CS 512 HW3
Harris Corner Implementation Report

Problem Statement 1:

As we are not allowed to use OpenCV functions directly so I have implemented Harris Corner Detection from scratch. Below I have given the algorithm in step by step which I used to implement Harris Corner Detector from scratch using Python and OpenCV.

Proposed Solution:

i)Compute x and y derivatives of the image:

For finding the gradients on each pixel (Ix, Iy) and applying the smoothing to reduce the noise in the image, I used "Gaussian Smoothing Filter". Because 2D Gaussian equation is separable, I take the first derivatives of that equation w.r.t. x and y and convolve them with the image to get the gradient values. SigmaX and sigmaY values are 5 and 5 by default and but my program is flexible enough that it can take different value from of sigma from user. To compute x and y derivative I have used Sobel Operators from OpenCV which can be given as,

```
Ix = cv2.Sobel(Blur, cv2.CV_64F, 1, 0, ksize=5)
Iy = cv2.Sobel(Blur, cv2.CV_64F, 0, 1, ksize=5)
```

ii) Compute the products of derivatives (Ixx, Iyy and Ixy):

After getting values of x and y derivatives using Sobel Operators, I am simply finding Ixx,Iyy by squaring it and finding Ixy by multiplying Ix and Iy. This can be given as,

```
Ixx = Ix ** 2
Ixy = Iy* Ix
Iyy = Iy ** 2
```

iii) Compute the sums of the products of derivatives at each pixel (Sxx,Syy and Sxy):

To compute covariance matrix for each pixel in the image by using values of Ix and I y, I am simply using the

$$C = \begin{bmatrix} \sum I_x{}^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y{}^2 \end{bmatrix}$$

Where,

Summation(Ix^2) =Sxx
Summation(Iy^2) =Syy
Summation(Ix*Iy) =Sxy

iv) Define at each pixel (x, y) the matrix:
Here, the matrix is constructed by the results of step 3 (Sxx, Syy, Sxy). This matrix shows the orientation of eigenvalues λ1 and λ2 and thus what the type of pixel is (corner, edge or flat region). The Eigen values of the Co-Variance Matrix at each pixel using SVD Decomposition can be given as,

$$C = U \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} V^T$$

v) Compute the response of the detector at each pixel:
By differing square of trace of the calculated matrix trace times the coefficient hCons from the matrix deter, I get the response(r) of the pixel.

```
r = deter - hCons * (trace ** 2)
```

Where,

```
deter = (Sxx * Syy) - (Sxy ** 2)
trace = Sxx + Syy
```

hCons =Harris detector free parameter in the equation.

vi) Check whether or not the response of a pixel exceeds the pre-defined threshold value:
Response value determined in step 5 is compared with a predefined threshold value, which is 1000000 by default, to classify the pixel as corner (if it's greater than threshold) or others we aren't interested in such as edge or flat region (if it's less than or equal to threshold). The value of the output image in given pixel is set as response value where the threshold is exceeded and is set as 0 (black) where the response fails exceeding the threshold. After finding all corners, my code is drawing empty red color rectangles over it.

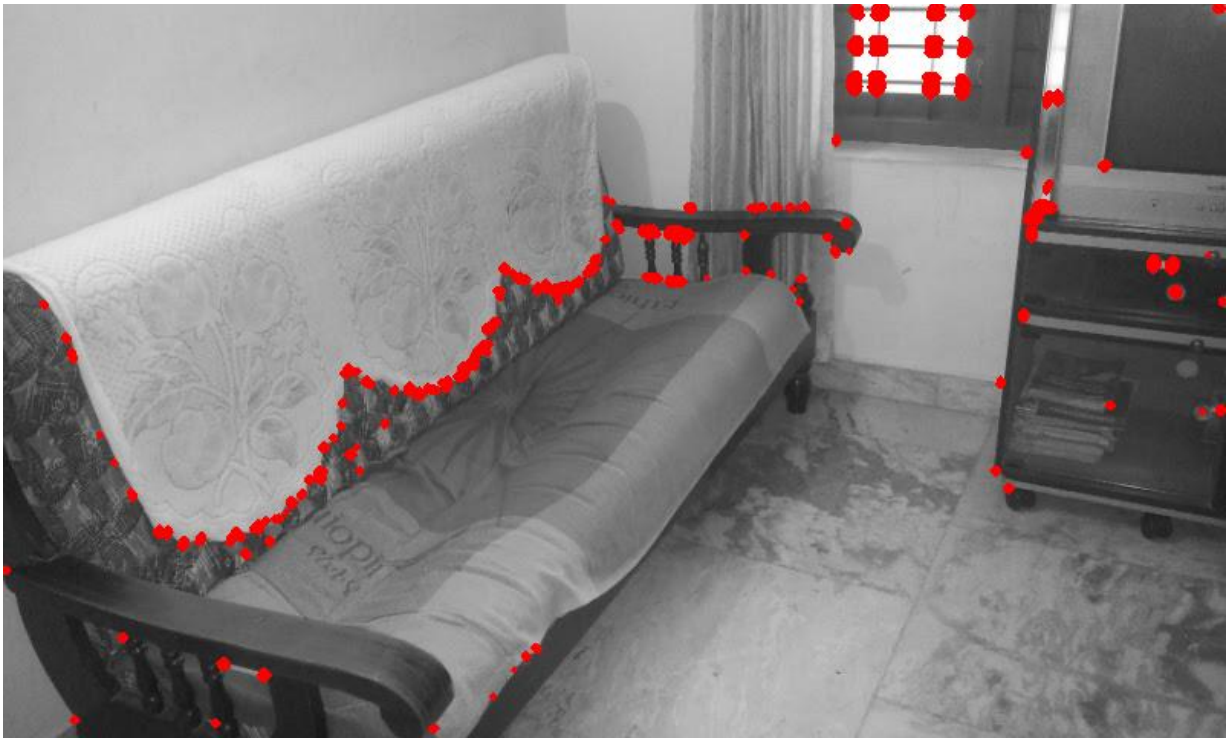
Again my code is flexible enough to take different threshold values to find corners in different scales.

```
if r > threshold:
  foundCorners.append([j, i, r])
  cv2.rectangle(output_img, (j, i), (j, i), (0, 0, 255), 3)
```

To Run This Program, please execute HarrisCornerDetection.py file in src folder.

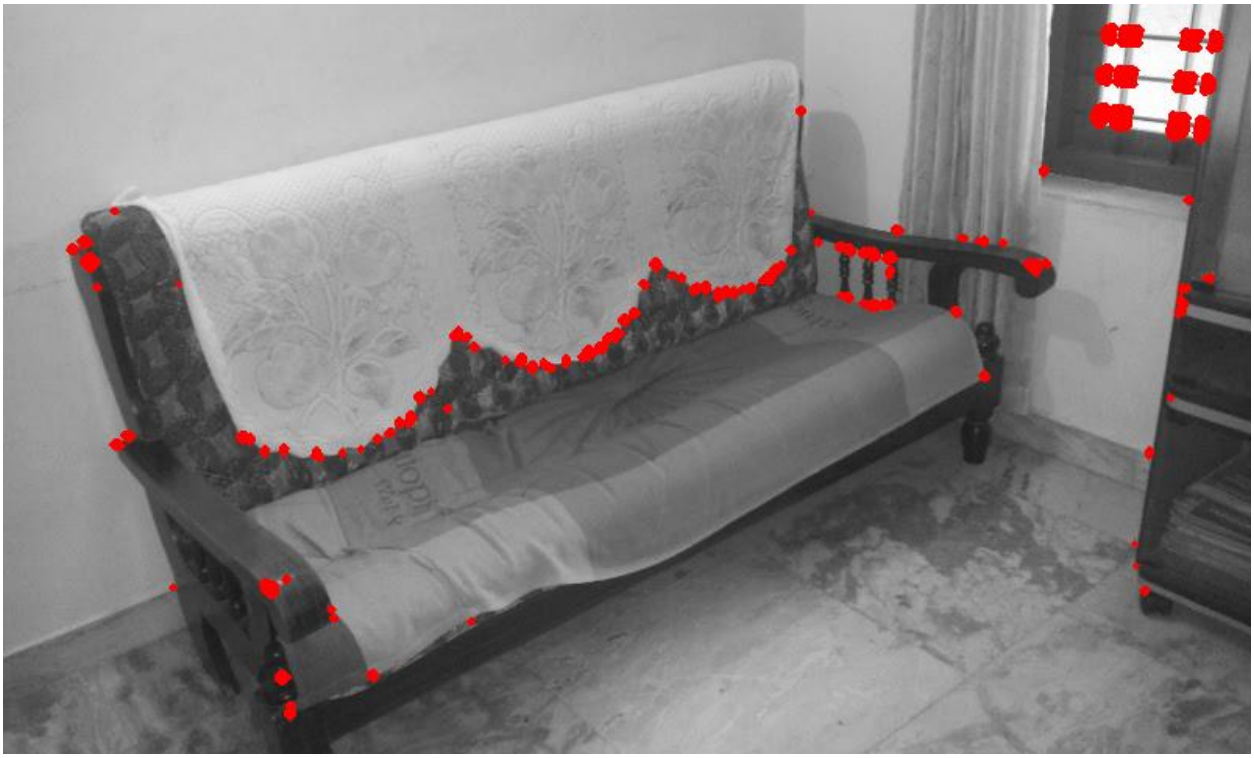
Input Image 1:

Output1 without Applying Good Localization:



Input Image 2 (Same image with different point of view):

Output2 without Applying Good Localization:



Problem Statement 2:
From the above output of corners detection, it's clear that we are having too many corners in a particular window size.

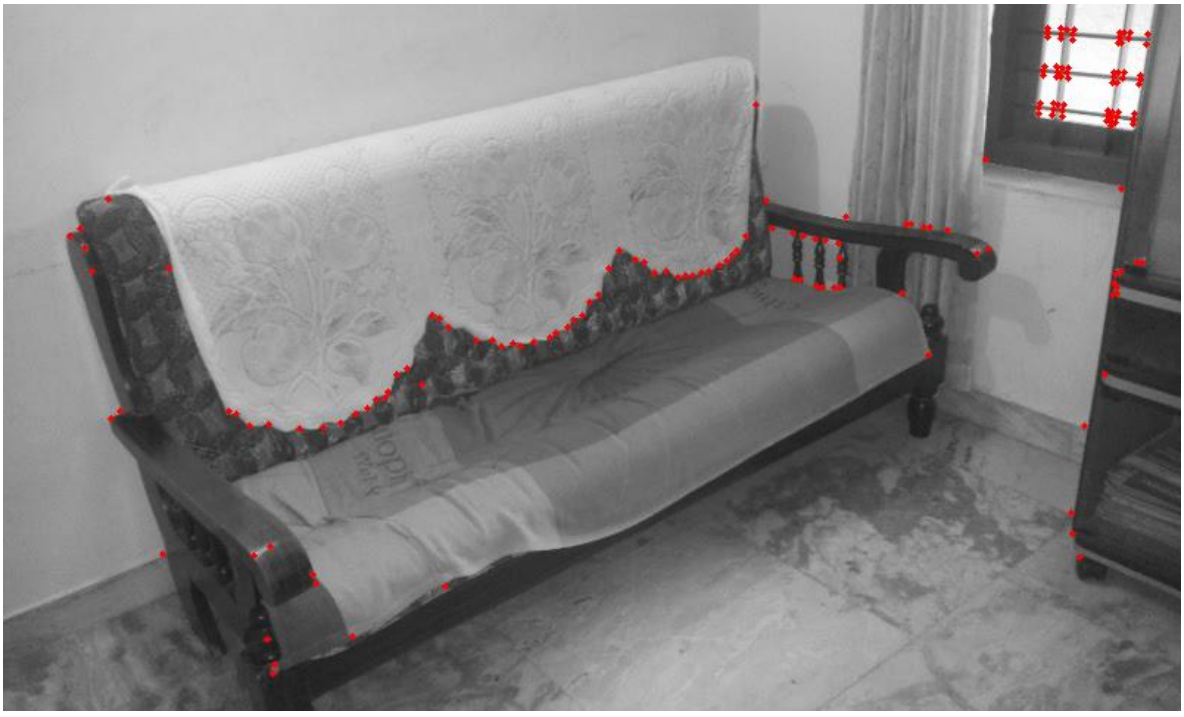
Proposed Solution:
Reduce number of corners using Non-Maximum Suppression. In Non-Maximum Suppression we basically pick up the optimal values to indicate corners, we find the local maxima by deleting corners in the vicinity of selected corners. The process of selecting non-maximum suppression can be stopped when deleting x% of pixels are completed in that detected corner window.

Input Image 1:

Output1 after Applying Good Localization:

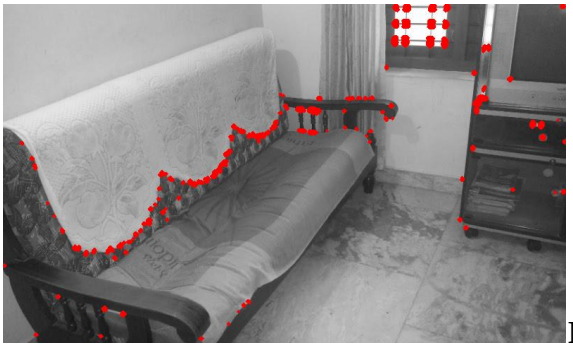
Input Image 2:

Output2 after Applying Good Localization:



Without Non-Maximum Suppression

With Non-Maximum Suppression

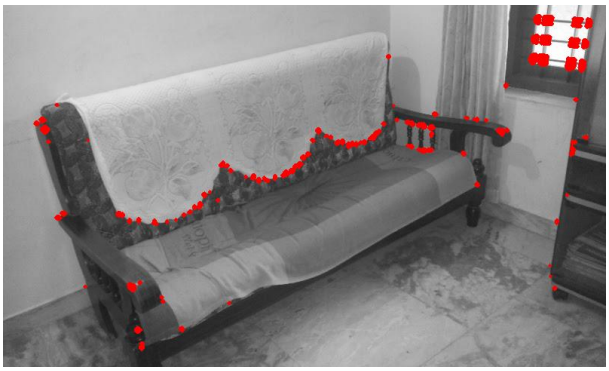


Number of corners before Suppression: 2754

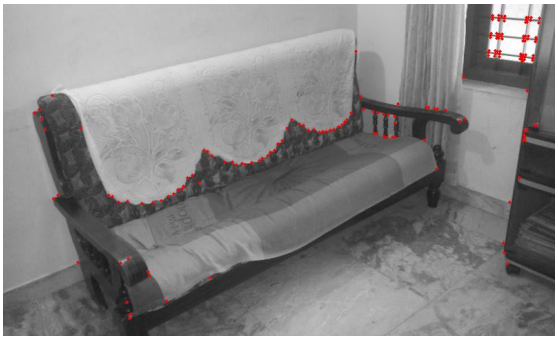
Number of corners after Suppression: 168

Without Non-Maximum Suppression

With Non-Maximum Suppression



Number of corners before Suppression: 2570



Number of corners after Suppression: 168

So from the above output results, we can observe that we are getting better corners after applying Non-Maximum Suppression in each window.

Problem Statement 3:
Feature Points Matching between Two Images using Harris corner vectors.

Proposed Solution:

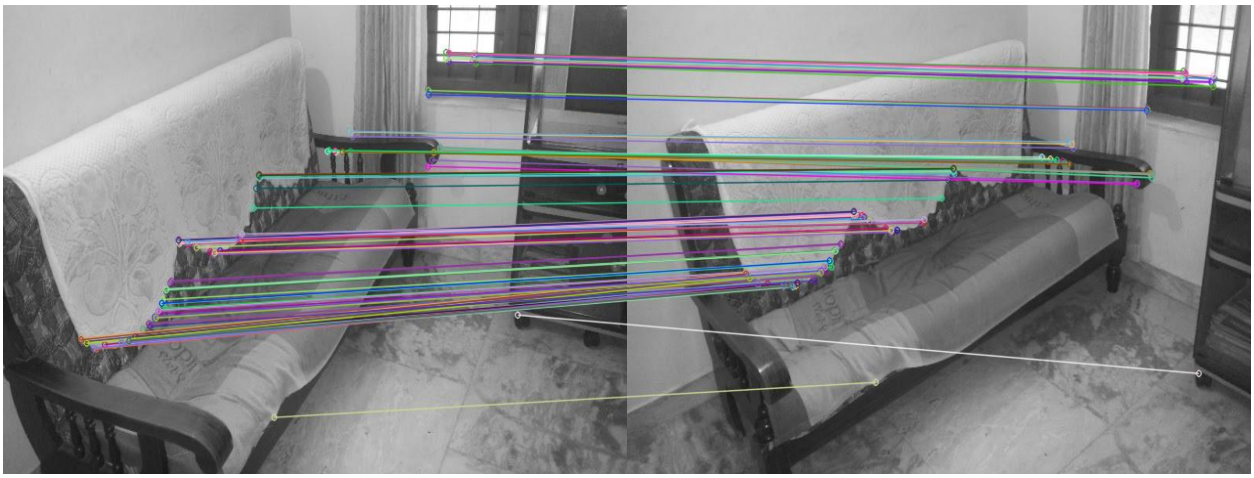
i) First the Harris corner features and the simple descriptors are computed for each of the images to be compared.
ii) Next, distance between each pair of corner feature descriptors is computed, by simply computing the *sum* the *absolute value of differences* between the *descriptor* elements.
iii) This *distance* is then used to compute the *best match* between a feature in one image and the set of features in another image by finding the one with the smallest distance.

$$SSD_{score} = \frac{1}{m^2} \sum_{i=1}^{n} (IP_1(i) - IP_2(i))^2$$

Output Result:
Input Image1:

Input Image2:

Matched Image based on the Harris Feature Vectors:



Challenges Faced and Way Forward:
It was challenging to match features between two images using Harris corner vectors as few edges are getting matched. Instead of only using Harris corner vectors and SSD we can use SIFT or SURF which would result in better features matching.

Conclusion:

In this assignment, I implemented the "Harris Corner Detection" algorithm from scratch. I used different sigma and threshold values to obtain the optimum results from the implementation. As a final statement, the implementation runs a little bit slower than expected and there may need to be optimization to run it more efficiently