cs512 Assignment 2

Chandni Patel
Department of Computer Science
Illinois Institute of Technology

October 20, 2020

Abstract

This report contains description of the corner detection problem, the algorithms used for solving the problem, and analysis of the results obtained for the programming part of Assignment 2.

1 Problem Statement

The goal is to write a program to perform Corner detection with following requirements:

1. **Corner Detection**: Load and display 2 images for corner detection. In both images:
    - Convert image to grayscale, estimate image gradient, and apply Harris corner detection algorithm without OpenCV.
    - Obtain a better localization of each corner.
    - Compute feature vectors for each corner point.
    - Display corners by drawing empty rectangles (in color) over the original image (grayscale) where corners are detected.
2. **Feature Comparison**: Using feature vectors, match feature points of both the images and number corresponding points with identical numbers in 2 images.
3. **Gaussian Noise and Smoothing**: Add gaussian noise to create test data and apply gaussian smoothing to evaluate performance of the algorithm.
4. **Interactively controlled parameters**:
    - Variance for gaussian (scale)
    - Smoothing value for gaussian smoothing (filter size)
    - Neighborhood size for computing the correlation matrix
    - Weight of the trace in the Harris corner detection
    - Threshold value

2 Proposed Solution

Solution for this programming assignment is created using Python with OpenCV. Major algorithms are implemented without using OpenCV functions to achieve various requirements like Harris corner detection, non-maximum suppression, corner localization, feature characterization, and comparing features.

Added the following **Support Keys** to apply main functions:

- 1 – Apply Harris Corner Detection on both images
- 2 – Compare Features of both images
- 3 – OpenCV corner detection for verification
- h – Display help
- i – Reload the original images
- w – Save the current images

3 Implementation Details

All interactive parameters can be changed using the trackbars. Once the trackbar values are set, press command key to see the results created with the updated trackbar values. The trackbars are not applied on change event, because when the trackbar values were applied simultaneously, there was a lot of lag and had to restart the terminal.

Corner Detection

For corner detection, detectCorners() function mainly plots rectangles (with dimensions of the window size selected) around the final list of corners which is obtained after implementing the following:

1. **Estimate Image Gradients:** Implemented getGradient() function to estimate gradient x and y for the image.

2. **Gaussian Noise and Smoothing:**

   I. Implemented addNoise() function to add gaussian noise in the image using variance from the trackbar.

II. Implemented applySmoothing() function for gaussian smoothing to reduce noise in the image taking filter size from the smoothing trackbar.

3. **Harris Corner Detection:** Implemented harrisCorners() function to apply Harris corner detection algorithm without using OpenCV.

   I. It scans image top to bottom, left to right at each pixel and selects a local neighborhood with fixed window size set in the trackbar.

   II. It builds correlation matrix for the window and computes cornerness measure using trace weight k from the trackbar.

      - $G(C) = \det(C) - k \cdot Tr^2(C)$

   III. Corner is detected if that value is higher than the threshold set in the trackbar.

4. **Non-maximum Supression:** Implemented suppressCorners() function to apply non-maximum suppression without using OpenCV.

   I. It sorts the corner point in reverse order using cornerness measure.

   II. It selects top of the list as corner and removes all other corners in its neighborhood from the list.

   III. If Euclidean distance between the points is less than or equal to the window size, then it is considered in the neighborhood.

5. **Corner Localization:** Implemented localizeCorners() function to locate exact corner location in the given window without using OpenCV.

   I. To determine if P is the corner, connect each point x_i to P and project the gradient at x_i onto $x_i - P$.

      - $P^* = C^{-1} \Sigma \nabla I(x_i) \nabla I(x_i)^T x_i$

   II. The best P will minimize the sum of all projections.

6. **Extract Feature Vectors:** Implemented featureVectors() function to create feature vectors of all the final corner points.

I. It splits each patch into cells (Not taking overlapping cells).
II. Creates orientation histogram (HOG) in each cell using gradient direction and gradient magnitude.
III. Rotates histogram so that the dominant direction is at start.
IV. Adds HOG of each cell into a matrix to create final feature vector.
V. Takes patch size = 27, cell size = 9 and bin size = 8 (because 360/45 degrees) by default.

Feature Comparison

First, for each image, get final list of corners, feature vectors of those corner and image with plotted corners using the detectCorners() function explained above. Then use implemented compareFeature() function to compare the features in each image.

I. Calculates distance from each feature in first image to all features in second image.
II. The feature in second image which is closest to the feature in the first image is mapped to that feature.
III. Creates a mapped list of feature points that match in both images.
IV. Inserts an identical number at corresponding points in both images.

OpenCV Corner Detection

This is implemented only for verification and evaluation purposes. For corner detection using OpenCV, cv2.cornerHarris() function is used and gray scale image, window size, and trace weight k are passed as parameter. For non-maximum suppression, top 1% of corners are detected and then plotted on the image.

Issues

- Corner detection and feature comparison is too slow when the number of corners increases. I would like to code it in Cython to increase the speed but due to lack of time, I could not recode.
- Also, range of threshold value is huge as different values work for different images.

Instructions to run the program

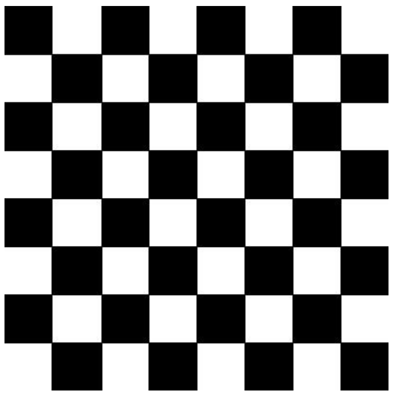
I. I have added default input image names in the code, which can be modified in the last line of code.

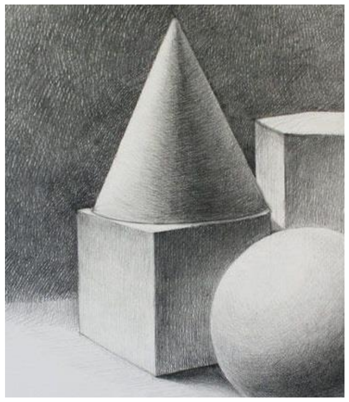
II. When running the programs, the Jupyter notebook and the images will have to be in the same folder location.

4 Results and Discussion

Tried multiple combinations of k, window size and threshold to get different solutions. Here are the optimal results obtained using the multiple images for testing and evaluation of the algorithm implemented:

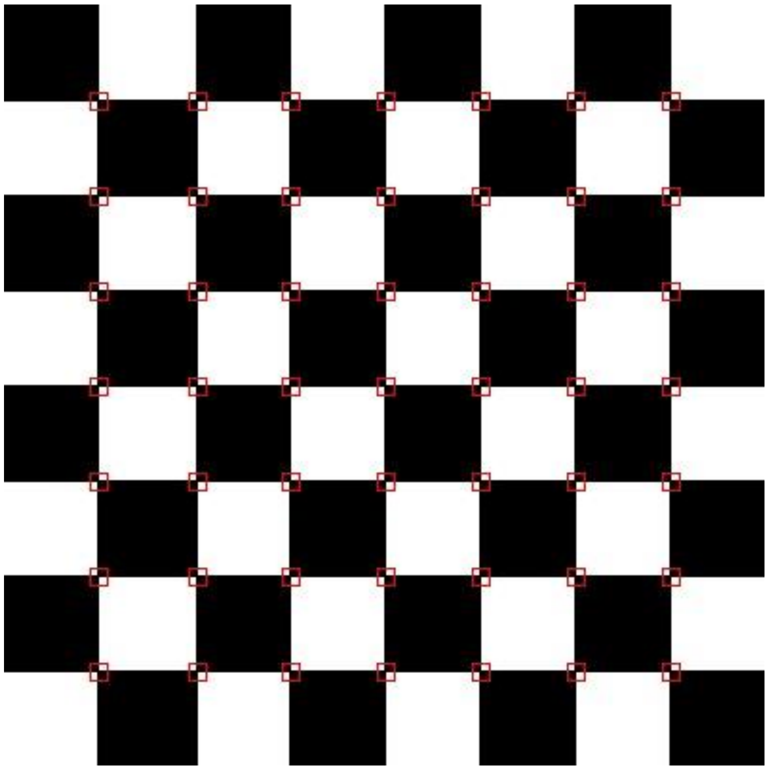
1. **Original Images:** Using a simple chessboard image to do initial testing. More complicated images used are: Building images that has many corners and Object sketch has a lot of noise.





2. Corner detection with implemented functions:

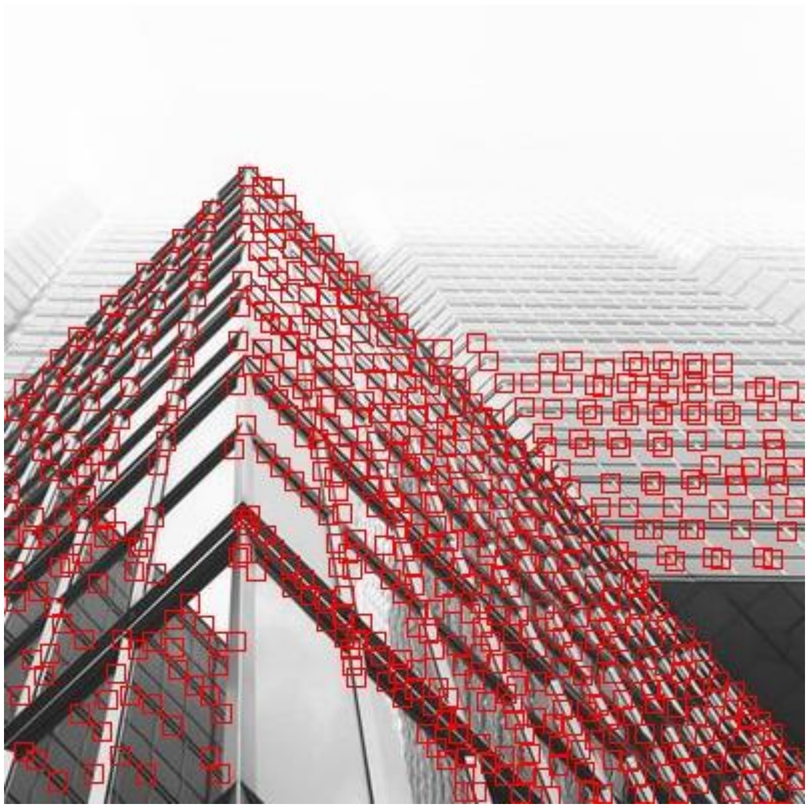
Simple corner detection in different images with best parameters.



k = 0.04

Window size = 9

Threshold = 500*100000000



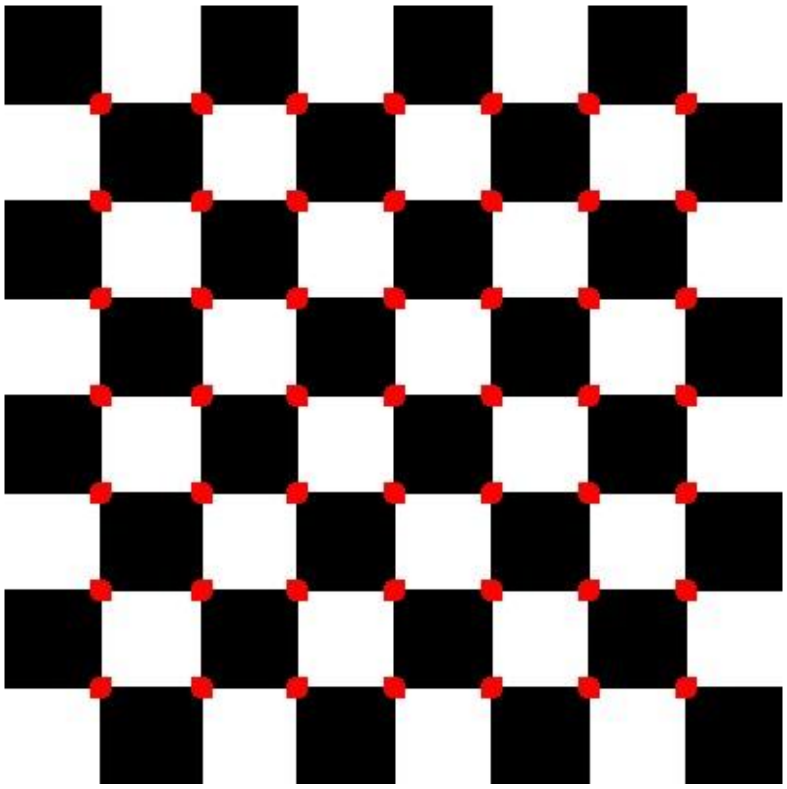
k = 0.04

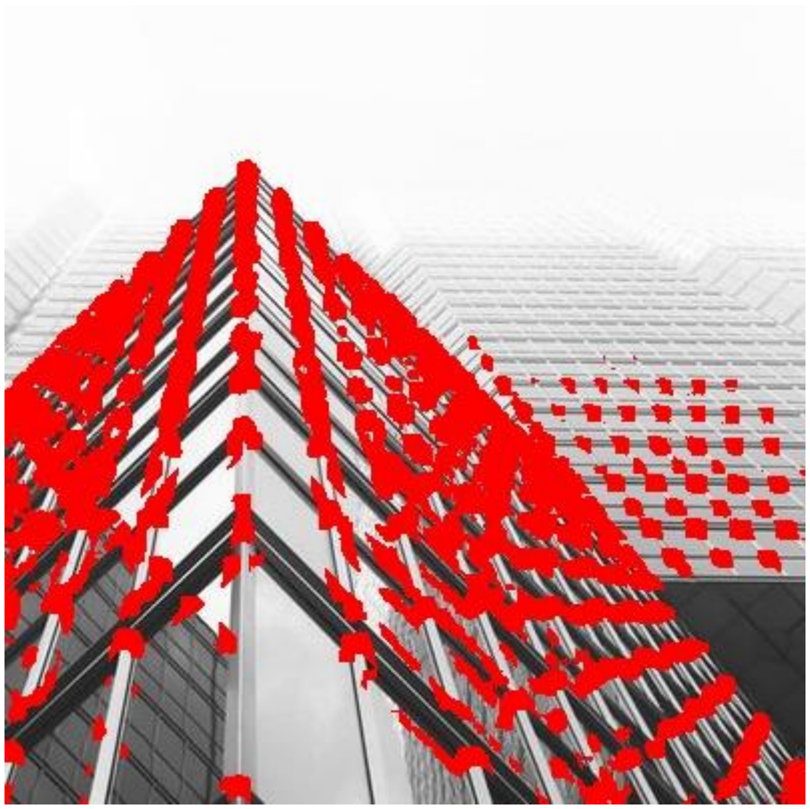
Window size = 9

Threshold = 1000*100000000

3. Corner detection with OpenCV:

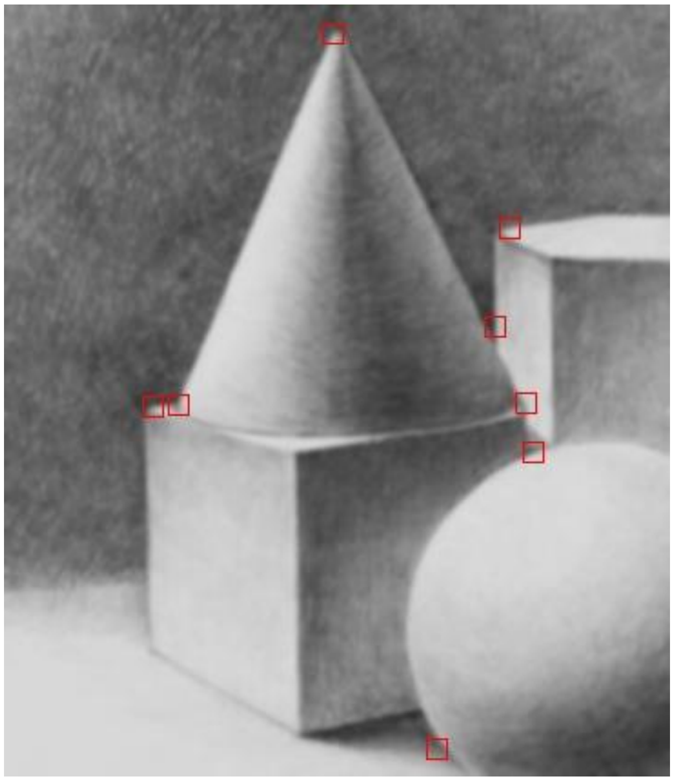
OpenCV corner detection with same parameters looks the same as above.





4. **Testing after smoothing a noisy image:**

   This pencil sketch has a lot of noise due to pencil strokes, therefore it requires more smoothing before corner detection.



Smoothing = 7

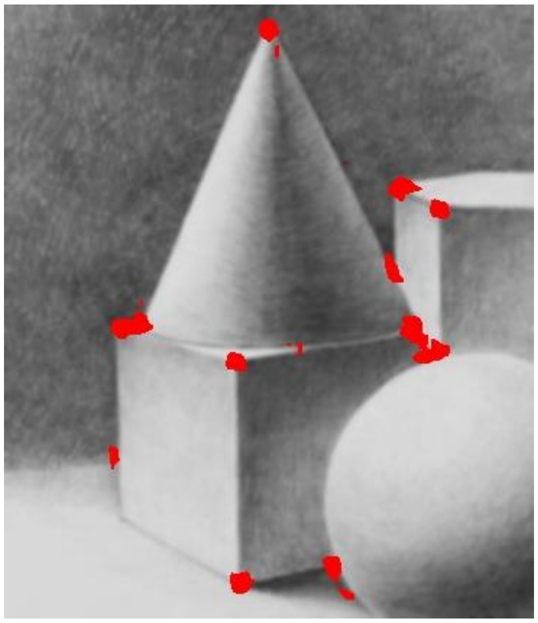
k = 0.04

Window size = 11

Threshold = 703*100000000

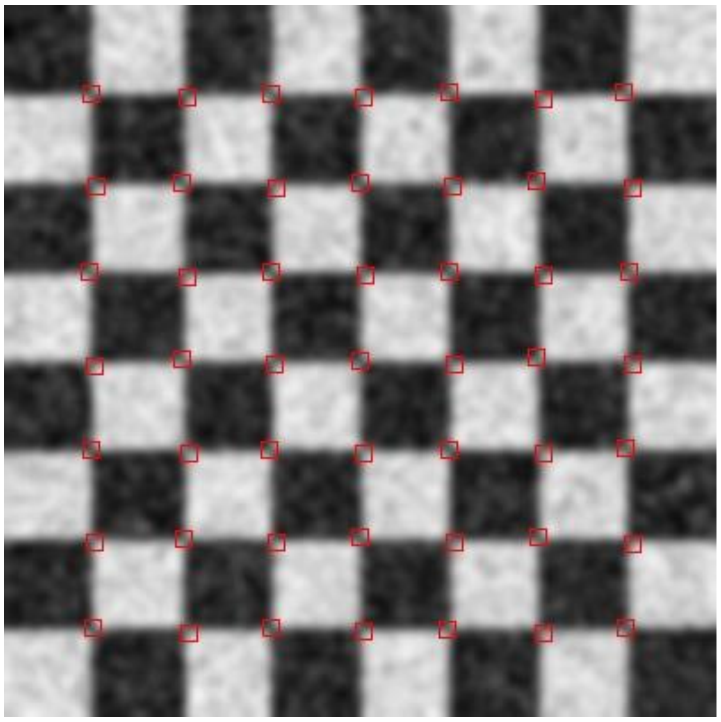
5. **Testing after smoothing a noisy image with OpenCV:**

   Corner detection with OpenCV in same pencil sketch image.



6. **Testing with Gaussian noise and smoothing:**

   Added gaussian noise to the image, then did gaussian smoothing, and at last detected the corners.



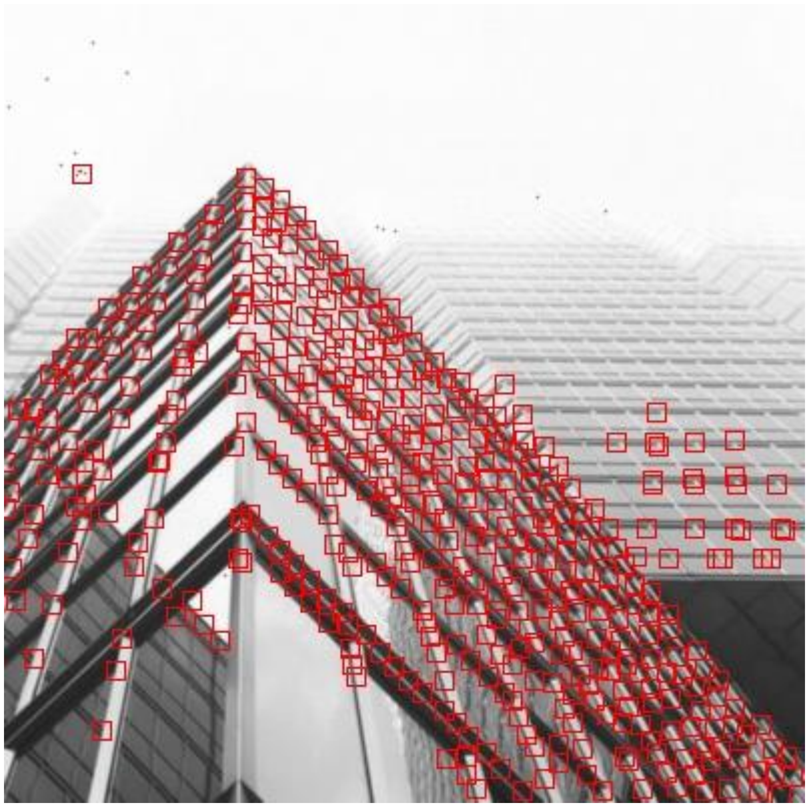
Variance = 1 ^ 0.05

Smoothing = 15

k = 0.04

Window size = 9

Threshold = 500*100000000



Variance = 1 ^ 0.05

Smoothing = 3

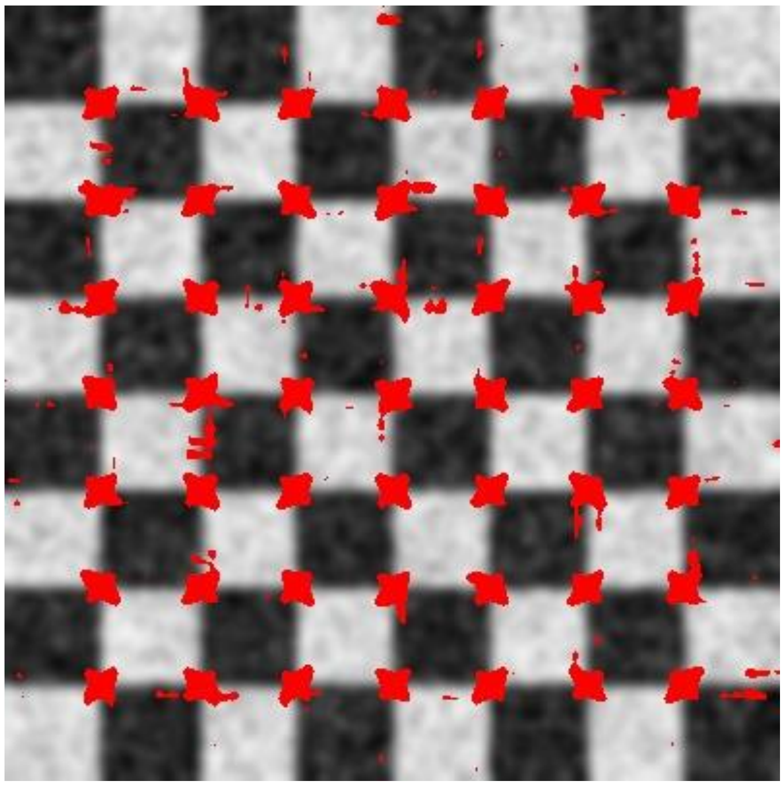
k = 0.04

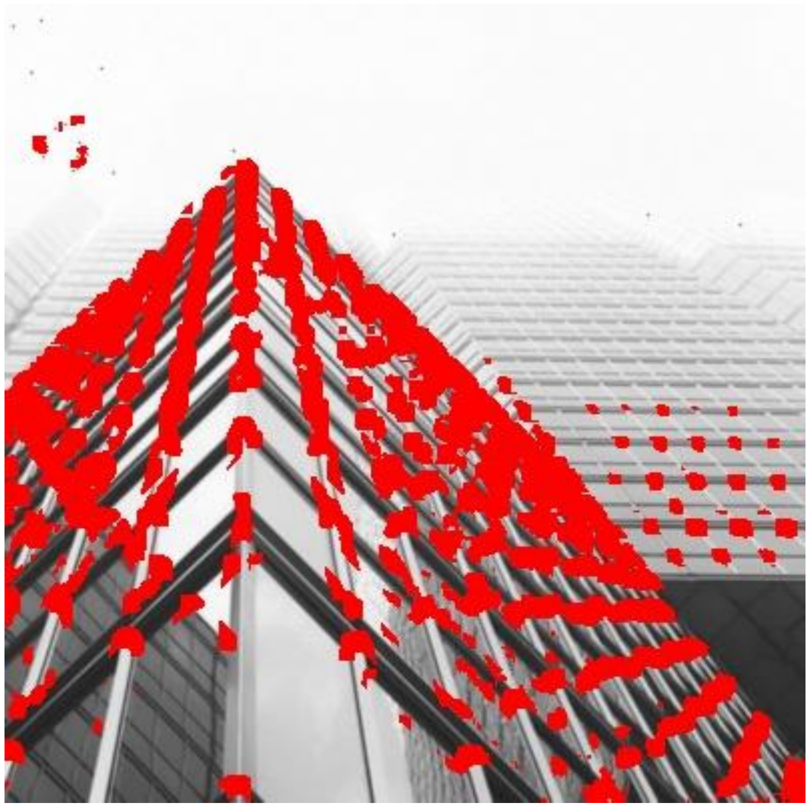
Window size = 9

Threshold = 1000*100000000

7. **Testing with Gaussian noise and smoothing with OpenCV:**

   Same gaussian noise and smoothing method but corner detection using OpenCV.

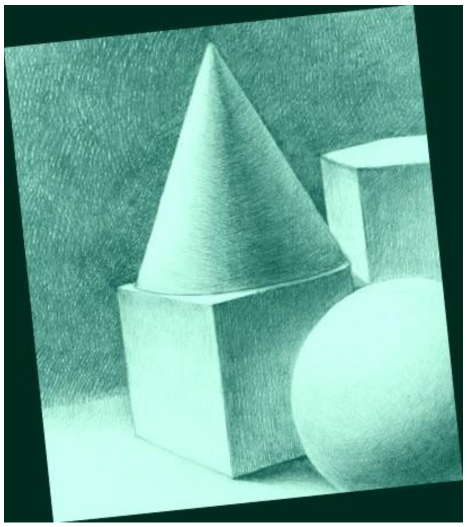




8. **Feature comparison:**

   Comparing features of 2 similar images using the feature vectors created. To get the second image, first image was rotated, brightness was decreased, and color frame was added on top of the original image.

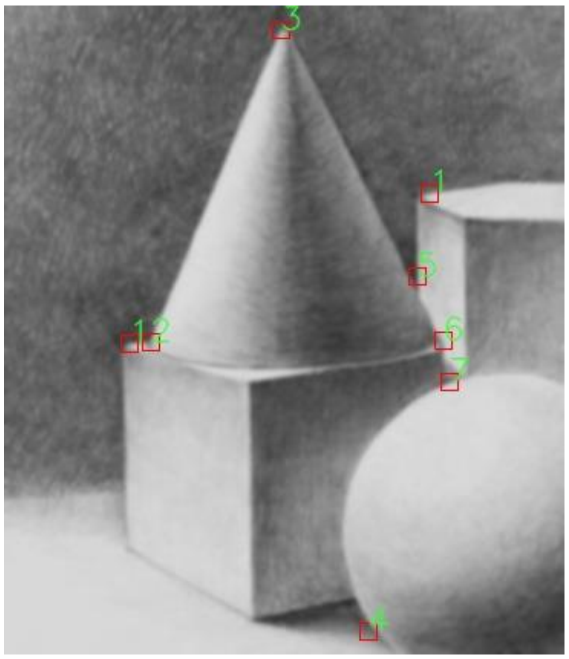
   Modified images:

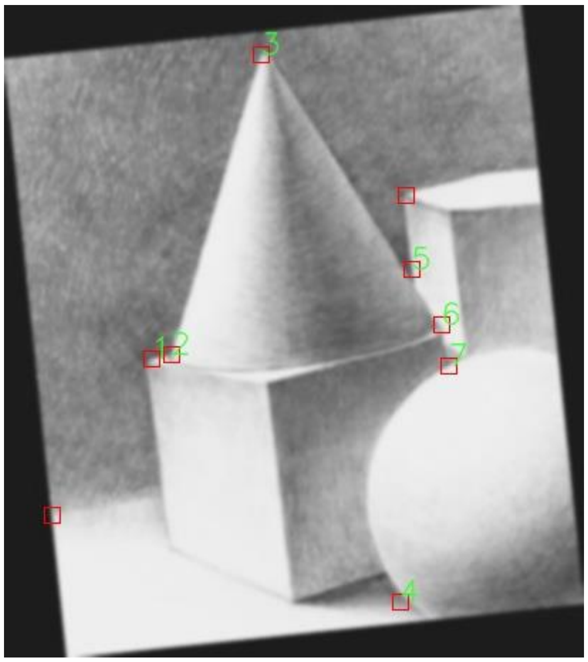


   Number of Corners detected in first image = 8

   Number of Corners detected in second image = 9

   Number of Features matched correctly in both images = 7

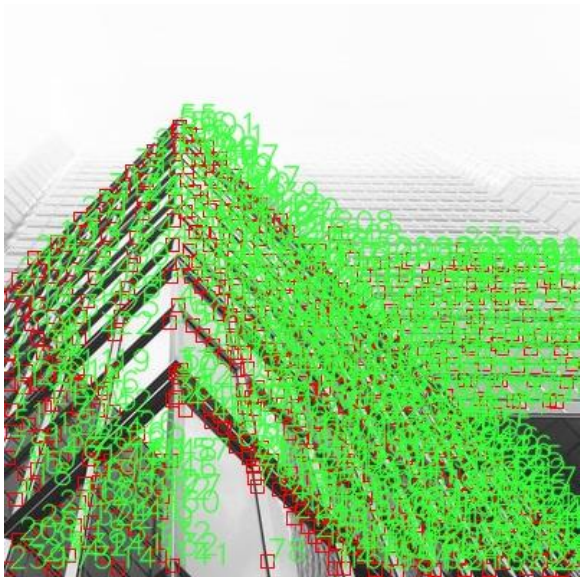
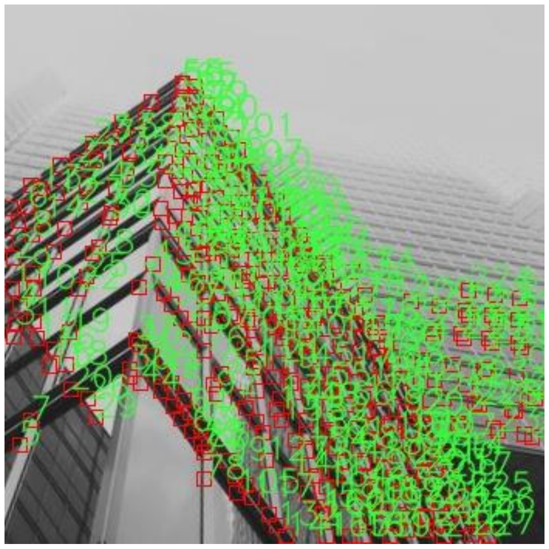




Number of Corners detected in first image = 600

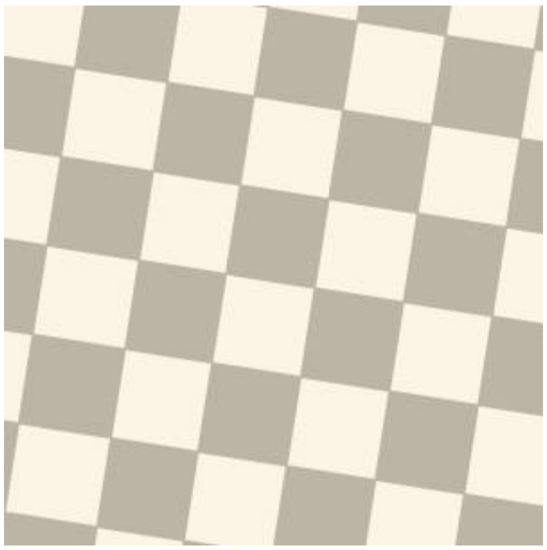
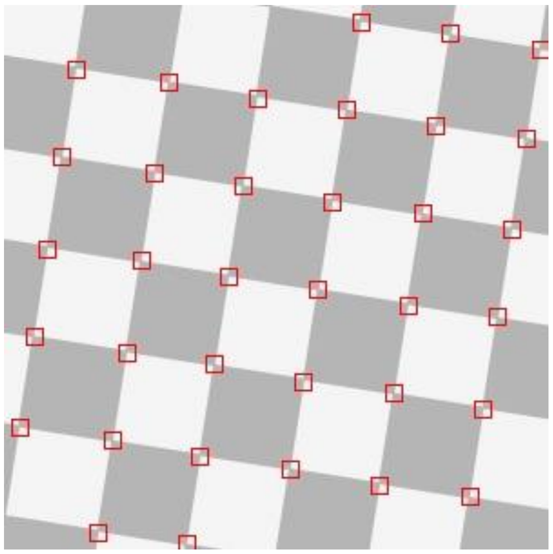
Number of Corners detected in second image = 372

Number of Features matched correctly in both images = 240

9. **Corner detection after image modification:**

   Images were rotated, scaled, brightness was changed, increased saturation, and added color frame on top of the original image.

5 References

OpenCV Corner Detection: https://medium.com/data-breach/introduction-to-harris-corner-detector-32a88850b3f6

HOG: https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/