Harris Corner Detection using First Principles

Parth Goyal
2017csb1095
2017csb1095@iitrpr.ac.in

Indian Institute of Technology
Ropar, 140001
Punjab, India

Abstract

This document is the report of Part-2 CS518 Computer Vision Assignment 2, where we implement the Harris Corner Detection Algorithm for Corner Detection from first principles. The Harris corner detector uses a multi-stage algorithm to detect a wide range of corners in images. It was developed by Chris Harris and Mike Stephens in 1988.

1 Introduction

One of the features are corner and to find them harris corner detection algorithm is widely used. Here we find the weighted squared value for pixel and then optimizes it. Linear Algebra and calulus along with taylor's theorem has been used, to come to results.

2 Methodology | Algorithm

The Harris Corner detection algorithm is composed of 4 Steps :

- Luminence Extraction
- Filtered Gradient calculation
- Finding Possible Corners
- Non-maximal Suppression

2.1 Luminence Extraction

We have worked on 2D luminence image, instead of the original 3D RGB image. Since RGB doesn't have any role to be played in corner detection. .

Algorithm 1 rgbtogray(img)

$R \leftarrow img[:,:,0]$
$G \leftarrow img[:,:,1]$
$B \leftarrow img[:,:,2]$
$grayscale \leftarrow (0.2989 * R) + (0.5870 * G) + (0.1140 * B)$
return grayscale

(a) Original Image (b) Luminence Image

Figure 1: Original Image and Luminence Image

2.2 Filtered Gradient Calculation

For the algorithm to work according to mathematical proof we need to have Ix, Iy etc. for this we need to calculate gradients hence sobel filters are used for this purpose. As done in canny detector.

Algorithm 2 getImageGradientComponents(img)

1: $sobel\_filter\_x \leftarrow array[[-1,0,1],[-2,0,2],[-1,0,1]]$
2: $sobel\_filter\_y \leftarrow array[[1,2,1],[0,0,0],[-1,-2,-1]]$
3: $gradx \leftarrow convolve(img, sobel\_filter\_x)$
4: $grady \leftarrow convolve(img, sobel\_filter\_y)$
5: **return** gradx, grady

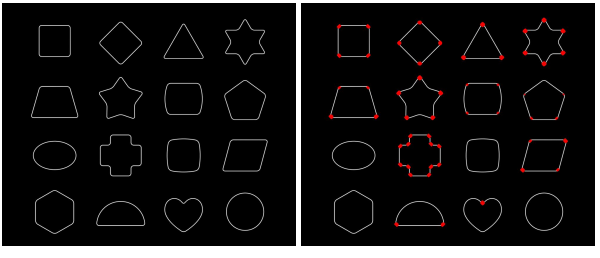
2.3 Finding Possible Corners

here we first set s, and according to it the window size is 2*s +1 For each pixel (x, y), we look in a window of size 2m+1 x 2m+1 around the pixel and from this we find the M and we here use weights to 1 otherwise gaussian weights may also be used. From this we calculate R value which is det- k* (trace)*trace k being a parameter.

then we use thresholding to detect our possible corners.

Algorithm 3 findCorners(img, Ix, Iy, m, k, thresh)

Initialize $Ixx = Ix^2, Iyy = Iy^2, Ixy = IyIx$, empty list cornerList and
$height, width = img.shape$ and Initialize zero matrix rMat(height, width)
for $x \leftarrow m : height - m$ **do**
    for $y \leftarrow m, width - m$ **do**
        $Sxx \leftarrow (Ixx(y-m : y+m+1, x-m : x+m+1)).sum()$
        $Sxy \leftarrow (Ixy(y-m : y+m+1, x-m : x+m+1)).sum()$
        $Syy \leftarrow (Iyy(y-m : y+m+1, x-m : x+m+1)).sum()$
        $det \leftarrow (Sxx * Syy) - (Sxy^2)$
        $trace = Sxx + Syy$
        $r = det - k(trace^2)$
        $r(x, y) = r$
    end for
end for
$T \leftarrow thresh * rMat.max$
for each pixel (i,j) in img **do**
    if $r(i, j) > T$ **then**
        $corners.append(j, i, r(j, i))$
    end if
end for
return corners



(a) Luminence Image (b) Corners

Figure 2: Corner Detected before applying NMS on Luminence Image, m = 4 k = 0.04

2.4 Non-Maximal Suppression

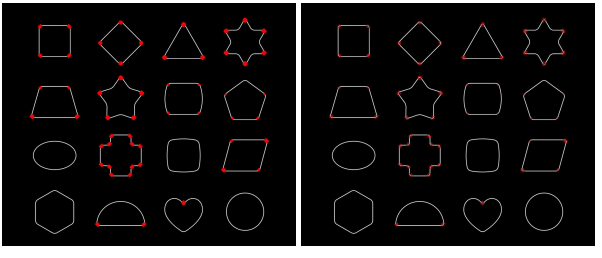
Now we need to reduce the cluster of corner points in an area to 1 point, where the the r value is the maximum. For this we sort the entire list in descending order and traverse the list. We mark all the visited corners in the list. For each unvisited pixel, we append the value into another list and mark the pixel as well as its neighbouring pixels as visited, if they are present in the list. We finally return the list that we made to store the unvisited corners as Non-maximal supressed corners.

We use the neighbourhood in this implementation to be within a distance of 3 pixels of the pixel in consideration.

After this step we have the final list of corner points, we simply add those points as in the image as some symbols. In this implementation we have used a red plus sign for corners.

Algorithm 4 nonMaximalSupressionCorners(img)

```
Initailize empty list finalCornerList
Initailize visited array with zeros
sort according to the r value
for i ← 0 : len(corners) do
    if visited[corners[i][0], corners[i][1]] != 1 then
        y ← corners[i][0]
        x ← corners[i][1]
        finalCornerList.append(x,y)
        visited[x,y] ← 1
        for each valid index p,q of neighbouring pixel of (x,y) all do
            visited[p,q] ← 1
        end for
    end if
end for
return finalCornerList
```



(a) Corners Before NMS (b) Harris Corner Detected

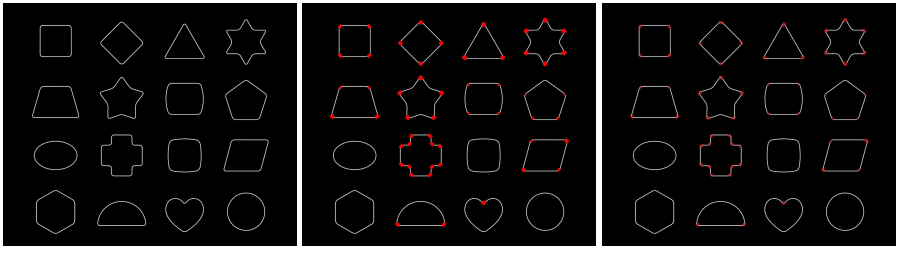
Figure 3: Harris Corner Detected Image, m = 4 k = 0.04

3 Observations

- For the toy image, circles show no corner detected, and
- If we remove the noise we may loose some corners, eg. in plane figure.
- More neihgbour can be chosen for non maximal supression which gives less corners.
- Features of humans and animals can be detected as a lot of features are being detected by this.

• 4 Walkthrough of the Harris Edge Detector Algorithm

Below is the ouput of the images by Harris Corner Detection Algorithm :

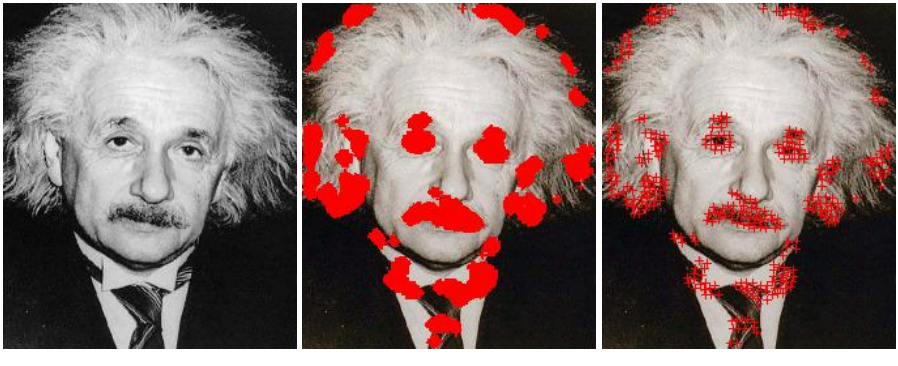


(a) Luminence Image (b) Threshold Image (c) Canny Edge Detected

Figure 4: All Intermediate and Final Images through Harris Edge Detector of Toy, m = 4 k = 0.04

(a) Luminence Image (b) Threshold Image (c) Canny Edge Detected

Figure 5: All Intermediate and Final Images through Harris Edge Detector of Plane, m = 4 k = 0.04



(a) Luminence Image (b) Threshold Image (c) Canny Edge Detected

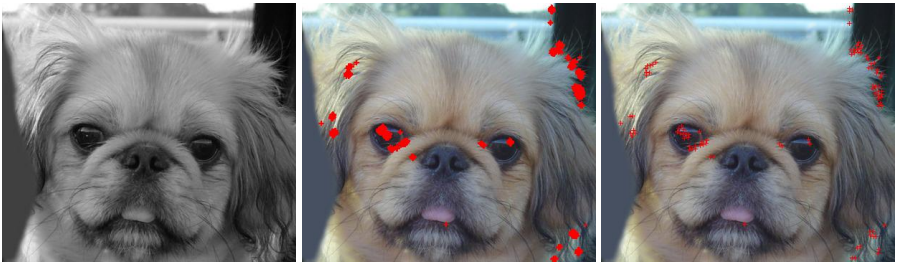
Figure 6: All Intermediate and Final Images through Harris Edge Detector of Einstein, m = 4 k = 0.04

(a) Luminence Image (b) Threshold Image (c) Canny Edge Detected

Figure 7: All Intermediate and Final Images through Harris Edge Detector of Dog, m = 4 k = 0.04

(a) Luminence Image (b) Threshold Image (c) Canny Edge Detected

Figure 8: All Intermediate and Final Images through Harris Edge Detector of Bird, m = 4 k = 0.04

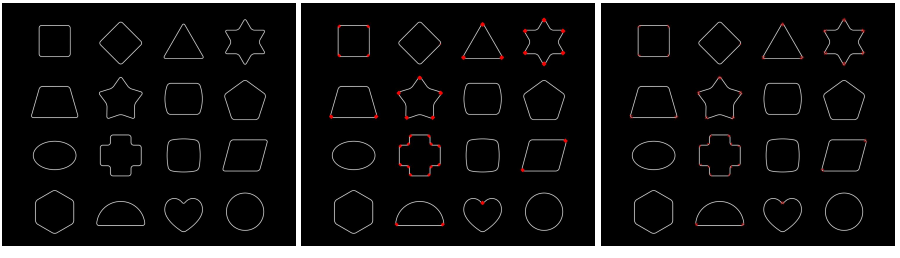


(a) Luminence Image (b) Threshold Image (c) Canny Edge Detected

Figure 9: All Intermediate and Final Images through Harris Edge Detector of Dog, m = 3 k = 0.05

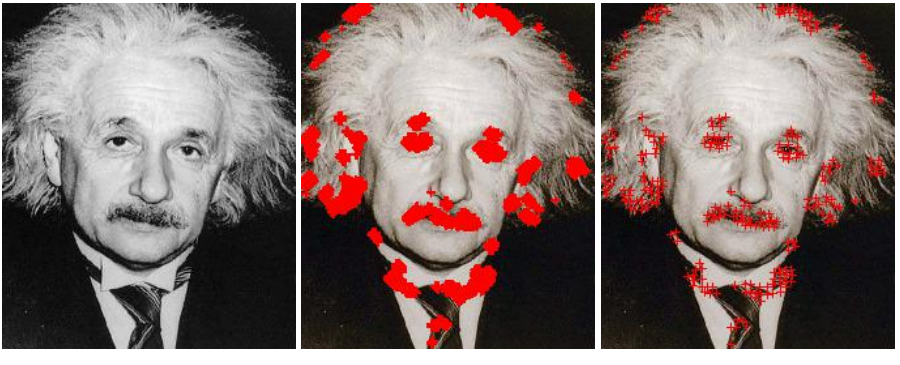
(a) Luminence Image (b) Threshold Image (c) Canny Edge Detected

Figure 10: All Intermediate and Final Images through Harris Edge Detector of Plane, m = 3 k = 0.05



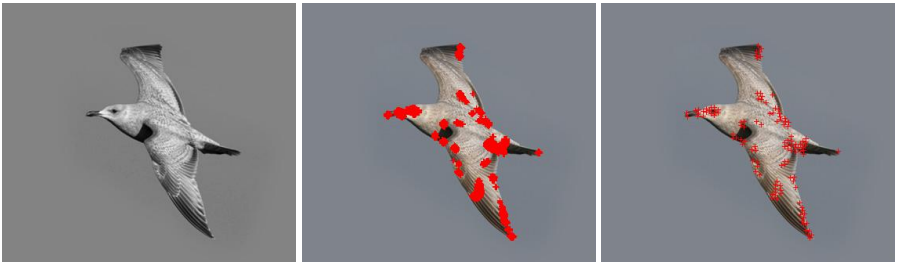
(a) Luminence Image (b) Threshold Image (c) Canny Edge Detected

Figure 11: All Intermediate and Final Images through Harris Edge Detector of Toy, m = 3 k = 0.05



(a) Luminence Image (b) Threshold Image (c) Canny Edge Detected

Figure 12: All Intermediate and Final Images through Harris Edge Detector of Einstein, m = 3 k = 0.05



(a) Luminence Image (b) Threshold Image (c) Canny Edge Detected

Figure 13: All Intermediate and Final Images through Harris Edge Detector of Bird, m = 3 k = 0.05