cs512 Assignment 2: Report

Yash Patel

Department of Computer Science

Illinois Institute of Technology

October 20, 2020

Abstract

This is a report of programming assignment 2 which covers Harris corner detection, HOG feature vectors extraction and compare HOG features in given images without using OpenCV directly.

1. Problem statement

Load and display two images containing similar content. In each image perform the following:

- Apply the Harris corner detection algorithm.
- Obtain a better localization of each corner.
- Computer a feature vector for each corner.
- Display the corner by drawing empty colored rectangles over the original colored image centered at location where corners were detected.

Using the feature vectors computed in both images match the feature points from one image to another image. Number corresponding points with identical numbers in the two images.

Interactively controlled parameters should include:

- The variance of the gaussian noise.
- The variance of the gaussian smoothing.
- The neighborhood/window size for computing the correlation matrix.
- The weight of the trace in the Harris corner detector.
- A threshold value.

Evaluate the performance of the algorithm using test data with added noise or occlusions.

2. Proposed solution

- **Help key for functionality description:**
    1) ‘h’: pressing this key will print help description.
    2) ‘i’: pressing this key will load input images into this application.
    3) ‘s’: pressing this key will save both processed images.
    4) ‘c’: pressing this key will apply corner detection in both images without using OpenCV functions directly.
    5) ‘C’: pressing this key will apply corner detection in both images with using OpenCV functions directly.
    6) ‘e’/<ESC>: pressing this key will close the application.
- **Controlled parameters:**
    1) Here two separate windows are used to compare outputs of both processed images.
    2) Each window has five separate trackbars to control required parameters.
        - The range of the noise variance is 0 to 100 and the value will be divided by 100.
        - Gaussian smoothing amount used is 5 and the range of the variance is 0 to 100 and the value will be divided by 100.
        - The range of the window size is 0 to 9. But, applied window size is (2 * window size) + 1.
        - The range of the weight of trace (k) is 0 to 50 and the value will be divided by 100.
        - The range of the threshold is 0 to 1000 and the value will be multiplied by 10000.
- **Pre-processing:**
    1) Add gaussian noise into both images.
    2) Convert color images into grayscale images.
    3) Apply gaussian smoothing on images for reducing noise.
- **Harris corner detection:**
    1) Find gradients using x and y derivatives.
    2) Scan image top-to-bottom and left-to-right.
    3) Take each pixel as a center of neighborhood in a local window and find correlation matrix.
    4) Find cornerness of each pixel.
    5) Compare cornerness with threshold and store coordinates, cornerness of pixel if there is a corner.
- **Non maximum suppression:**
    1) After detecting pixels with corner in local neighborhood, sort all the coordinates of pixels in decreasing order on the basis of cornerness of that pixel.

2) Iterate sorted pixels and select if there is no pixel in its local neighborhood in the new list of pixels.

- **Localization of corner:**
    1) Compute required matrices to find better localization (P*) of each corner in every window those are selected after non maximum suppression.
    2) To determine if P is the center connect each point xi to P and project the gradient at xi onto (xi-P).
    3) The best P will minimize the scan of all projections.
    4) Find P* for each window which is a localized point of corner in a local window.
- **Extract HOG features:**
    1) To calculate HOG features at every pixel where corner is detected, create patch of size 18x18. In patch create non-overlapping cells of size 6x6. Calculate feature for each cell.
    2) To calculate HOG features for every cell, use bin size of 8 for feature array that represent angle 45. At each pixel find strength, angle and bin number of gradient. Update value of feature array at index of bin number using strength and angel.
    3) Shift the feature array as highest value of histogram comes first in feature array.
    4) Store all features of one patch in an array of size 3x3.
- **Compare HOG features:**
    1) For comparison, find sum of Euclidean distance between features of each patch in first image with features of each patch in second image.
    2) Match every patch in first image with a patch in second image with minimum total distance.
- **Draw rectangles and numbers on images:**
    1) Draw rectangles at each corner points with extra pixel padding of 5 on every side of rectangle on both images using OpenCV’s “rectangle” method.
    2) Find unique identity number for each matching by creating dictionary of (key, value) = (features of patch in first image, list of features of patches in second image).
    3) Draw corners’ unique identity number at each corner points on both images using OpenCV’s “putText” method.

3. Implementation details

- **Design issues:**
    - The one design issue I got was that showing rectangles at the corner with different window size. I took rectangle size as window size and also added

some pixel padding for making rectangles with small window size looks large.

- **Problems faced and solutions:**
    - The main problem was comparing my output with OpenCV's output. Detecting corners in images with OpenCV was not tough but the tough part was creating feature vectors for both images and compare with each other. Also, I got the feature vectors but it was hard to understand the dimensions of vectors and to compare with another image's feature vectors.
    - Another problem that I faced was showing unique number for each feature vectors comparison when more than one feature vectors in one image match with single feature vector in another image. For this problem I used dictionary data structure that save feature vector from one image as key and list of matched feature vectors from another image as value.
- **Instructions for using program:**
    - Put all source code from src and image files from data together.
    - Choose two images with similar content and rename them with "in1.jpg" and "in2.jpg".
    - As first image choose normal and as second image choose rotated, scaled, translated or illuminated.
    - Execute Homework-2.ipynb file in Jupyter notebook.
    - First it will show help.
    - Before applying any processing on the images, load these input images using 'i' key. It will initialize the input images.
    - After loading these input images, you can apply save or corner detection on the images.

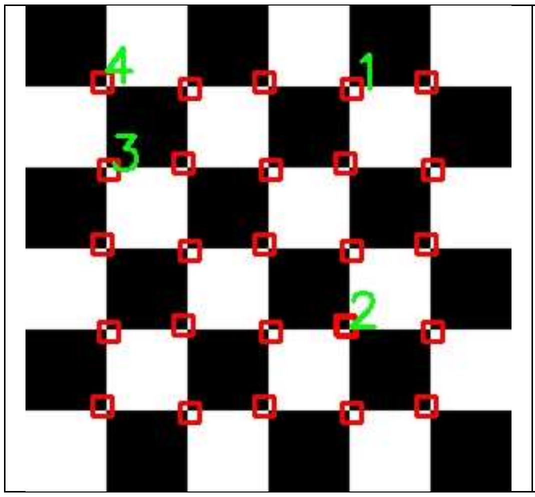
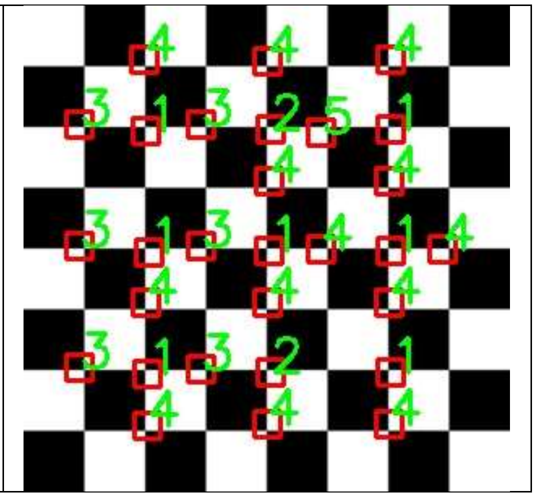
4. Results and Discussion

Here, I test my program on two different king of images and applied different values of parameters. I am showing the best results and its parameters that I got.

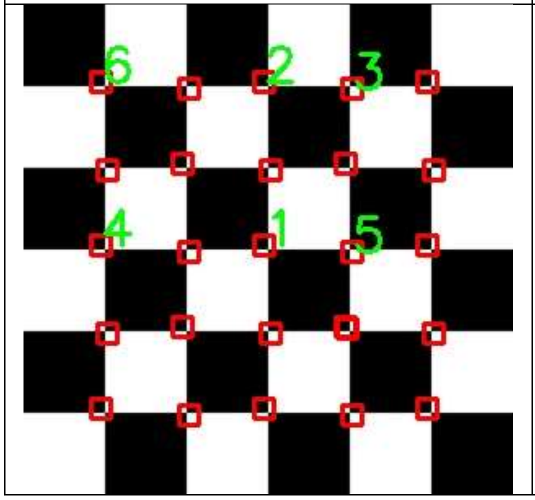
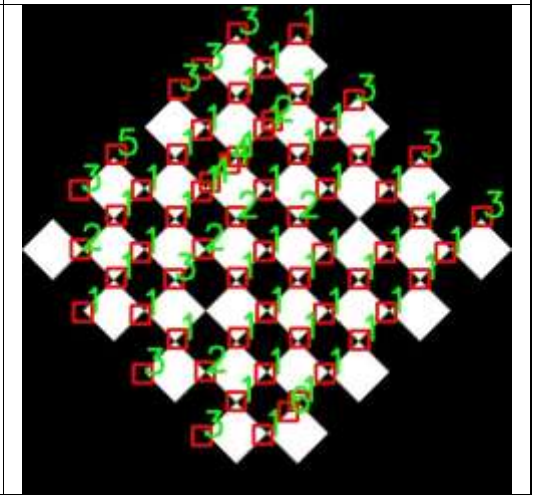
I. Test-1

here, I chose one image of chess board with 6x6 grid and second image of chess board with 8x8 grid.

Noise : 0/100 : 0 Gaussian variance : 10/100 : 0.1 Window size : 1 = (2*1)+1 = 3 K : 0/100 : 0 Threshold : 21*10000 = 210000	Noise : 0/100 : 0 Gaussian variance : 10/100 : 0.1 Window size : 3 = (2*3)+1 = 7 K : 0/100 : 0 Threshold : 56*10000 : 560000

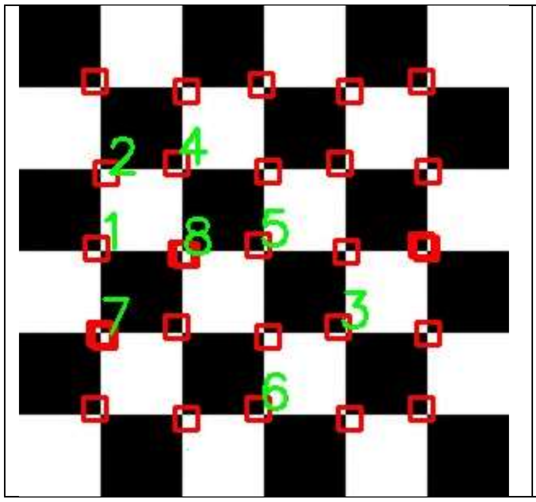
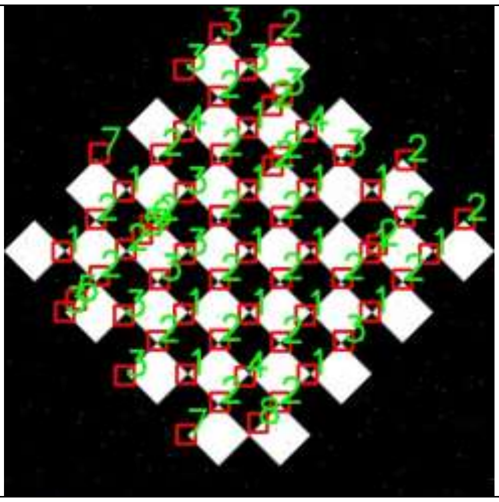
	

here, I chose one image of chess board with 6x6 grid and second image of chess board with 8x8 grid but rotated with 45 degree.

Noise : 0/100 : 0 Gaussian variance : 10/100 : 0.1 Window size : 1 = (2*1)+1 = 3 K : 0/100 : 0 Threshold : 21*10000 = 210000	Noise : 0/100 : 0 Gaussian variance : 75/100 : 0.75 Window size : 3 = (2*3)+1 = 7 K : 4/100 : 0.04 Threshold : 199*10000 = 1990000
	

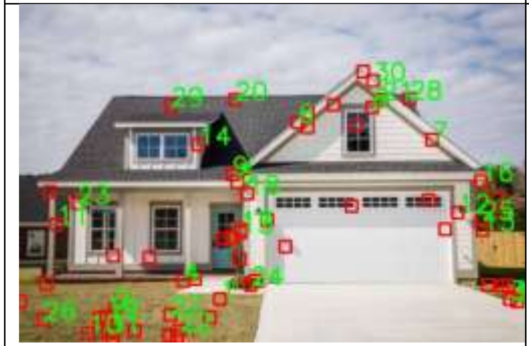
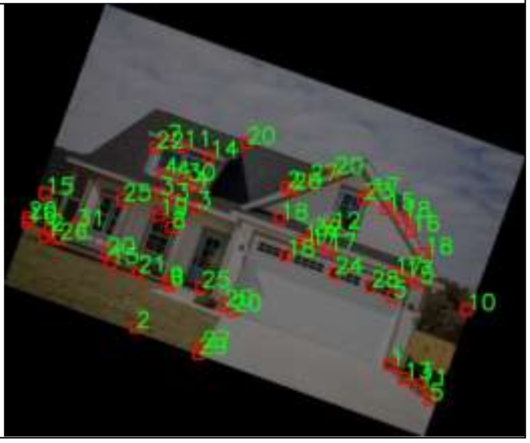
here, I chose one image of chess board with 6x6 grid and second image of chess board with 8x8 grid but rotated with 45 degree and noise in both images.

Noise : 5/100 : 0.05 Gaussian variance : 25/100 : 0.25 Window size : 2 = (2*2)+1 = 5 K : 4/100 : 0.04 Threshold : 21*10000 = 210000	Noise: 10/100 : 0.1 Gaussian variance : 75/100 : 0.75 Window size : 3 = (2*3)+1 = 7 K : 4/100 : 0.04 Threshold : 225*10000 = 2250000

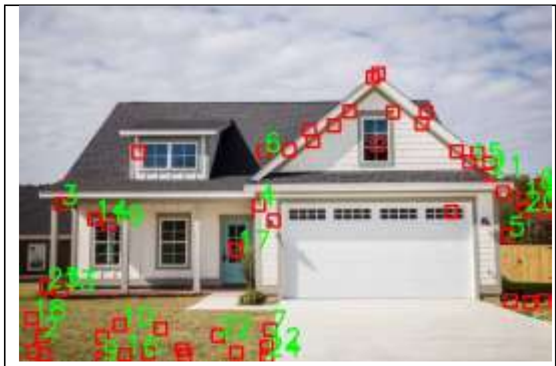
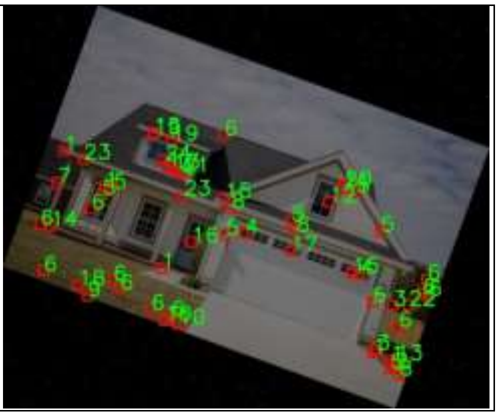
II. **Test-2**

here, I chose one image of house and second image of same house but rotated with 20 degree and changed brightness.

Noise : 0/100 : 0 Gaussian variance : 25/100 : 0.25 Window size : 2 = (2*2)+1 = 5 K : 4/100 : 0.04 Threshold : 220*10000 = 2200000	Noise : 0/100 : 0 Gaussian variance : 30/100 : 0.30 Window size : 2 = (2*2)+1 = 5 K : 4/100 : 0.04 Threshold : 241*10000 = 2410000
	

here, I chose one image of house and second image of same house but rotated with 20 degree and changed brightness and noise in both images.

Noise : 5/100 : 0.05 Gaussian variance : 25/100 : 0.25 Window size : 2 = (2*2)+1 = 5 K : 4/100 : 0.04 Threshold : 200*10000 = 2000000	Noise : 10/100 : 0.10 Gaussian variance : 30/100 : 0.30 Window size : 2 = (2*2)+1 = 5 K : 4/100 : 0.04 Threshold : 230*10000 = 2300000

After analyzing these output images, I found the following points:

- As noise increases, I need to apply gaussian smoothing with higher gaussian variance.
- As noise increases, I need to apply higher window size with larger neighborhood to detect corners in the image.
- As gaussian smoothing increases, program detects windows with corner more accurately. But too much smoothing decreases number of detected windows with corner.
- As window size increases, I need to apply higher threshold to remove windows with small cornerness. Because as window size increases, cornerness also increases.
- As noise increases, I need to apply higher threshold to remove windows with small cornerness. Because as noise increases, cornerness also increases.
- As noise increases, more windows get detected as corners even that window doesn’t have corner in it.
- As value of weight on trace (k) increases, more windows get detected which has edges but not corner in it.
- During test on the images with chess board, detection of the corners was mostly accurate and similar to OpenCV’s detection and most of the feature vectors in second image were matched with few feature vectors in first image.
- During test on the images with house, detection of the corners was less accurate and very less similar to OpenCV’s detection and most of the feature vectors in second image were matched with few feature vectors in first image and also matched windows with corners in both images were not same even content in second images were not changed much.