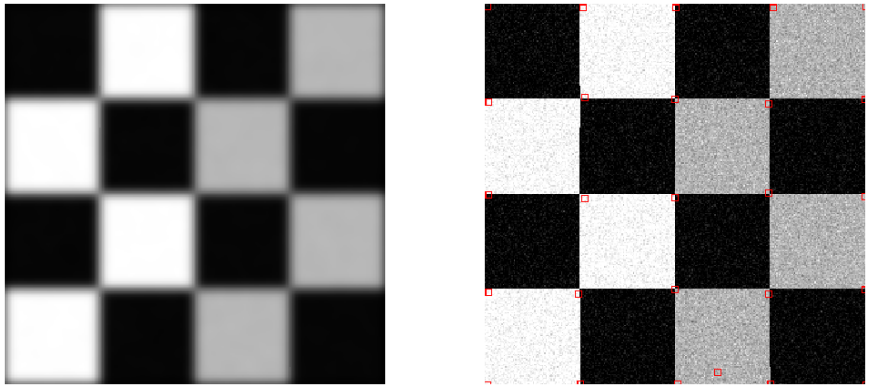

Figure 4 & 5 - Smoothed Checkerboard Image and Results, sigma2= 8

When testing different sigma values, it was found that results improved increasing from 1 to 8 and then started to degrade after increasing past 8. Many more false positives were found in a trial using 10 as the sigma. The best results are show below where instead of blurring the image more, we increase the disk size.

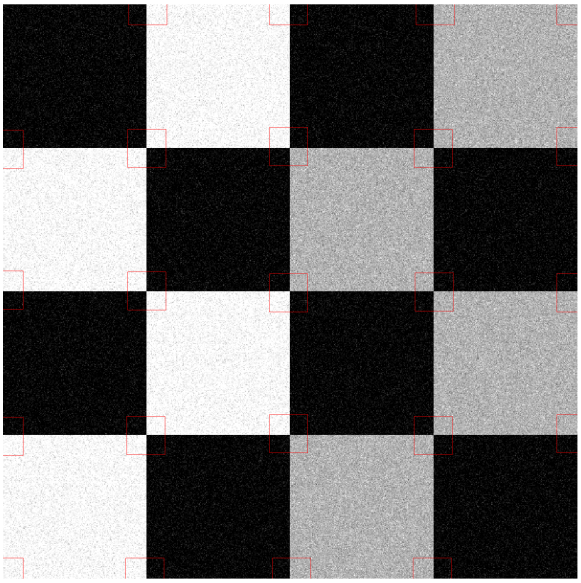

Figure 6 - Checkerboard Image Corner Detection, disk size = 120 Sigma2= 2

In this trial, we can see that we have no false positives for corner detection and that each of the rectangles appear centered on the corner. We also see that there is a 100% accuracy level as all the corners have been identified.

A very similar process was followed for performing corner detection on the Go board image. Default values were used initially and tweaked until satisfactory results were attained. The best results occurred at Sigma2 = 2 and disk = 13. Trying more combinations and variations of variables may give “better” results, however it depends on what should be considered a corner which is in the eye of the user. Our group found that just changing Sigma2 gives the best results as nearly every corner on the board is found, finds the intersection of pieces & squares, and finds the corners of the watermark.

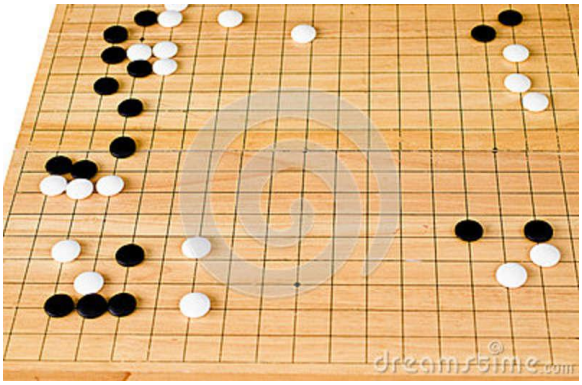

Figure 7 - Go board Image

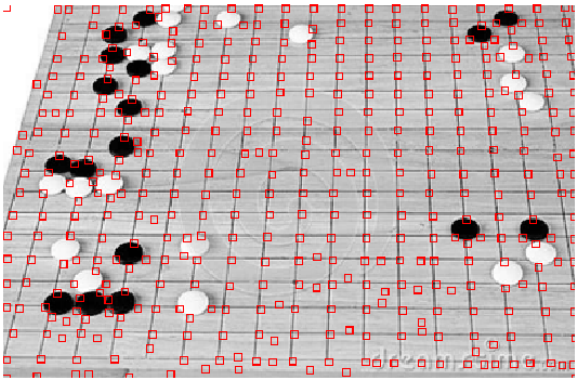

Figure 8 - Go board Image, default values results

The corner detection with the default values does do a pretty good job of detecting corners when compared to default values and the noisy checkerboard. That being said, it also does encounter a lot of false positives still and we can improve our results.

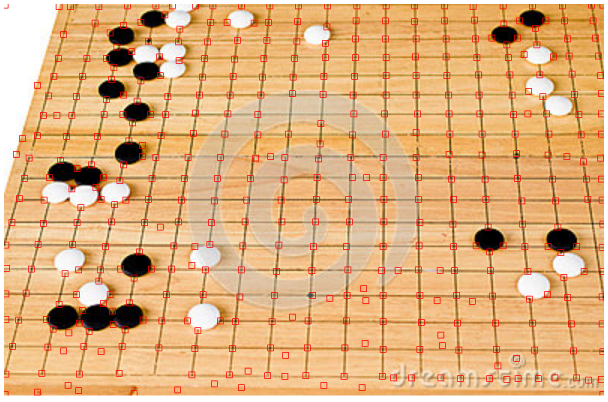

Figure 9 - Go Board Corner Detection, Sigma = 2

Part 2 - Feature Description

Assignment

Write a function that extracts the feature vector from the detected keypoints. The function should have the format: [extracted_features] = my_extractFeatures_a/b(image, detected_pts) For this part, you can use MATLAB's built-in feature detectors (such as FAST, SURF) to detect keypoints. Limit the number of detected key points to no more than 100 per image. Check the coordinates of the returned keypoints.

a. First use raw pixel data in a small square window (say 5x5) around the keypoint as the feature descriptor. This should work well when the images you are comparing are related by only a translation.
b. Next, implement a SIFT-like feature descriptor. You do not need to implement the full SIFT (for example no orientation normalization if no rotation is involved).

Procedure

The procedure for part a is very straightforward. First a built-in function like detectSURFFeatures() is used on our image. This gives us important feature points; next we shorten this list to be the 100 strongest points. After that, we can extract only the coordinate values of these 100 key points. For these 100 key points, we can then iterate through loops to extract a 5x5 window around each keypoint. This 5x5 window is then concatenated to extract our feature descriptor.

Part b uses a similar approach at the start. We use the same method to extract the coordinates for the 100 key points. Now we perform the SIFT feature description creation. For each keypoint we create a 16 by 16 window, which is then split into sixteen 4x4 cells. For each cell, the gradient

magnitude and direction is calculated. A histogram is then created and split into 8 bins based on angle (0 to 45, 45 to 90, 90 to 135…). Based on a pixel value's gradient direction, it's magnitude is added to the corresponding bin. This means for every cell, we get a vector with 8 magnitudes. So every 16x16 window produces a vector with 128 magnitude values. Finally we extract 100 of these vectors for each of our 100 key points.

Result

Because these parts have no visual results, the code has been attached to describe the function operation.

Part a:

```
function [output] = FAST_complete(image)
    % extract 100 strongest keypoints
    points = detectSURFFeatures(image);
    points = points.selectStrongest(100);
    imshow(image)
    hold on
    plot(points)
    % extract keypoint coordinates
    % points is a structure
    coords = round(points.Location)
    for i=1:length(points.Location)
        mat = zeros(5,5);
        vect = zeros(25, 100);
        x=coords(i); y=coords(i+100);
        mat(:,:) = image(y-2:y+2,x-2:x+2)
        vect(1:25,i) = mat(:);
    end
    % each column is the descriptor vector for one point
    output = vect
end
```

Part b:

```
function [output] = SIFT_complete(image)
```

```
    % output = a cell vector with 100 descriptors
    % each descriptor is for 1 keypoint
    % each descriptor is a vector with 128 values

    % obtain 100 keypoints
    points = detectSURFFeatures(image);
    points = points.selectStrongest(100);
    imshow(image)
    hold on
    plot(points)
    % find coordinates for each keypoint
    coords = round(points.Location);
    features = {}; % output cell vector
    for i=1:100
        mat = zeros(16,16);
        x=coords(i); y=coords(i+100);
        % create the 16x16 window
        mat(:,:) = image(y-8:y+7,x-8:x+7);
        vect = zeros(1,128); % feature descriptor
        for j=1:4
            for k = 1:4 % create the  4x4 cell
                cell = zeros(4,4);
                cell(:,:) = mat((j*4)-3:j*4 ,(k*4)-3:k*4);
                [Gx,Gy] = imgradientxy(cell);
                [Gmag,Gdir] = imgradient(Gx,Gy);

                for l=1:8
                    mag = Gmag(l);
                    theta = Gdir(l);
                    % bin location based on angle
                    bin = (fix((theta+180)/45)+1)
                    % bin offset based on current 4x4 cell
                    bin = bin + ((j-1)*16);
                    vect(bin) = mag + vect(bin);
                end
            end
        end
        features{i} = vect;
    end
    output = features;
end
```

The result vector from part A gives us 25 rows and 100 columns. The twenty five rows correlate to the 5x5 surround matrix of each point, while the one hundred columns connects to the total number of points. The result vector from part B gives us 128 rows with 100 column. The one hundred twenty eight rows correlate to the 8x16 surround matrix of each point, while the one hundred columns connects to the total number of points.

Part 3 - Feature Matching

Assignment

Now that you have detected interest points and created the respective feature vectors, the next step is to match them in two images, i.e., given a feature in one image, find the best matching feature in the other image.

a. Write a procedure that compares two features and outputs a distance between them.
b. Find the best match for each detected keypoint in image 1.
c. Your routine should return NULL or some other indicator if there is no good match in the other image. This can be done by
    i. a threshold on the nearest neighbor distance
    ii. the “ratio test”.
d. Perform feature matching on the three pairs of test images shown above (files on Canvas). You can use showMatchFeatures() to display the matches

Procedure

Part a is very simple, for a given vector from image 1 and image 2, the euclidean distance is calculated between them. The following formulae is used and can be seen in the function defined in the results section:

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \mathrm{d}(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

For part b, each detected keypoint in image 1 is compared to each of the 100 key points detected in image 2. Using 2 nested loops and the function defined in part a, we can calculate the distance between every keypoint. First every keypoint in image 2 is compared to one of the key points in image 1, and the point with the smallest distance is found. Next if this smallest distance between two vector descriptors is under a given threshold, we can declare a match has been found between two key points. The output is then a vector, where the index represents the point number of image 1, and the value stored at the index refers to the point number that matches from image 2. If there are no found matches, then at a given index the value will be zero. As an example if we have an output of:

[22, 0, 0, 3…]

Then Keypoint 1 of image 1 matches with keypoint 22 of image 2. Keypoint 2 and 3 of image 1 have no good matches. And Keypoint 4 of image 1 matches with Keypoint 3 of image 2.

Results

```
Part a:
function [output] = feature_distance(vect1, vect2)
    % find euclidean distance between two vectors
    % for vector of length 128
    sum = 0;
    for i=1:128
```

```
        sum = sum + ((vect1(i) - vect2(i))^2);
    end
    output = sqrt(sum);
end

Part b&c:
function [output] = feature_matching(cell1, cell2)
    % compare the 100 keypoints of two images
    % inputs are cell vectors of length 100 containing the
    % feature descriptors
    points = zeros(1,100)
    for i=1:100 % keypoints of image 1
        match = 1000; % threshold
        for j=1:100  % keypoints of image 2
            distance = feature_distance(cell1{i}, cell2{j});
            if distance < match;
                match = distance;
                points(i) = j;
                % if index value is left a 0,
                % then no good match was found
            end
        end
    end
    % index corresponds to location on image 1
    % stored value corresponds to location on image 2
    output = points;
end
```

```
Part d:
im1 = imread('cars1.ppm');
I1 = rgb2gray(im1);
imshow(I1)
im2 = imread('cars2.ppm');
I2 = rgb2gray(im2);
imshow(I2)

points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
[f1, vpts1] = extractFeatures(I1,points1.selectStrongest(100));
[f2, vpts2] = extractFeatures(I2,points2.selectStrongest(100));


indexPairs = matchFeatures(f1, f2) ;
matchedPoints1 = vpts1(indexPairs(:, 1));
```

```
matchedPoints2 = vpts2(indexPairs(:, 2));

figure;
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2);
title('Putative point matches');
legend('matchedPts1','matchedPts2');
```

Results

Figure 10 & 11 - Original Bikes 1 and Bikes 2 images

Figure 12 - Feature matching bikes images

Figure 13 & 14 - Original Cars 1 and Cars 2 images

Figure 15 - Feature matching cars images

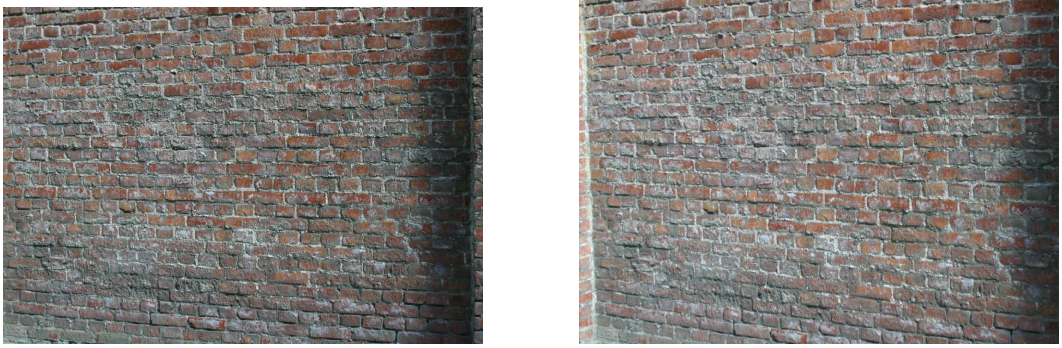

Figure 16 & 17 - Original Wall 1 and Wall 2 Images

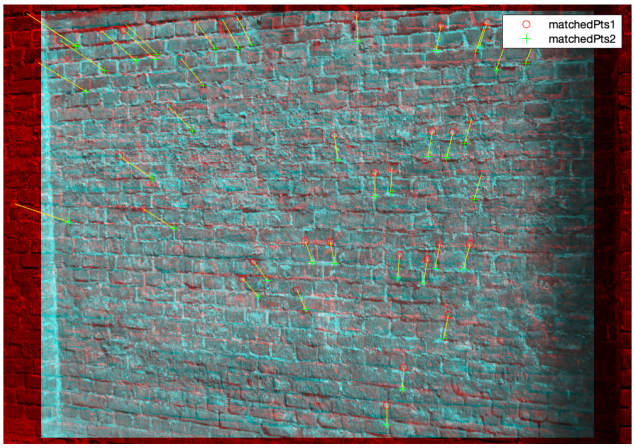

Figure 18 - Feature matching Wall images

The results for all three test cases are very accurate and succesful. As show, the wall and the cars were the more accurate image sets while the bikes images had a few mismatches. This mismatches can be corrected by using special techniques such as RANSAC. In the wall and car images we see can visually see nearly all points match up, as well as the slopes of the points corrolating meaning that they are indeed matches.

Reflection

Jay Sisodia

I learned that implementing a feature detection algorithm can be extremely complex. The hardest part was determining the best way to store various types of data, especially with the SIFT algorithm. I also realized that this is a very computationally intensive task. Even for 100 key points for two images, doing all the math and data storage for each feature vector takes a while.

Jeffrey Wexler

I've learned alot about Harris corner detection and was extremely enjoyable. I learned that Harris corner detection is extremely useful and has a wide range of application. One similar personal project that I have been working on is a chess computer vision system to identify all the white pieces vs all the black pieces like this project. I have also been working on trying to identify the corners of the board, between the white and black squares and will be able to use Harris corner detection. I also learned about SIFT, or very similar functions, and how difficult it is to implement on your own. Creating a feature description is very complicated and hard to do without the built in functions. A big part of this project was learning some of the built in features and just applying them. I am looking forward to learning about color image processing and image segmentation.

Nathan Jaggers

This project was a great experience to gain some insight and hands on experience with corner detection, feature detection, and feature matching. During part A it was very interesting to implement the Harris corner detection and see how effective it was on the checker board. When

fine tuning values and getting a sense of what parameters made the most difference in the results it was enlightening to see that what worked for the checkerboard was not ideal for the go board. In fact the parameters that made little difference in the checkerboard seemed to be very effective in the go board. This goes to show you that every situation/enviornment requires different specifications. Personally, Part B and C were quite difficult. I felt I had a solid grasp of the basic concepts in how SIFT worked as well as feature extraction and matching, but I had gaps in my understanding that this assignment certainly help me fill. I have an appreciation for the complexity and robustness of SIFT as well as the computations involved in creating the keypoint descriptors, and conducting the feature matching.

Teamwork

Everyone on the group contributed fairly evenly on the report. When it came to the code itself each member did each part individually and compared results. The code on this report is parts from each person. We all did part 1 together in class, Jay primarily did part 2, and Jeffrey did part 3.

Appendix

Part 1

```
close all;
clear;
clc;

%%
% read image and cast to double so negative values are allowed
during computation
%image = double(imread("checkerboard-noisy1.tif"));
image = double(rgb2gray(imread("go1.jpg")));
figure(1); imshow(image,[]);

%%
% apply Gaussian filter
figure(2);
sigma = 1;
I = imfilter(image, fspecial("gaussian", round(6*sigma+1),
sigma));
imshow(I,[]);

%%
% compute the gradient
Sx = [-1 -2 -1 ; 0 0 0; 1 2 1]; Ix = imfilter(I,Sx);
Sy = Sx'; Iy = imfilter(I, Sy);

% compute M (the second moment matrix)
Ixx = Ix.*Ix;
Ixy = Ix.*Iy;
Iyy = Iy.*Iy;

% Sum the dertivatives over the window size using Gaussian
window
sigma = 3;
W = fspecial('gaussian', round(6*sigma+1), sigma);
M11 = imfilter(Ixx, W);
M12 = imfilter(Ixy, W);
M22 = imfilter(Iyy, W);

% compute harris corner response
alpha = 0.06;
detM = M11.*M22 - M12.^2;
```

```
traceM = M11 + M22;
R = detM - alpha*traceM.^2;

% apply maximum suppression
% find local maxima
R_thresh = 10e5;
Lmax = (R == imdilate(R,strel('disk',15)) & R > R_thresh);
[rows, cols] = find(Lmax); %get a list of indicies of potential
interest points

% draw a box around the detected corners
figure(3);
imshow(image,[]);
hold on
for i = 1:length(rows)
    x = cols(i);
    y = rows(i);
    rectangle('Position',[x-2 y-2 10 10],'EdgeColor','r');
end

%%
% comparing to built-in matlab functions
%image = imread("checkerboard-noisy1.tif");
image = rgb2gray(imread("go1.jpg"));
corners = detectHarrisFeatures(image);
imageNboxes = insertMarker(image, corners,
"circle",'color','red','size',5);
figure; imshow(imageNboxes);
```

Part 2

```
Part a:
function [output] = FAST_complete(image)
    % extract 100 strongest keypoints
    points = detectSURFFeatures(image);
    points = points.selectStrongest(100);
    imshow(image)
    hold on
    plot(points)
    % extract keypoint coordinates
    % points is a structure
    coords = round(points.Location)
    for i=1:length(points.Location)
```

```
        mat = zeros(5,5);
        vect = zeros(25, 100);
        x=coords(i); y=coords(i+100);
        mat(:,:) = image(y-2:y+2,x-2:x+2)
        vect(1:25,i) = mat(:);
    end
    % each column is the descriptor vector for one point
    output = vect
end
```

Part b:

```
function [output] = SIFT_complete(image)
    % output = a cell vector with 100 descriptors
    % each descriptor is for 1 keypoint
    % each descriptor is a vector with 128 values

    % obtain 100 keypoints
    points = detectSURFFeatures(image);
    points = points.selectStrongest(100);
    imshow(image)
    hold on
    plot(points)
    % find coordinates for each keypoint
    coords = round(points.Location);
    features = {}; % output cell vector
    for i=1:100
        mat = zeros(16,16);
        x=coords(i); y=coords(i+100);
        % create the 16x16 window
        mat(:,:) = image(y-8:y+7,x-8:x+7);
        vect = zeros(1,128); % feature descriptor
        for j=1:4
            for k = 1:4 % create the  4x4 cell
                cell = zeros(4,4);
                cell(:,:) = mat((j*4)-3:j*4 ,(k*4)-3:k*4);
                [Gx,Gy] = imgradientxy(cell);
                [Gmag,Gdir] = imgradient(Gx,Gy);

                for l=1:8
                    mag = Gmag(l);
                    theta = Gdir(l);
                    % bin location based on angle
```

```
                    bin = (fix((theta+180)/45)+1)
                    % bin offset based on current 4x4 cell
                    bin = bin + ((j-1)*16);
                    vect(bin) = mag + vect(bin);
                end
            end
        end
        features{i} = vect;
    end
    output = features;
end
```

Part 3

```
Part a:
function [output] = feature_distance(vect1, vect2)
    % find euclidean distance between two vectors
    % for vector of length 128
    sum = 0;
    for i=1:128
        sum = sum + ((vect1(i) - vect2(i))^2);
    end
    output = sqrt(sum);
end
```

```
Part b&c:
function [output] = feature_matching(cell1, cell2)
    % compare the 100 keypoints of two images
    % inputs are cell vectors of length 100 containing the
    % feature descriptors
    points = zeros(1,100)
    for i=1:100 % keypoints of image 1
        match = 1000; % threshold
        for j=1:100  % keypoints of image 2
            distance = feature_distance(cell1{i}, cell2{j});
            if distance < match;
                match = distance;
                points(i) = j;
                % if index value is left a 0,
```

```
                % then no good match was found
            end
        end
    end
    % index corresponds to location on image 1
    % stored value corresponds to location on image 2
    output = points;
end
```

Part d:

```
im1 = imread('cars1.ppm');
I1 = rgb2gray(im1);
imshow(I1)
im2 = imread('cars2.ppm');
I2 = rgb2gray(im2);
imshow(I2)

points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
[f1, vpts1] = extractFeatures(I1,points1.selectStrongest(100));
[f2, vpts2] = extractFeatures(I2,points2.selectStrongest(100));


indexPairs = matchFeatures(f1, f2) ;
matchedPoints1 = vpts1(indexPairs(:, 1));
matchedPoints2 = vpts2(indexPairs(:, 2));

figure;
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2);
title('Putative point matches');
legend('matchedPts1','matchedPts2');
```