EECE5639 Computer Vision Project 4 Report

Liao, Zhi Yuan
Dept. of EECE
Northeastern University
liao.z@husky.neu.edu

Lin, Yu Shun
Dept. of EECE
Northeastern University
lin.yus@husky.neu.edu

I. Abstract:

In this project, Object Category Detection, we have implemented an algorithm to detect objects of a given class by using a Generalized Hough Transform (GHT) Approach. We are going to use different types of Computer Vision techniques such as use the Harris Corner Detector (HCD) and Non-Maximum Suppression (NMS) algorithms that used from the second project. In addition, we are going to get familiar with the use case of Bag of Word (BoW) and displacement vectors to set up a voting mechanism on the GHT domain for object detection.

II. Description of Algorithms:

The following is a simple description of the Algorithm we generated for our project

Training Stage:

1. Read images from the CarTrainImages folder.
2. Apply HCD to find corners training images.
3. Use NMS to find the highest points with the highest interested value.
4. Extract 25x25 pixels image patch around each point of interest.
5. Vectorize of raw pixel intensities as descriptors.
6. Combine all descriptors from all patches into a BoW.
7. Classify the BoW using K-Means and identify their center of mass
8. For each feature in K, record their displacement vectors.

Testing Stage:

9. Read images from the CarTestImages folder.
10. Apply HCD to find corners on testing images.
11. Use NMS to find the highest points with the highest interested value.
12. Extract 25x25 pixels image patch around each point of interest.
13. Vectorize of raw pixel intensities as descriptors.
14. Classify all descriptors using the shortest distances from the centers of mass learned from K-Means

15. For each image, using displacement vectors learned from training to vote on the GHT space (vote_board).
16. Find the areas with the highest votes in the accumulator matrix (vote_board).
17. Create rectangle areas from the points found (center of the area).

Scoring Stage:

18. Load CarsGroundTruthBoundingBoxes from the GroundTruth folder.
19. Construct the ground truth matrix from each entry
20. Calculate the overlapping area.
21. Calculate a score based on the overlapping area over the union of rectangles.

III. Experiments:

Experiments for different Harris corners function:

Since we have tried using different versions of corner detection:

- HarrisCornerDetector (By Liao)
- Harris (By Lin)
- Computer Vision harrisFeature (MATLAB)

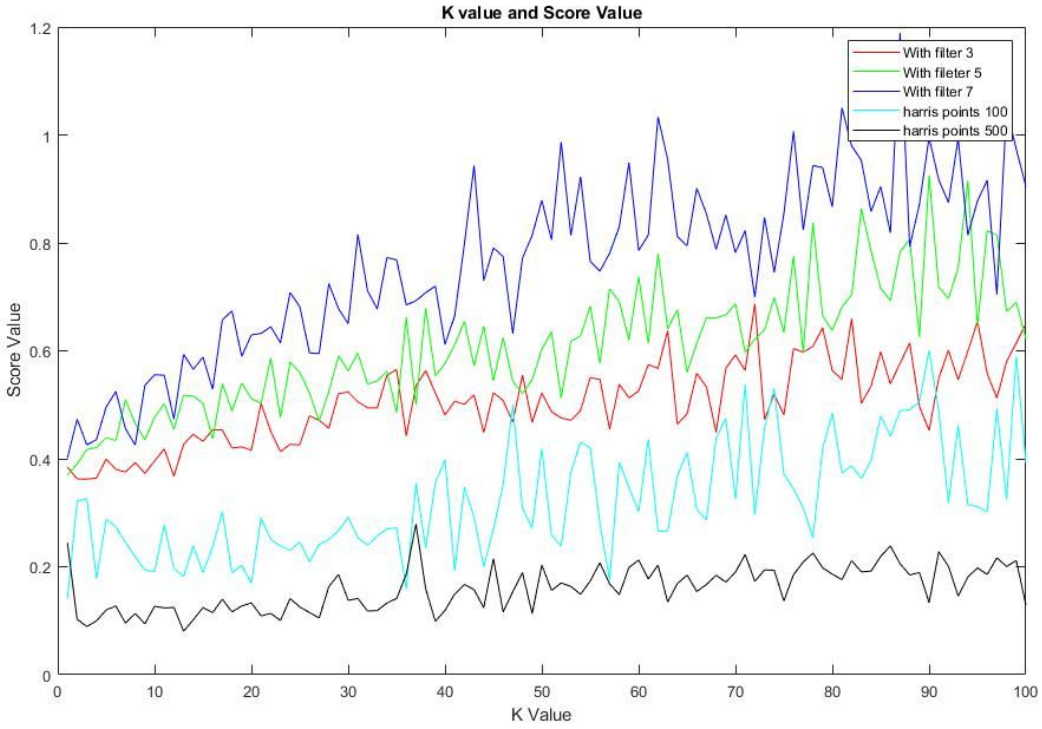
We have used these detectors and found that the result of the voting score will be different because the threshold setting, number of vocabulary used and corner decision in each harris detector is different.

We have experiment on the following situation:

- Find the maximum value of harris defined as the corners
- Define the specific number of corners
- Apply different smoothing filter when finding corners (the number of corners will be different)
- Different number of vocabulary size
- Different number of cluster for K-mean

The relationship of value K of K-Mean clustering and Final Voting Score value

EECE5639 Computer Vision Project 4 Report

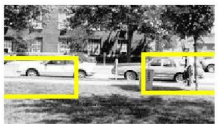


Figure(1), the relationship of K value and Scores

According to the testing, we can choose the best harris corner detector for the good case. Moreover, we also test the other harris detector to compare the difference between the results.

Here, we chose some results which plugin with the specific K value, each clustering is up to 2000 iterations to converge.

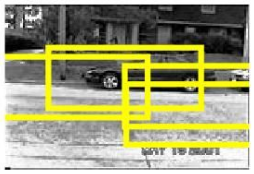
Some good case when K = 16, Smooth Filter = 7*7

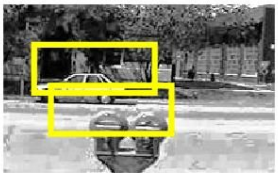


In the test image number 5, 50, and 100. The rectangle roughly could find the vehicle's location. But some vehicles could not be found because the vocabulary set is small which makes feature match range huge and confused the detector, like the case of test image number 5.

EECE5639 Computer Vision Project 4 Report

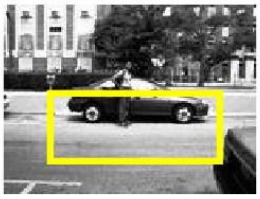
We use K = 16 and the harris feature smoothing filter with 7*7 filter.



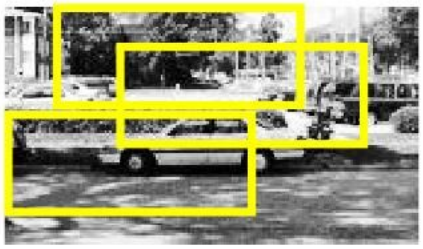


Here show the bad case of K = 16, with filter = 7*7. In these two cases, we could see the detector made decisions on more than one location. This might be caused by the corners feature detectors, which decide similar corners as the same features. Thus, we try to strict the threshold and test if it is getting better or not. Moreover, we also tried other K values and other harris corner detectors.

With K = 90, filter = 5*5. According to the K value and score plot, with filter 5*5 would not get better result than 7's one. However, some testing cases still show better results.



In test image 34 and 31, the detector shows the right location of the vehicle. This might be caused the picture has clear features. (clear wheels and no large obstacles that will make the detector confused.



.

A bad case of the K = 90 smoothing Filter =5*5 detector. The trees and shadows have occluded the features and causes the detector be confused. Thus, the detector made the wrong decision voting on the wrong features.

EECE5639 Computer Vision Project 4 Report

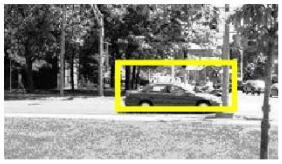
To prove that our idea, which methods will make the decision be worse, is correct. We choose the harris corner detector by setting specific corners. We set the number of corners = 500, which means the threshold would be very low to decide if it is a feature.

According to the figure , we already know that the score for each testing case is very low. Harris corners count 100 show a higher score from the training images. These show the more feature will make the detector more confused.

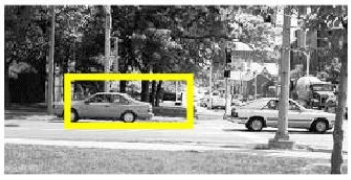


The testing case above shows that the detector can not find the right location. the testing cases show the decision is out of the range. The decision rectangle might have a shape similar or description similar blocks caused by the low threshold.

Thus, we choose a harris corner detector that only make the decision on maximum value as the feature. (harrisCornerDetector by Liao)

Good case using K = 16 with only take the max number of votes in the patch. Both the vehicles are detected correctly in these cases. But a little bias happened on the left one. This might caused by the ground shadow is more obvious than the right one. Which means there are corners make the decision be not fully overlap.



In left test case, the vehicles on the right side are not detected. This might caused by less features on the right vehicle or occlusion.

EECE5639 Computer Vision Project 4 Report

IV. Values of Parameters Used:

For HCD, the Threshold was set to +/-50000;

s.t. R(i,j)>+TH ⇒ Corners;
R(i,j)<-TH ⇒ Edges;
Else ⇒ Flats;

K-Mean clustering testing value 1-100, clustering iteration max = 2000
Harris corners feature detecting Filter Size : 3, 5, 7
Harris specific extract corners points: 100
Detection area 100 x 40 pixels (WxH) with Linewidth = 3
2-Dimensional GHT space of location of the pixels.

V. Observations:

Observation of different amount of vocabulary descriptions:

We used multiple combination of threshold, corner detection methods, different size of the vocabulary set and different feature of patches are used in the training processing. In the smoothing corner detector, we find that the larger the filter we used, the fewer features will be detected. This influences the amount of vocabulary set. We found if the source is less of vocabulary, some vehicles could not be detected in the result. Increase the vocabulary amount does solve the problem. However, when the vocabulary amount is too much, which means there are too many corners decided as features. The score will also decrease. (Figure 1).

The simple idea is, the different features detector might have different number of vocabulary. And the training clustering will also influence the number of vocabulary. Originally, training iteration is 100 did not pass the good result in the end. We chose up to 2000 that make sure it will covergend

The average score is over 60 percent when K = 16 with Smoothing filter = 7*7. As the growth of the K value. The score seems to increase. However, different situations happened in different training images. This begets to the non-stationary increasing of the score when K increases. We have tried that K = 87 which has a high score around 0.91. Seems the result become better. But the detector might still be confused when the vocabulary is too much.

With smoothing method to detect the feature shown the best result. Average score can be more than 60 percent when K > 50. By specific the amount of corner harris method.

EECE5639 Computer Vision Project 4 Report

Average score is less than 20 percent if the corners (features) are too much. With max_session method, score is around 0.4~0.6.

According to the observation above. An appropriate K value and corner detector should be chosen for training. Smoothing filter 7*7 makes the feature in each train picture appropriately (around 15~30 features). This makes the final result be the best in our testing. We think there might have another parameter to make the result being better, or more train sets should be used when training to let the prediction be more correctly.

VI. Conclusions:

We have learned several techniques from this project, which include using GHT and displacement vectors for object detection. The experiment was able to identify around 60 percent of testing images correctly (score is bigger than 0.5). There are several challenges/improvements we can make. The object we want to track appears at multiple maxima in the test image and finding the reasonable point(s) is difficult to predict. Another challenge we found and maybe a good project we can do in the future is the best approach to determine the number of vocabulary set to use with image set. Moreover, we could try some more training sets. Generally the learning take about more than 1 million training samples. Thus, we should do that.

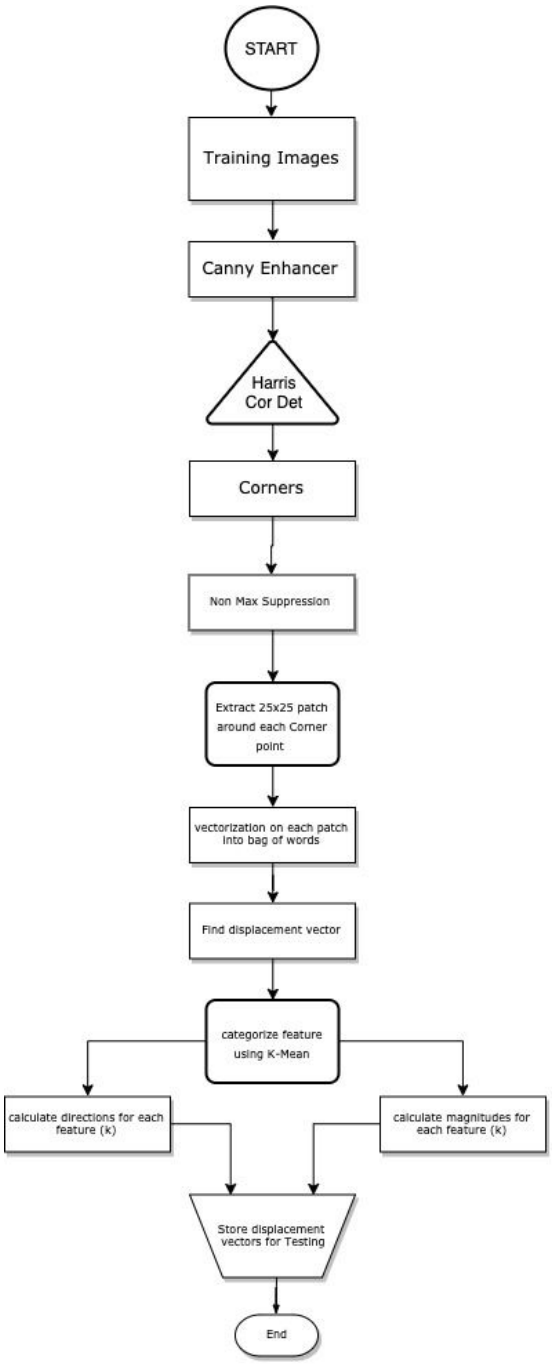
VII. References:

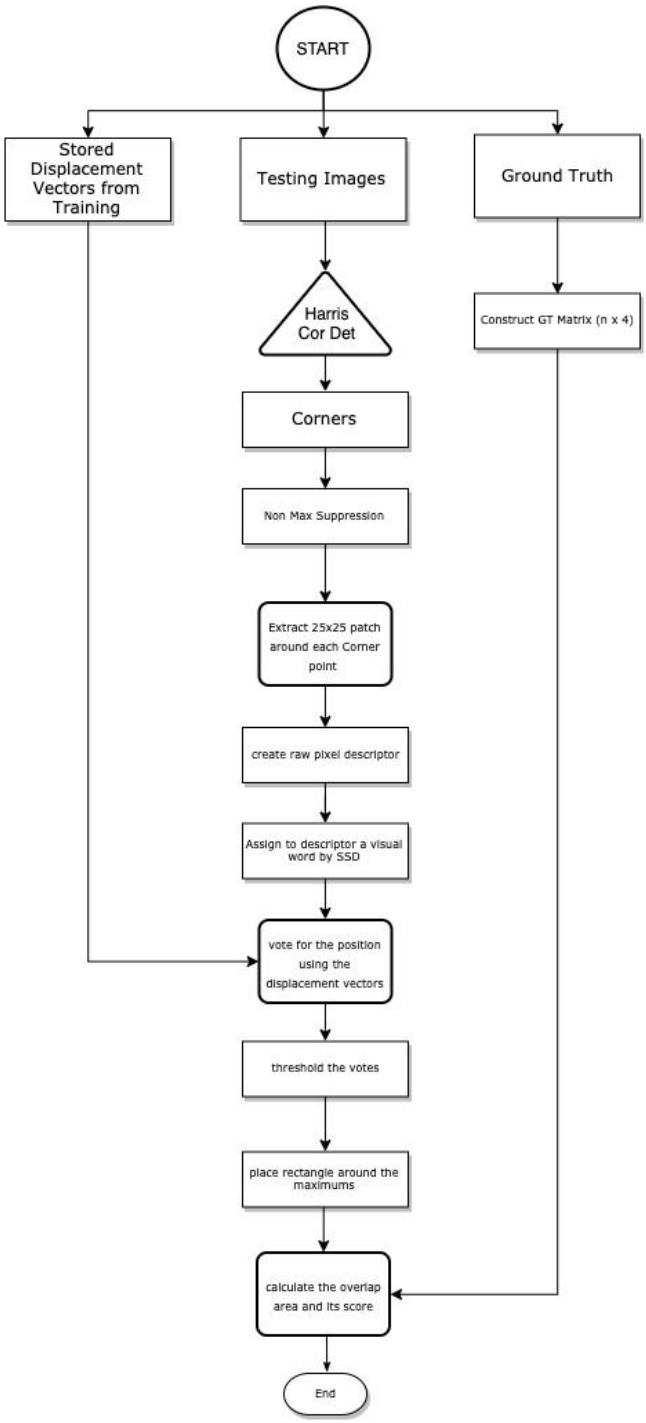
R. Szeliski, Springer. (2010) “Computer Vision Algorithms and Applications”
http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf
Matlab Documentation
https://www.mathworks.com/help/
Comaniciu, Dorin., Meer, Peter. “Robust Analysis of Feature Spaces: Color Image Segmentation” http://comaniciu.net/Papers/RobustAnalysisFeatureSpaces.pdf
Min, D., Choi, S., Lu, J., Ham, B., Sohn, K. & Do, M. (2014) “Fast Global Image Smoothing Based on Weighted Least Squares”
https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6942220
Shi,Jianbo., Tomasi, Carlo. “Good Feature to Track”
http://www.ai.mit.edu/courses/6.891/handouts/shi94good.pdf

EECE5639 Computer Vision Project 4 Report

VIII. Appendix:

FLOWCHART





EECE5639 Computer Vision Project 4 Report

Matlab Codes:

```
close all; clear; clc;

% for reproducibility, this can be commented out for real training.
rng('default');

bagofwords={}; %training
train_displ={}; %train displacement

k=20;

%% Training
% Given a set of training cropped images prepare a vocabulary and
% GHT translation vectors:
tic;
image_folder = '/Users/zhiyuanliao/Documents/MATLAB/CarTrainImages/';
file_names = dir(fullfile(image_folder,'*.jpg'));
total_images = numel(file_names);

for i=1:total_images
    ImageA = imread(fullfile(image_folder,file_names(i).name));
    %ImageA_gray = double(rgb2gray(ImageA));
    %imshow(ImageA);

    % Use detector to collect interesting points in the training images.
    [R_ImageA, Corners_A] = HarrisCornerDet(ImageA);
    %imshow(Corners_A);

    % Apply Canny_Enhancer to the training image;
    %[Es_A, Eo_A] = canny_enhancer(ImageA);
    % Apply non-maimum suppression to get a sparse set of corner features.
    [I_N_A] = nonMax_Supression(Corners_A, R_ImageA);
    %imshow(I_N_A);
    %showInterestPoints(ImageA,I_N_A)

    % At each interest point, extract a fixed (25 ×25 pixels)size image patch
    % use the vector of raw pixel intensities as the descriptor.
    [img_descriptors, img_points] = extractPatches(ImageA, I_N_A);
    bagofwords{end+1} = img_descriptors;
    train_displ{end+1} = img_points;
```

```
end

% convert bag of word from cell to matrix
bagofwords=[bagofwords{:}];
train_displ=[train_displ{:}];

% Cluster the patches from the training images into clusters using K-means
% (use kmeans command in Matlab). This step is meant to significantly
% reduce the number of possible visual words. The clusters that you find
% here constitute a visual vocabulary.
[idxOfKmean, C] = kmeans(bagofwords',k); % it needs to find the best k value

% Having found the vocabulary "C", to assign each training example of local
% patches to visual words in the vocabulary. An image patch is
% assigned to the visual word which is closest using Euclidean distance (SSD).

% For each visual word occurrence in the training examples, record the
% possible displacement vectors between it and the object center. Assume
% that the object center is the center of the cropped training image.
% Note that a given visual word may have multiple displacement vectors,
% depending on its appearance (for example, the "wheel-like" word in car images.)
disp_mag=zeros(1,length(train_displ));
disp_dir=zeros(1,length(train_displ));

for i=1:length(train_displ)
   disp_mag(i)=norm([20,50]-train_displ(:,i)');
   disp_dir(i)= -atan2d((50-train_displ(2,i)),(20-train_displ(1,i)));
end

% show total time spent on running training
fprintf('Total training time spent: %f\n', toc)
%% Testing
load('/Users/zhiyuanliao/Documents/MATLAB/GroundTruth/CarsGroundTruthBoundingBoxes.m
at')

tic;
image_folder = '/Users/zhiyuanliao/Documents/MATLAB/CarTestImages/';
file_names = dir(fullfile(image_folder,'*.jpg'));
total_images = numel(file_names);

% manually change the image or run the whole image list
TESTIMAGE = 1;     %this is same as i
```

EECE5639 Computer Vision Project 4 Report

```
for i=1:1   % to test with single image from CarTestImages
    % Given a novel test image, detect instances of the object
    ImageB = imread(fullfile(image_folder,file_names(TESTIMAGE).name));
    [ib_row,ib_col]=size(ImageB);
    vote_board=zeros(ib_row, ib_col);
    %ImageB_gray = double(rgb2gray(ImageB));
    %imshow(ImageB);

    % Extract the groundtruth matrix
    [GT_row, GT_col]=size(groundtruth(TESTIMAGE).topLeftLocs);
    if GT_row == 1
        GT=struct2array(groundtruth(TESTIMAGE));
    else
        temp1=ones(GT_row,1)*groundtruth(TESTIMAGE).boxW(1);
        temp2=ones(GT_row,1)*groundtruth(TESTIMAGE).boxH(1);
        GT=horzcat(groundtruth(TESTIMAGE).topLeftLocs, temp1, temp2);
    end

    % Run the corner detector to find interesting points.
    [R_ImageB, Corners_B] = HarrisCornerDet(ImageB);
    %imshow(Corners_B);

    % Apply Canny_Enhancer to the testing images;
    %[Es_B, Eo_B] = canny_enhancer(ImageB);
    % Apply non-maimum suppression to get a sparse set of corner features.
    [I_N_B] = nonMax_Supression(Corners_B, R_ImageB);
    %imshow(I_N_B);
    %showInterestPoints(ImageB,I_N_B)

    % At each interesting point, use a fixed image patch (of the same size
    % as the ones used during training) to create a raw pixel descriptor.
    [test_descriptors, test_points] = extractPatches(ImageB, I_N_B);

    % Assign to each patch a visual word.
    [t_row, t_col]=size(test_points);
    for j=1:t_col
        %distances = pdist2(test_descriptors(:,j)', C);
        %[minDistance, indexOfMinDistance] = min(distances);
        [minDist, idxOfMinDist] = min(pdist2(test_descriptors(:,j)', C));

        %Let each visual word occurrence vote for the position of the
```

```
    % object using the stored displacement vectors.
    for l=1:length(idxOfKmean)
        if idxOfKmean(l) == idxOfMinDist
            % start point: test_points
            % direction: disp_dir
            % magnitude: disp_mag
            vote_row=round(test_points(1,j)+disp_mag(l)*sind(disp_dir(l)));
            vote_col=round(test_points(2,j)+disp_mag(l)*cosd(disp_dir(l)));
            if vote_row>0 && vote_row<=ib_row && vote_col>0 && vote_col<=ib_col
                vote_board(vote_row,vote_col)=vote_board(vote_row,vote_col)+1;
            end

        end
    end

end

% After all votes are cast, analyze the votes in the accumulator array,
max_vote=max(vote_board(:));
%imshow(vote_board);

% threshold and predict where the object occurs.
%vote_board(find(abs(vote_board)<max_vote)) = 0;    %Not efficient
vote_board=vote_board.*(vote_board>max_vote-1);    %Better efficiency
imshow(vote_board);
[rec_row, rec_col]=find(vote_board);


% To predict the fixed size bounding box placement, assume that the
% object center is the bounding box center. Note that the object of
% interest may occur multiple times in the test image.
% choose boxW = 100 boxH = 40; same as groundtruth
overlap = 0;        % calculate total overlap area
score = 0;          % calculate total score
imshow(ImageB);
hold on;
for j=1:length(rec_row)
    rectangle('Position',[rec_col(j)-50 rec_row(j)-20 100 40],...
        'EdgeColor','y','LineWidth',3)

    % Compute the accuracy of the predictions.
    % based on the overlap between the predicted and true bounding boxes.
```

```
        % Specifically, a predicted detection is counted as correct if the area of
        % the intersection of the boxes, normalized by the area of their union exceeds 0.5.
        area = rectint([rec_col(j)-50 rec_row(j)-20 100 40],GT);

        overlap = overlap + sum(area(:));
        score = overlap/((GT_row + 1) * groundtruth(TESTIMAGE).boxW(1) *...
            groundtruth(TESTIMAGE).boxH(1) - GT_row*overlap);

    end
    hold off;
    fprintf('Score for Image %i is: %f\n', TESTIMAGE, score)
end

% show the total time spent on running training
fprintf('Total testing time spent: %f\n', toc)


function [R, Corners_HCD]  = HarrisCornerDet(I)

% Compute Harris R function over the image
% Input as an image I
%   I: input image
% Output as Harris R function over the image I and sparse set of corner features
%   R: Harris R function over the image I.
%   Corners_HCD: Corners of the image I
%

% Check the dimensions of the image. number of Color Bands should be = 1.
[row, col, band] = size(I);
if band > 1
    % It's not gray scale Image and convert it to gray scale Image
    I_gray = rgb2gray(I);
elseif band == 1
    % It's grayscale Image and copies it
    I_gray = I;
end

R=zeros(row,col);
M=zeros(2,2);
k=0.05;
highTH=50000;
lowTH=-50000;
```

EECE5639 Computer Vision Project 4 Report

```
Corners_HCD=zeros(row,col);
Edges_HCD=zeros(row,col);
Flats_HCD=zeros(row,col);

% Compute the Image Gradient Gx, Gy
[Gx,Gy] = imgradientxy(I_gray);

% Compute products of derivatives at each pixel
Gxx = Gx.*Gx;
Gyy = Gy.*Gy;
Gxy = Gx.*Gy;

% Compute the sums of the products at each pixel using a window averaging:
Sxx = movmean(Gxx,9);
Syy = movmean(Gyy,9);
Sxy = movmean(Gxy,9);


for i = 1:row
   for j = 1:col
      % Define the Matrix at each pixel M = [Sxx Sxy ; Sxy Syy]
      M = [Sxx(i,j), Sxy(i,j); Sxy(i,j), Syy(i,j)];

      % Compute the response R = det(M) - k trace(M)^2
      R(i,j) = det(M) - k*(trace(M)^2);

      % Threshold R identify Corner, Edges and Flats.
      if R(i,j)>highTH
         Corners_HCD(i,j) = I_gray(i,j);
      elseif R(i,j)<lowTH
         Edges_HCD(i,j) = I_gray(i,j);
      else
         Flats_HCD(i,j) = I_gray(i,j);
      end

   end
end


function [Es, Eo]  = canny_enhancer(I)
% function CANNY_ENHANCER used to compute the gradient information
```

EECE5639 Computer Vision Project 4 Report

```
% Input:
%   I: input image
%   G: zero mean Gaussian filter (std = ?)
% Output:
%   Es: Estimate edge strength
%   Eo: Estimate edge orientation
%



% Check the dimensions of the image. number of Color Bands should be = 1.
[row, col, band] = size(I);
if band > 1
   % It's not gray scale Image and convert it to gray scale Image
   I_gray = rgb2gray(I);
elseif band == 1
   % It's grayscale Image and copies it
   I_gray = I;
end

gfilter = gradient(fspecial('gaussian',[1,3],2));       %Gaussian filter

J = imgaussfilt(I_gray, 2);

[Es,Eo] = imgradient(J,'prewitt');



function [I_N]  = nonMax_Supression(Es, Eo)
% Compute NONMAX Supression to get a sparse set of input features.
% Input as The inputs are Es and Eo (outputs of CANNY_ENHANCER)
% Consider 4 directions D={ 0,45,90,135} with respect to (wrt) x
%   Es: Estimate edge strength: Es(i,j) = (Jx2(i,j)+ Jy2(i,j))1/2
%   Eo: Estimate edge orientation: Eo(i,j) = arctan(Jx(i,j)/Jy(i,j))
% Output is the thinned edge image I_N
%   I_N: the thinned edge image
%

% Assume Es and Eo have same size.
[row, col]=size(Es);
marg=9;   % margin, MUST be odd number;
win_mask=zeros(marg); % 1;
win_mask(ceil(marg/2),ceil(marg/2))=1;
```

EECE5639 Computer Vision Project 4 Report

```
%pad_I=conv2(Es,[0,0,0;0,1,0;0,0,0]);
pad_I=conv2(Es,win_mask);
I_N = zeros(row, col);

if marg == 1
% For each pixel (i,j), if {Es(i,j) is smaller than at least one of its
% neighor along d, then IN(i,j)=0, Otherwise, IN(i,j)= Es(i,j)
   for i = 1:row
      for j = 1:col
         if i-marg<1 && j-marg<1
            % Check UPPER LEFT corner Start
            if  Es(i,j) < Es(i+marg,j)
   %             break;
            elseif Es(i,j) < Es(i,j+marg)
   %             break;
            elseif Es(i,j) < Es(i+marg,j+marg)
   %             break;
            else
               I_N(i,j) = Es(i,j);
            end
            % Check UPPER LEFT corner End
         elseif i-marg<1 && j+marg>col
            % Check UPPER RIGHT corner Start
            if Es(i,j) < Es(i,j-marg)
   %             break;
            elseif Es(i,j) < Es(i+marg,j-marg)
   %             break;
            elseif Es(i,j) < Es(i+marg,j)
   %             break;
            else
               I_N(i,j) = Es(i,j);
            end
            % Check UPPER RIGHT corner End
         elseif i+marg>row && j-marg<1
            % Check LOWER LEFT corner Start
            if Es(i,j) < Es(i-marg,j)
   %             break;
            elseif Es(i,j) < Es(i-marg,j+marg)
   %             break;
            elseif Es(i,j) < Es(i,j+marg)
   %             break;
            else
```

```
            I_N(i,j) = Es(i,j);
        end
        % Check LOWER LEFT corner End
    elseif i+marg>row && j+marg>col
        % Check LOWER RIGHT corner Start
        if Es(i,j) < Es(i-marg,j-marg)
%             break;
        elseif Es(i,j) < Es(i,j-marg)
%             break;
        elseif Es(i,j) < Es(i-marg,j)
%             break;
        else
            I_N(i,j) = Es(i,j);
        end
        % Check LOWER RIGHT corner End
    elseif i-marg<1
        % Check UPPER side Start
        if Es(i,j) < Es(i,j-marg)
%             break;
        elseif Es(i,j) < Es(i+marg,j-marg)
%             break;
        elseif Es(i,j) < Es(i+marg,j)
%             break;
        elseif Es(i,j) < Es(i,j+marg)
%             break;
        elseif Es(i,j) < Es(i+marg,j+marg)
%             break;
        else
            I_N(i,j) = Es(i,j);
        end
        % Check UPPER side End
    elseif j-marg<1
        % Check LEFT side Start
        if Es(i,j) < Es(i-marg,j)
%             break;
        elseif Es(i,j) < Es(i+1,j)
%             break;
        elseif Es(i,j) < Es(i-marg,j+marg)
%             break;
        elseif Es(i,j) < Es(i,j+marg)
%             break;
        elseif Es(i,j) < Es(i+marg,j+marg)
```

```
%             break;
          else
             I_N(i,j) = Es(i,j);
          end
          % Check LEFT side End
       elseif j+marg>col
          % Check RIGHT side Start
          if Es(i,j) < Es(i-marg,j-marg)
%             break;
          elseif Es(i,j) < Es(i,j-marg)
%             break;
          elseif Es(i,j) < Es(i+marg,j-marg)
%             break;
          elseif Es(i,j) < Es(i-marg,j)
%             break;
          elseif Es(i,j) < Es(i+marg,j)
%             break;
          else
             I_N(i,j) = Es(i,j);
          end
          % Check RIGHT side End
       elseif i+marg>row
          % Check BOTTOM side Start
          if Es(i,j) < Es(i-marg,j-marg)
%             break;
          elseif Es(i,j) < Es(i,j-marg)
%             break;
          elseif Es(i,j) < Es(i-marg,j)
%             break;
          elseif Es(i,j) < Es(i-marg,j+marg)
%             break;
          elseif Es(i,j) < Es(i,j+marg)
%             break;
          else
             I_N(i,j) = Es(i,j);
          end
          % Check BOTTOM side End
       else
          % Check every direction Start
          if Es(i,j) < Es(i-marg,j-marg)
%             break;
          elseif Es(i,j) < Es(i,j-marg)
```

```
    %               break;
            elseif Es(i,j) < Es(i+marg,j-marg)
    %               break;
            elseif Es(i,j) < Es(i-marg,j)
    %               break;
            elseif Es(i,j) < Es(i+marg,j)
    %               break;
            elseif Es(i,j) < Es(i-marg,j+marg)
    %               break;
            elseif Es(i,j) < Es(i,j+marg)
    %               break;
            elseif Es(i,j) < Es(i+marg,j+marg)
    %               break;
            else
                I_N(i,j) = Es(i,j);
            end
            % Check every directions End
        end
    end
end

elseif mod(marg,2)==1
    for i=1:row
        for j=1:col
            win_mask=pad_I(i:i+marg-1, j:j+marg-1);     %reuse for temp
            win_mask(ceil(marg/2),ceil(marg/2))=0;      %zero middle cell
            if Es(i,j) > max(win_mask,[],'all')         %find only the max
                I_N(i,j)=Es(i,j);
            end

        end
    end

else
    disp('error, marg should be an odd number.');
end



function [img_sub, img_dis]  = extractPatches(I,P)
% function extractPatches extract a fixed size image patch (25 × 25 pixels)
% and use the vector of raw pixel intensities as the "descriptor".
% Input:
```

EECE5639 Computer Vision Project 4 Report

```
%   I: input image
%   P: interest points
% Output:
%   img_sub: cropped image in vectorization
%   img_dis: cropped image center pixel location
%

% this has to be an odd x odd pixels, interested point at the center
pSize = 25; % 25 x 25 pixels
[rId, cId] = find(P);
marg=floor(pSize/2);
img_count=0;
img_sub={};
img_dis={};

% Check the dimensions of the image. The number of Color Bands should be = 1.
% This section can comment out if we know the image is grayscale
[row, col, band] = size(I);
if band > 1
   % It's not gray scale Image and convert it to gray scale
   I_gray = double(rgb2gray(I));
elseif band == 1
   % It's gray scale Image
   I_gray = double(I);
end

% Output the patches around interest points, ignore the boundary ones
for i = 1:length(rId)
   if (rId(i) > marg) && (rId(i) < row-marg)...
         &&(cId(i) > marg) && (cId(i) < col-marg)

      img_count=img_count+1;
      % Vectorization patch into n x p data matrix
      img_pat=I_gray(rId(i)-marg:rId(i)+marg,cId(i)-marg:cId(i)+marg);
      img_loc=[rId(i),cId(i)];
      %imshow(img_pat)
      % "img_pat(:)" turns image into 625x1 vector
      img_sub{end+1} = img_pat(:);
      img_dis{end+1} = img_loc(:);
   end

end
```

```
img_sub=[img_sub{:}];
img_dis=[img_dis{:}];
```

harris by Lin:

```
function [varargout] = harris(I,varargin)

param = checkargs(size(I),varargin{:});

I = double(I);

[nr,nc] = size(I);

% create gradient masks
dx = [-1 0 1; -1 0 1; -1 0 1]/3;
dy = dx';

% calculate image gradients
Ix = convolve2(I,dx,'same');
Iy = convolve2(I,dy,'same');

% calculate gradient products
Ixlx = Ix.*Ix;
Iyly = Iy.*Iy;
Ixly = Ix.*Iy;

% smooth squared image gradients
gmask = fspecial('gaussian',param.hsize,param.sigma);
if param.fft == false
  Ixlx = convolve2(Ixlx,gmask,'same');
  Iyly = convolve2(Iyly,gmask,'same');
  Ixly = convolve2(Ixly,gmask,'same');
else
  m = size(Ixlx,1)+size(gmask,1)-1;
  n = size(Ixlx,2)+size(gmask,2)-1;
  try
    % previous values
    pv_t = load(strcat(tempdir,'harris_prev_val.mat'));
    if m == pv_t.m && n == pv_t.n && ...
            param.hsize == pv_t.hsize && param.sigma == pv_t.sigma
```

```
    % load previously stored smoothing filter fft
    %disp('loading gmask_fft');
    gmask_fft = pv_t.gmask_fft;
  else
    %disp('generating gmask_fft');
    gmask_fft = fft2(gmask,m,n);
  end
 catch
  %disp('no previous values exist, generating gmask_fft')
  hsize = param.hsize;
  sigma = param.sigma;
  % no previous values exist
  gmask_fft = fft2(gmask,m,n);
  save(strcat(tempdir,'harris_prev_val.mat'), ...
       'hsize','sigma','m','n','gmask_fft');
 end

 Ixlx = real(ifft2(fft2(Ixlx,m,n).*gmask_fft));
 Iyly = real(ifft2(fft2(Iyly,m,n).*gmask_fft));
 Ixly = real(ifft2(fft2(Ixly,m,n).*gmask_fft));
 % keep 'same' portion
 w = (param.hsize-1)/2; % hsize is assumed to be odd
 Ixlx = Ixlx(w+1:w+nr,w+1:w+nc);
 Iyly = Iyly(w+1:w+nr,w+1:w+nc);
 Ixly = Ixly(w+1:w+nr,w+1:w+nc);
 clear hsize sigma m n gmask_fft gmask;
end

% calculate the eigenvalues of the matrix
% [Ixlx Ixly
%  Ixly Iyly]
% old way
%lambda1a = (Ixlx+Iyly)/2 + 1/2*sqrt((Ixlx+Iyly).^2-4*(Ixlx.*Iyly-Ixly.*Ixly));
%lambda2a = (Ixlx+Iyly)/2 - 1/2*sqrt((Ixlx+Iyly).^2-4*(Ixlx.*Iyly-Ixly.*Ixly));
% faster way
B = (Ixlx+Iyly);
SQRTRM = sqrt(B.^2 - 4*(Ixlx.*Iyly-Ixly.^2));
lambda1 = (B+SQRTRM)/2;
lambda2 = (B-SQRTRM)/2;
clear B SQRTRM Ixlx Iyly Ixly;

% corner detection is based upon the desired corner response function
```

EECE5639 Computer Vision Project 4 Report

```
if param.eig
  % minimum eigenvalue
  R = real(min(lambda1,lambda2));
else
  % Harris corner response function
  % trace equals sum of eigenvalues
  % determinant equals product of eigenvalues
  R = lambda1.*lambda2 - 0.04*(lambda1+lambda2).^2;
end

% locate local maxima based upon eight-connected neighborhood
Maxima = (imregionalmax(R) & param.mask).*R;

% sort interest points by reponse function
% only consider pixels where the gradient matrix is positive definite
% i.e. both eigenvalues are greater than zero
[i,j] = find(Maxima > param.thresh);
m = Maxima((j-1)*nr+i); %shortcut for maxima(sub2ind(size(maxima),i,j));
[m,idx] = sort(m);
i = i(idx);
j = j(idx);

% interest points were sorted ascendingly by response function,
% flip so largest response points are first
m = m(end:-1:1);
i = i(end:-1:1);
j = j(end:-1:1);

if param.tile(1) > 1 && param.tile(2) > 1
  % process image regionally so that corners are uniformally
  % extracted across image regions
  ii = [];
  jj = [];
  mm = [];
  Npts_per_region = round(param.N/prod(param.tile));
  xx = round(linspace(1,nc,param.tile(2)+1)); % region boundaries
  yy = round(linspace(1,nr,param.tile(1)+1));
  for pp = 2:length(xx)
    % points falling within the region's x boundaries
    idx = find((j >= xx(pp-1)) & (j < xx(pp)));
    for qq = 2:length(yy)
      % points falling within the region's y boundaries
```

```
      idy = find((i >= yy(qq-1)) & (i < yy(qq)));

      % their common intersection
      ind = intersect(idx,idy);

      % return the strongest N points as defined by user
      ii = [ii; i(ind(1:min(end,Npts_per_region)))];
      jj = [jj; j(ind(1:min(end,Npts_per_region)))];
      mm = [mm; m(ind(1:min(end,Npts_per_region)))];
    end
  end
else
  ii = i(1:min(end,param.N));
  jj = j(1:min(end,param.N));
  mm = m(1:min(end,param.N));
end

if param.subpixel
  % refine corner locations with subpixel accuracy
  % fit a quadratic surface to maxima location
  % and calculate it's analytic peak
  [ii,jj] = subpixel(ii,jj,R);
end

if param.disp || nargout == 0
  % overlay corner points on original image
%   figure;
%   imagesc(I);
%   colormap gray;
%   hold on;
%   plot(jj,ii,'y+');
%   hold off;
%   drawnow;
end

if nargout >= 2
  varargout{1} = ii;
  varargout{2} = jj;
end
if nargout == 3
  varargout{3} = mm;
end
```

EECE5639 Computer Vision Project 4 Report

```
%=====================================================================
function param = checkargs(isize,varargin)

% set defaults
param.disp = 0;      % overlay corner points on image
param.N = 500;       % number of interest points to return
param.subpixel = false;  % refine corner location with subpixel accuracy
param.thresh = 0;    % threshold value for smallest response magnitude
param.hsize = 3;     % size of gaussian smoothing mask
param.sigma = 0.5;   % standard deviation of gaussian mask
param.eig   = false; % use corner response function as originally
                     % proposed by Harris
param.tile = [1 1];  % do not process image regionally
param.fft  = false;  % implement conv2 in spatial domain

%---------------------------------------------
% replace defaults with user specified values
%---------------------------------------------
% check to see if number of corner points was specified
% otherwise use default
if nargin > 1 && isnumeric(varargin{1})
  param.N = varargin{1};
  ii = 2;
else
  ii = 1;
end

% loop through parameter/value pairs
while ii <= nargin - 1
  switch lower(varargin{ii})
   case 'disp'
    param.disp = 1;
    ii = ii+1;
   case 'subpixel'
    param.subpixel = true;
    ii = ii+1;
   case 'eig'
    param.eig = true;
    ii = ii+1;
   case 'thresh'
    param.thresh = varargin{ii+1};
```

```
    ii = ii+2;
   case 'hsize'
    param.hsize = varargin{ii+1};
    ii = ii+2;
   case 'sigma'
    param.sigma = varargin{ii+1};
    ii = ii+2;
   case 'mask'
    param.mask  = logical(varargin{ii+1});
    ii = ii+2;
   case 'tile'
    param.tile  = varargin{ii+1};
    ii = ii+2;
   case 'fft'
    param.fft = true;
    ii = ii+1;
   otherwise
    error(sprintf('Uknown option "%s"',varargin{ii}));
  end
end

if ~isfield(param,'mask')
  % corner mask defines region to compute interest points
  % define default mask to include "valid" smoothed
  % image gradient portions from convolution
  param.mask = true(isize);
  %h = ceil((param.hsize-1)/2);
  %param.mask(1:h,:) = false;
  %param.mask(end-h-1:end,:) = false;
  %param.mask(:,1:h) = false;
  %param.mask(:,end-h-1:end) = false;
end

%=============================================================
% the current implementation no longer uses this function
function mask = regionalmask(tile,msize,ii,jj)

x = 1:floor((msize(2)-1)/tile(2)):msize(2);
y = 1:floor((msize(1)-1)/tile(1)):msize(1);

mask = false(msize);
mask(y(jj):y(jj+1)-1,x(ii):x(ii+1)-1) = true;
```

EECE5639 Computer Vision Project 4 Report

```
%=========================================================================
function [isub,jsub] = subpixel(i,j,R)

% fit a quadratic surface to each integer maxima
% location defined by [i,j] using its surrounding
% eight neighbors.
%
% a*i^2 + b*i*j + c*j^2 + d*i + e*j+f = R(i,j)
%
% write above equation in matrix format
% [i^2 i*j j^2 i j 1][a b c d e f]' = R(i,j)
%
% ALGORITHM OUTLINE:
%-------------------------------------------------------
% stack equations for all 9 pixels PER maxima
% i.e. A*pvec = B
% solve using least squares for one 6x1 parameter
% vector which defines a quadratic surface for maxima
% i.e. pvec = B\A where pvec = [a b c d e f]'
%-------------------------------------------------------

% number of feature points
N = length(i);

% location of quadratic surface maximum as expressed
% in local coordinates
jmax = zeros(N,1);
imax = zeros(N,1);

% [p,q] are the local pixel coordinates centered around
% feature point [i,j]
p = [-1  0  1 -1 0 1 -1 0 1]';
q = [-1 -1 -1  0 0 0  1 1 1]';
A = [p.^2, p.*q, q.^2, p, q, ones(9,1)];
B = zeros(9,1);

for n = 1:N
  % calculate measurement vector
  for k=1:9
    B(k) = R(i(n)+p(k),j(n)+q(k));
```

```
  end

  % calculate least-squares parameter vector
  pvec = B\A;
  a = pvec(1); b = pvec(2); c = pvec(3);
  d = pvec(4); e = pvec(5); f = pvec(6);

  % analytically calculate location of surface maxima
  % by solving the two equations listed below
  % d/di = 0 = 2*a*i + b*j + d
  % d/dj = 0 = b*i + 2*c*j + e
  jmax(n) = (2*a*e-d*b)/(b^2-4*a*c);
  imax(n) = -(2*c*jmax(n)+e)/b;
end

% update maxima locations with subpixel precision
isub = i+imax;
jsub = j+jmax;
```

Other method extract the patch:

```
function patches = extractPatch_v2(ImageA,corners,Size)

[height, width] = size(ImageA);
r = (Size-1)/2;
x = corners(:,1);
y = corners(:,2);
corners_c = length(corners(:,1));

for i = 1:corners_c
    if x > r & x <= width-r & y > r & y < height-r
        patches(:,:,i) = ImageA(x-r:x+r,y-r:y+r);
    end
end
```