Harris Corner Detector From First Principles

Aman Bilaiya
2018csb1069

Indian Institute Of Technology Ropar
Rupnagar, Punjab

Abstract

Feature Extraction is an important tool to draw meaningful insights from the image and perform analysis. Edges and Corners play an important role in various Image Feature Extraction Algorithms and Computer Vision Applications. Hence it becomes essential to detect corners and edges efficiently in an image. This report presents the implementation, observations and results of extracting edge and corners features from images using first principles, using python. This work consists of - Implementing a **Harris Corner detector.**

1 Introduction

Harris Corner Detector is a corner detection operator that is commonly used tool in computer vision algorithms to extract corners and infer features of an image.

A corner is a point whose local neighbourhood stands in two dominant and different edge directions. In other words, a corner is point of intersection of edges. Thus, while an edge has a gradient change in any one direction, a corner has a gradients change along multiple directions. The gradient of the corner (in both directions) have a high variation, which can be used to detect it. This is the principle used in Harris corner detector.

Also, it is popular because it is rotation, scale and illumination invariant. It was first introduced by Chris Harris and Mike Stephens in 1988.

2 Methodology

Harris Corner detection algorithm comprises of the following steps :-

[1] Computing smoothed gradients: First step is converting RGB image to gray scale. Gradients are computed by convolving image with the x and y Guassian derivatives. This is done by convolving image with standard 3*3 sobel filter.

Algorithm 1 convolveWithGaussianDerivative(img)

```
1: Fx ← $covolve(sobel\_x, img)$
2: Fy ← $covolve(sobel\_y, img)$
3: G ← $\sqrt{Fx^2 + Fy^2}$
4: Theta ← $arctan(\frac{Fy}{Fx})$
5: return Fx, Fy
```

[2] Harris Corner Score Calculation and Thresholding:

Using a window of size (2m+1)X(2m+1) the gradient image is scanned and the covariance matrix is computed, which contains the average of the products of x and y gradients. C at each point(x,y) is computed as follows:

$$C = 1/(2m+1)^2 \sum_u \sum_v \begin{bmatrix} F_x^2 & F_x F_y \\ F_x F_y & F_y^2 \end{bmatrix} = \begin{bmatrix} \langle F_x^2 \rangle & \langle F_x F_y \rangle \\ \langle F_x F_y \rangle & \langle F_y^2 \rangle \end{bmatrix}$$

Here (u, v) are the coordinates within the window: u = -m, ..., m, and v = -m, ..., m, the brackets (< and >) denote a dot product operation, and the gradients Fx and Fy on the right hand side of the above equation are read from the locations (x + u, y + v). [We used m=4 here]

R-score is calculated as half the harmonic mean of the Eigen values of the matrix C computed above, for each pixel. The Harris corner detection paper uses an approximation of this score as equivalent to as :

$$det(C) - k(Trace(C)^2)$$

We use the same formula to calculate the corner scores and using an appropriate threshold, find possible corner points in the image.

Algorithm 2 DetectCorners(img, Fx, Fy, m, k, Th)

```
1:  $corner\_list \leftarrow []$
2:  for $y \leftarrow m : img\_height - m$ do
3:      for $x \leftarrow m : img\_width - m$ do
4:          $Kxx \leftarrow Ixx[y-m : y+m+1, x-m : x+m+1]$
5:          $Kxy \leftarrow Ixy[y-m : y+m+1, x-m : x+m+1]$
6:          $Kyy \leftarrow Iyy[y-m : y+m+1, x-m : x+m+1]$
7:          $Sxx \leftarrow sum(Kxx)$
8:          $Sxy \leftarrow sum(Kxy)$
9:          $Syy \leftarrow sum(Kyy)$
10:         $determinant \leftarrow (Sxx * Syy) - square(Sxy)$
11:         $trace \leftarrow Sxx + Syy$
12:         $Rscore \leftarrow determinant - (k * square(trace))$
13:         $R\_mat[y][x] \leftarrow Rscore$
14:         if $Rscore \geq Th$ then
15:             $corner\_list.append([y, x, Rscore])$
16:         end if
17:     end for
18: end for
19: return R_mat, corner_list
```

[3] Get Corner Image: Using the possible corner_list we generate a corner image before applying NMS by iterating over the list and assigning image pixel 1.0 for red channel.

Algorithm 3 MakeCornerImg(img, corner_list)

```
1: $output \leftarrow img.copy()$
2: for $i$ in $range(len(corner\_list))$ do
3:     $y \leftarrow corner\_list[i][0]$
4:     $x \leftarrow corner\_list[i][1]$
5:     $output[y][x][0] \leftarrow 1.0$
6:     $output[y][x][1] \leftarrow 0.0$
7:     $output[y][x][2] \leftarrow 0.0$
8: end for
9: return output
```

[4] Non Maximal Suppresion: We reduce the thickness of corners using non-maximum suppression technique. From the above algorithm we will get the detected corners list along with their Rscore. Now, Sort this list based on Rscore in descending order. Then iterate this list from starting, for each point p, remove all points in the connected neighborhood of p that occur later in the corner_list. In implementation, we use the neighbourhood to be within a distance of 3 pixels to the pixel in consideration. After this step we have the final list of corner points, we simply highlight those points as in the image using some symbols. In this assignment we have used red plus sign for corners.

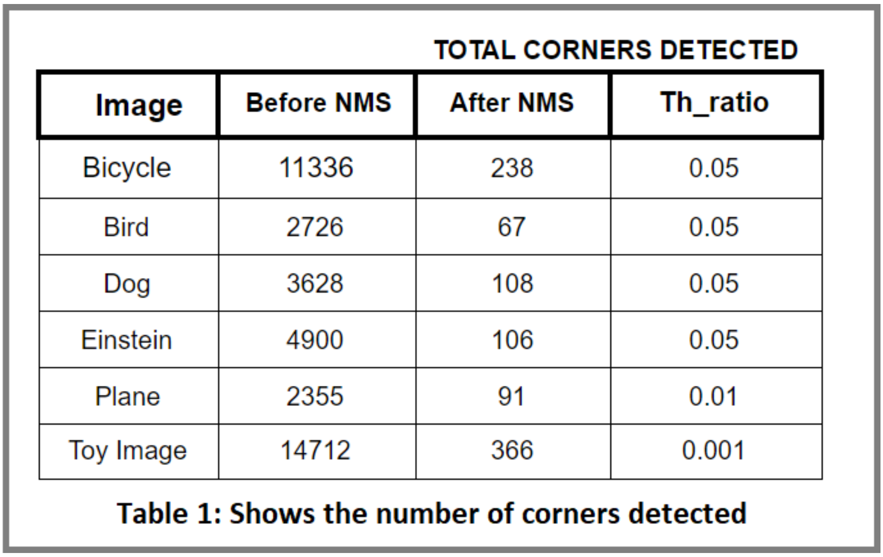
Algorithm 4 NonMaximalSuppression(corner_list)

```
1: $sorted\_L \leftarrow reverse\_sort(corner\_list)$
2: $final\_corner\_list \leftarrow []$
3: $final\_corner\_list.append(sorted\_L[0][:-1])$
4: $dis \leftarrow 3$
5: for $i$ in $sorted\_L$ do
6:     for $j$ $final\_corner\_list$ do
7:         if $(abs(i[0]-j[0] \leq dis)$ $and$ $abs(i[1]-j[1]) \leq dis)$ then
8:             $break$
9:         else
10:            $final\_corner\_list.append(i[:-1])$
11:        end if
12:    end for
13: end for
14: return final_corner_list
```

3 Results & Observations

1) For the Harris corner detection, the optimal window size was found to be (9 X 9) i.e, m=4. The corner response was found to be largely dependent on the threshold kept for the harris corner score calculated using the formula given above.

2) Finding the Eigen value from covariance matrix and then calculating the Rscore was computation heavy, the approximate method of using the difference of determinant and square of trace of the covariance matrix gave reasonable output,using k=0.04, and is less computational.



TOTAL CORNERS DETECTED

Image	Before NMS	After NMS	Th_ratio
Bicycle	11336	238	0.05
Bird	2726	67	0.05
Dog	3628	108	0.05
Einstein	4900	106	0.05
Plane	2355	91	0.01
Toy Image	14712	366	0.001

Table 1: Shows the number of corners detected

3) Generally, Harris constant (k) belongs to [0.04,0.06] for all images.

4) Lowering the threshold value increases the number of corners detected (even it may not be a corner) and also increases algorithm's time complexity. As threshold increases, less number of corners are detected. Threshold value should be set so as to get around 10-100 corners.

5) Junction of blur edges were not detected as corners.

6) For the toy_image, shapes with rounded corners were not detected. This is an expected drawback of Harris corner detector as it is not scale invariant.

7) Upon changing the neighbourhood limit in the Non-maximal suppression there is noticeable change in the final output.

8) In the dog and Einstein image, the detector algorithm gives highlights lot of points near eyes and other facial features. Thus, could have an application in face recognition and other.

4 Insights and Take Aways

1) Corners have high positive R value. Large negative R values represent edges and small absolute R values, thus the flat region.

2) Harris corner detector is invariant to rotation & linear translation because R score of any pixel remains same after rotation or translation.

3) Harris corner detector is invaraint to intensity shift because first derivative involved in the algorithm remains same on linear shift.

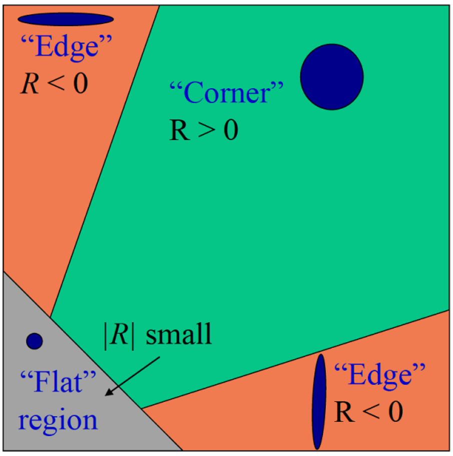


Figure 1: Mathematical Interpretation of R-score using Eigen values

4) Harris corner detector is not invariant to scaling. On scaling up, corner appears as edge and on scaling down, edge appear to be corner. Relative neighbourhood intensity shifts change on scaling and thus the R score.

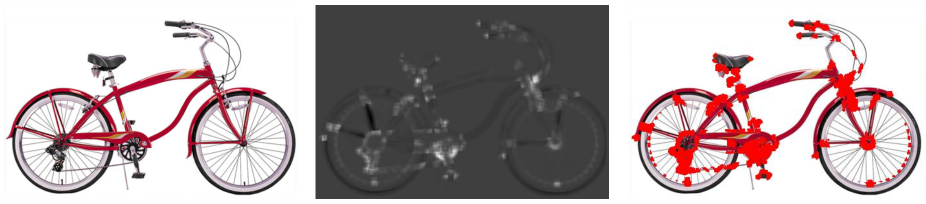
5) Harris corner detector is sensitive to noise in the image. So, Gaussian smoothing is used to remove noise effects on corner detection.

6) Harris corner detector depends on a threshold parameter which in a way controls the accuracy and precision of the algorithm. This parameter varies significantly for different images.

5 References

1) IITM Summer School
2) Wikipedia
3) Lectures slides
4) Online resources
5) opencv.org

Figures[2-7] are the final and intermediate output images obtained from our Harris corner detector algorithm :- Taken m=4, k=0.04 and Th_ratio was chosen such that the output is similar to what our human eye will observe.
With proper thresholds chosen, corners are detected close to accurate.


(a) Input (b) R values (c) Possible Corners

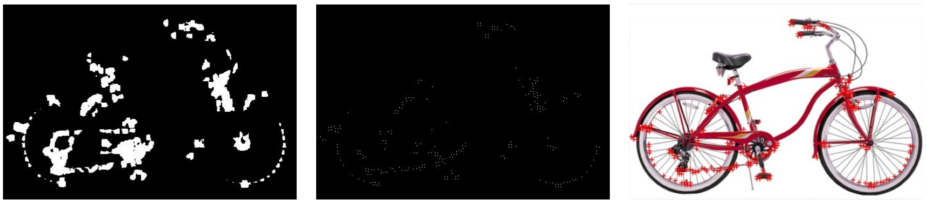
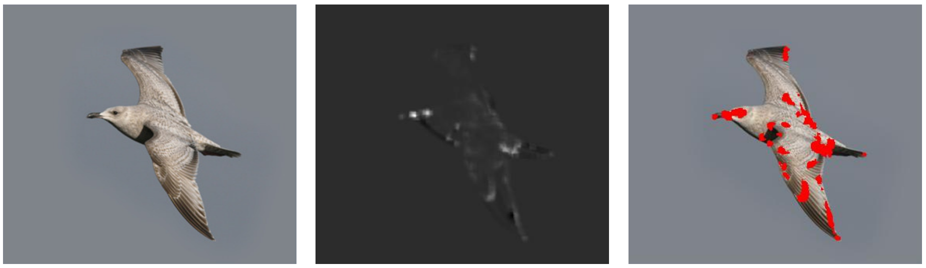

(d) Before NMS (e) After NMS (f) Marked Corners

Figure 2: Intermediate and Final images of Bicycle with Th_ratio=0.05


(a) Input (b) R values (c) Possible Corners

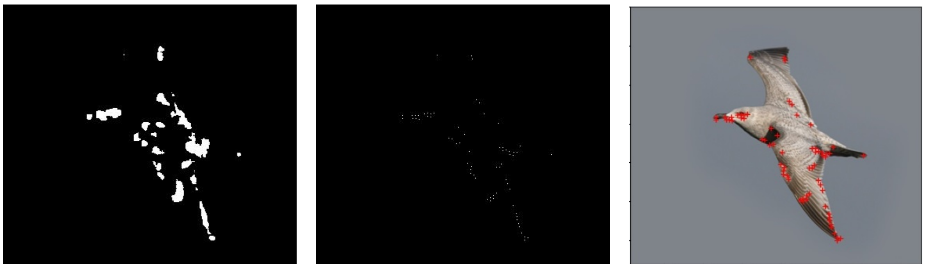
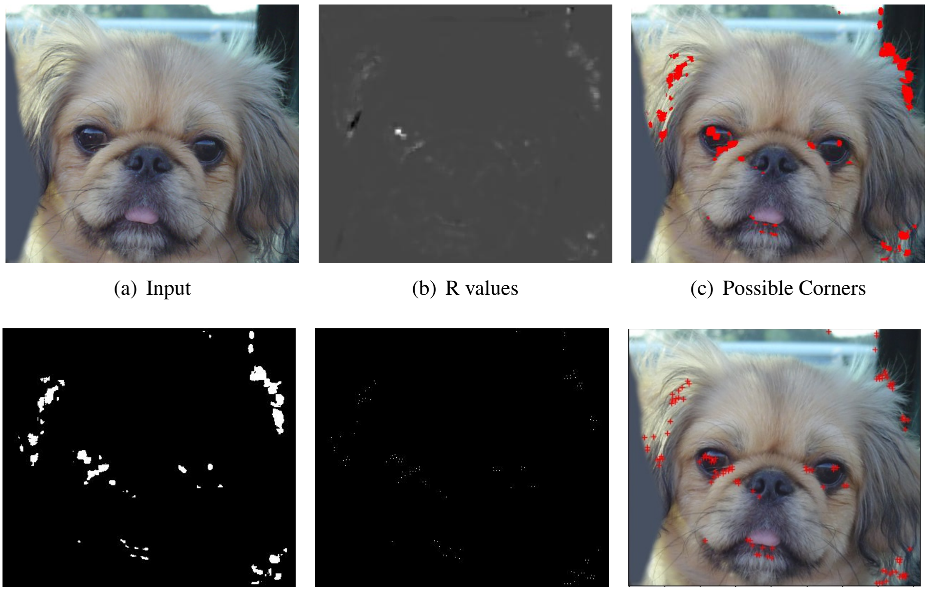
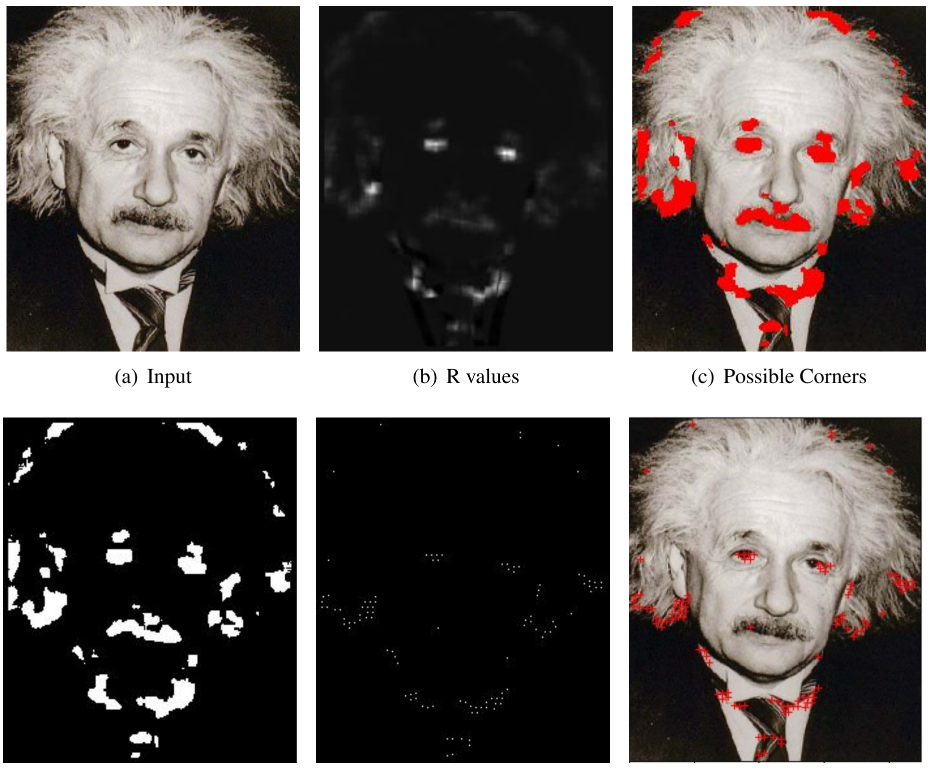

(d) Before NMS (e) After NMS (f) Marked Corners

Figure 3: Intermediate and Final images of bird with Th_ratio=0.05


(a) Input (b) R values (c) Possible Corners

(d) Before NMS (e) After NMS (f) Marked Corners

Figure 4: Intermediate and Final images of dog with Th_ratio=0.05



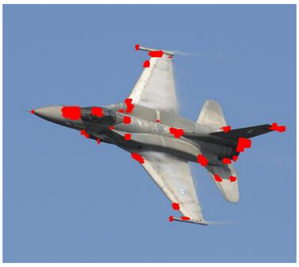
(a) Input (b) R values (c) Possible Corners

(d) Before NMS (e) After NMS (f) Marked Corners

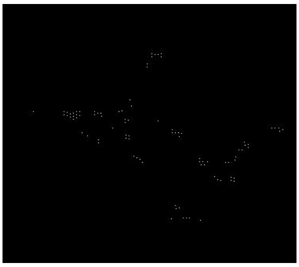
Figure 5: Intermediate and Final images of einstein with Th_ratio=0.05

(a) Input

(b) R values


(c) Possible Corners

(d) Before NMS


(e) After NMS

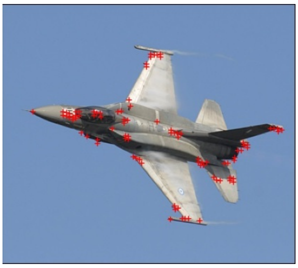
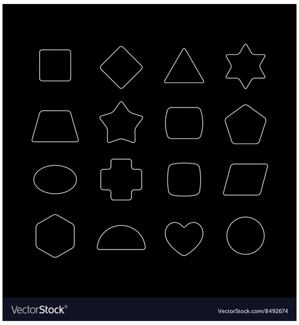
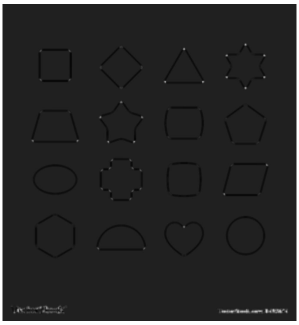
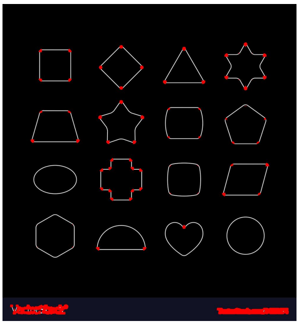

(f) Marked Corners

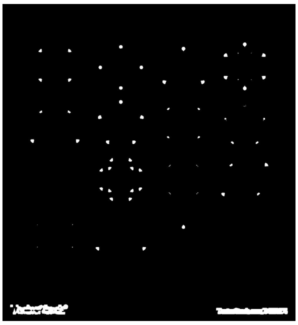
Figure 6: Intermediate and Final images of plane with Th_ratio=0.01

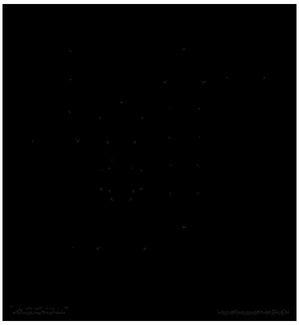


(a) Input


(b) R values


(c) Possible Corners


(d) Before NMS


(e) After NMS

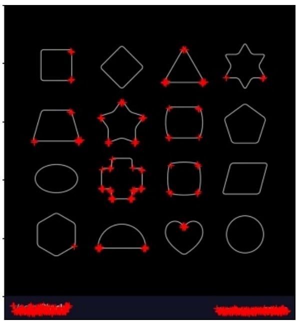
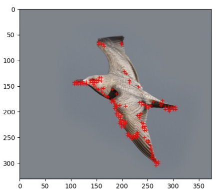
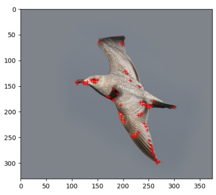
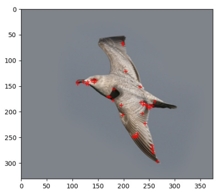
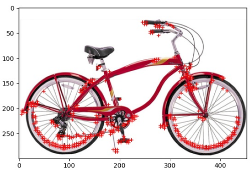

(f) Marked Corners

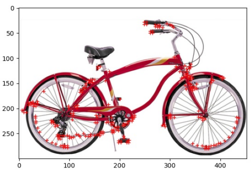
Figure 7: Intermediate and Final images of toy$_{i}magewithTh\_ratio = 0.001$

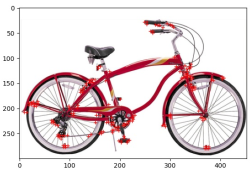

(a) Th_ratio 0.001

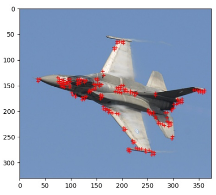

(b) Th_ratio 0.05

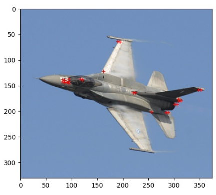

(c) Th_ratio 0.1


(d) Th_ratio 0.001


(e) Th_ratio 0.05


(f) Th_ratio 0.1


(g) Th_ratio 0.001


(h) Th_ratio 0.05

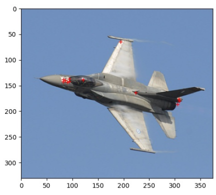

(i) Th_ratio 0.1

Figure 8: Experimentation with different threshold ratios on input images