Implementation of Canny Edge and Harris Corner Detector

Mcmillon Walton Student
2017chb1047@iitrpr.ac.in

The Indian Institute
of Technology,Punjab
Ropar, India

Abstract

Edges and corners carry vital information about an image. They are the **primary image features** for extraction by low level processing techniques and are the initial step for many Computer Vision projects. This report discusses on the Canny Edge [1] and Harris corner algorithm [2] which are used to extract the said features.

1 Introduction

Canny Edge Detector is a multi-stage algorithm to detect the imperative edges and ignore any useless or redundant information which may confuse the computer[1]. It was developed by John F.Canny in 1986. It is useful for extracting structural information from objects and reduce data for further processing.

Harris Corner Detector similar to Canny Edge Detector is used to detect the Imperative Corners present in the image. Corner here is simply where two or more edges meet[2] in the image. It was first introduced by Chris Harris and Mike Stephens in 1988.

2 Methodology and Algorithm

The **Canny Edge algorithm** comprises of 5 steps

Conversion to gray scale

First the RGB Color image is converted to gray scale by this formula.

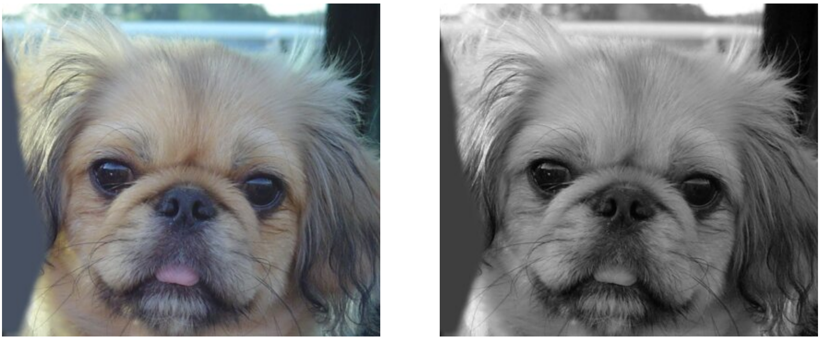
$GrayScaleimage \leftarrow 0.2126 \times R + 0.7152 \times G + 0.0722 \times B$

Here R,G and B[3] are matrix with the dimensions of the image.

[1]We want to make the algorithm insusceptible to noise and extract generic edges which are important and is enough for a human to tell what the image contains

[2]Where gradient dominantly changes in more than two directions

[3]They are the individual channel of the RGB image



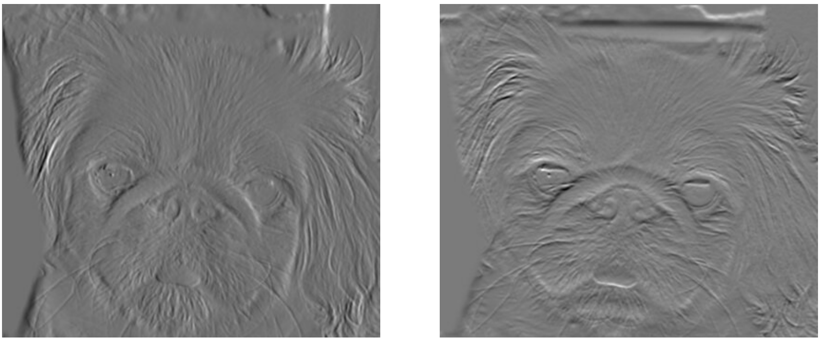
(a) Original image (b) Gray scale image

Figure 1: Conversion to Gray scale

Compute Derivative of Gaussian

Usually Gaussian is first applied to the image[4] and then the derivative filter is applied to get the gradient present in the image. But in order to save some computation cost and reduce one operation. We exploit the property of convolution being associative and instead first convolve the gaussian filter with derivative filter.

The derivative/gradient filter used here for x-y component are simply.
for x-component: $\begin{bmatrix} -1 & 1 \end{bmatrix}$
for y-component: $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$
Then simply convolve these 2 filters with the image.



(a) Fx (b) Fy

Figure 2: X and Y gradient components

[4] to reduce noise

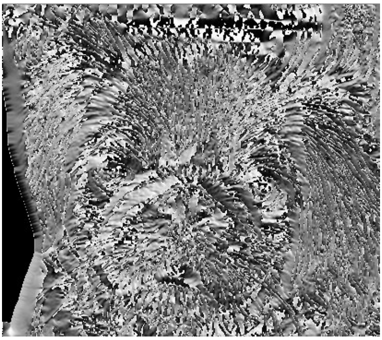
Compute the Gradient Magnitude and Strength

After getting the x and y gradient components of the image. We can easily find the magnitude and direction of the gradient. By this formula

Gradient-magnitude = $\sqrt{Fx^2 + Fy^2}$

Gradient-direction = $\arctan(\frac{Fy}{Fx})$

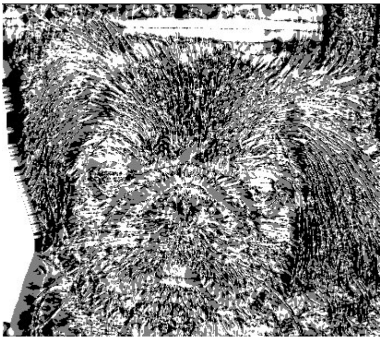
(a) Gradient magnitude



(b) Gradient Direction

Non Maximal Supression(NMS)

Before applying NMS. We select four fundamental orientation of gradient for the pixels *i.e* (0,π/4,π/2, 3π/4) .We need to find for each pixel's gradient orientation (Which we already found in the above step), to which fundamental orientation does it closely corresponds to[5]. After that simply check for each pixel, if the edge strength is greater than atleast one of its neighbors in that direction/orientation. If not set the pixel intensity to zero otherwise simply let that pixel intensity remain. See Algorithm 1 NMS.



(c) Discrete fundamental Gradient orientation

(d) Non Maximal Supression

[5] We want our gradient values to be positive,simple and discrete

Hysteresis Thresholding

Now according to our needs and which is suitable for the image, define two Threshold values T_ high and T_low. If the Value of the pixel intensity is higher than T_ high then that is said to be part of a **strong edge**. And those pixels which are less than T_ high but more than T_ low are considered **weak edges** of the image. But those smaller than both aren't considered as edge pixels at all.

Now simply track which weak edge pixels are connected to the strong edge pixels like chain. Only those are to be considered as "edges". The rest will then no longer be considered edges[6]. See Algorithm 2 Hysteresis.

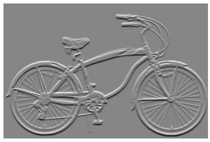
(e) Double Hysteresis Thresholding

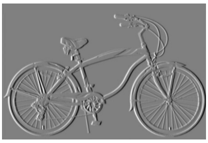
Figure 3: Final output of Canny Edge Detection

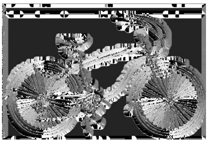
[6]They could potentially be noise

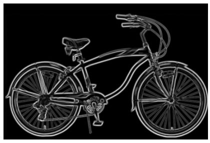

(a) Original image

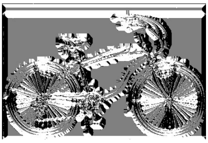

(b) Gray scale image

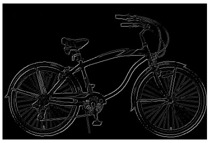

(c) Fy


(d) Fx


(e) Orientation (D)


(f) Gradient magnitude(F)


(g) Discrete Orientation (D*)


(h) Edge NMS

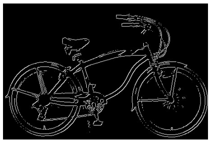

(i) Final Output

Figure 4: Intermediate stages of Canny Edge Detection for bicycle image

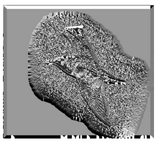

(a) Original image

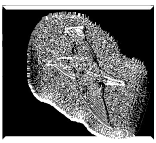

(b) Gray scale image

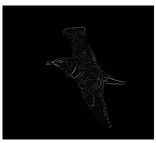

(c) Fy


(d) Fx


(e) Gradient magnitude


(f) Orientation


(g) Discrete Orientation (D*)


(h) Edge NMS

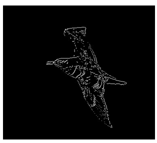
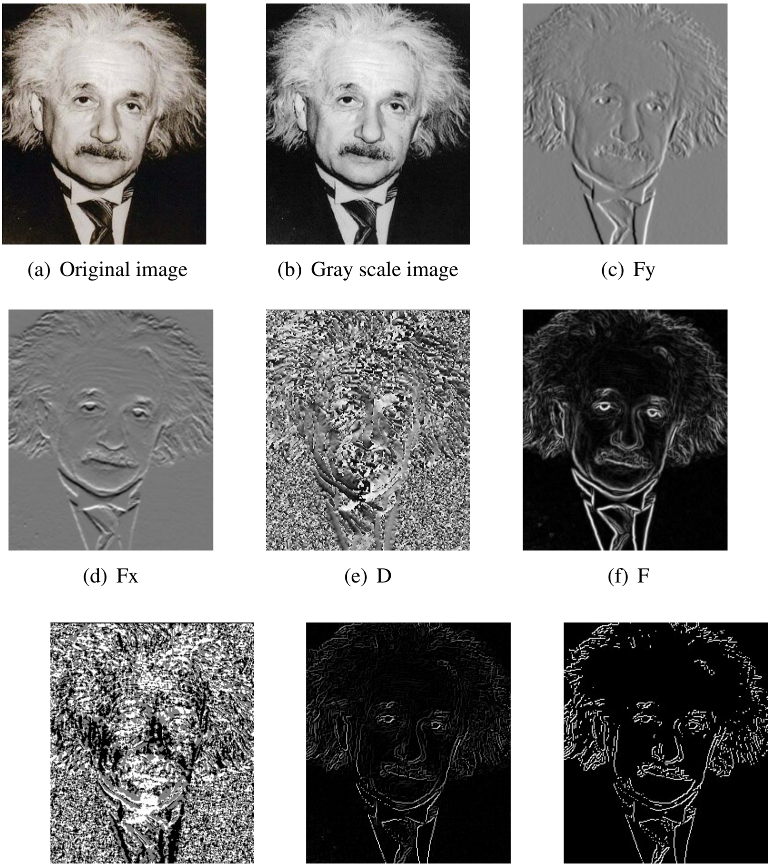

(i) Final Output

Figure 5: Intermediate stages of Canny Edge Detection for Bird image



(a) Original image (b) Gray scale image (c) Fy

(d) Fx (e) D (f) F

(g) D* (h) Edge NMS (i) Final Output

Figure 6: Einstein

Algorithm 1 NMS

Gradient Magnitude = F ▷ F and D are matrices storing the respective value for each pixel
Discrete Gradient Direction= D
M is the length of the image
N is the width of the image
Matrix with same dimension of F = I ▷ This will store the NMS output (Initially all values are zero)

```
for i ← 1 to N-1 do
    for j ← 1 to M-1 do
        d ← D|i,j|
        p1,p2 ← 1,1                ▷ These are adjacent pixels which are in the d direction
        if d==0 then
            p1,p2← F|i,j+1|,F|i,j-1|
        else if d==pi/2 then
            p1,p2← F|i+1,j|,F|i-1,j|
        else if d==pi/4 then
            p1,p2← F|i+1,j+1|,F|i-1,j-1|
        else if d==3pi/4 then
            p1,p2← F|i+1,j-1|,F|i-1,j+1|
        end if
        if F|i,j|>p1 and F|i,j|>p2 then
            I|i,j|=F|i,j|
        end if
    end for
```

Algorithm 2 Hysteresis

Low Threshold value = T_low
High Threshold value = T_high
Indicator matrix to store which pixels are strong edged, weak edged or are not an edge pixel at all

Require: Matrix I From Algorithm 1

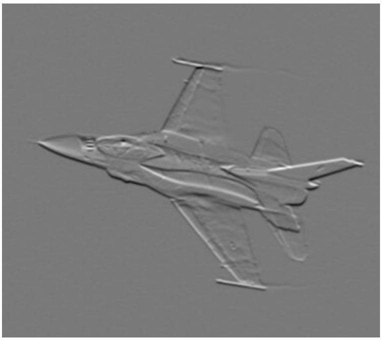
```
for each element p ∈ I do                                        ▷ i,j is position of the pixel p
    if p ≥ T_high then
        Indicator|i,j| ← 2                                    ▷ 2 indicates strong edged pixel
    else if p ≥ T_low then
        Indicator|i,j| ← 1                                      ▷ 1 indicates weak edged pixel
    else
        Indicator|i,j| ← 0                                        ▷ 0 indicates no edge pixel
    end if
end for
for each element p ∈ Indicator do
    if p is a Weak edged pixel and connected to one of the strong edged pixels then
        Indicator |i,j| ←2
    end if
end for
for each element p ∈ Indicator do
    if Indicator|i,j|≠2 then
        Indicator|i,j| ← 0      ▷ The pixels which are not labelled strong after the previous
"for" loop are either not edge or not connected to strong edge pixels
    end if
end for
                                                                                          =0
```

Harris corner Detection is of 3 steps:
(Refer to Algorithm 3 Harris corner detection)

Compute Gradients of both components
Similarly to the Canny Edge detector compute Fx and Fy[7].

(a) Original image

(b) Gray scale image



(c) Fx

(d) Fy

Figure 7: Results after first step

Find corners
Corners are where there are 2 dominant directions where intensity of pixel changes *i.e.* high gradient. Choose a window of size 2m+1 $\times$ 2m+1 and for each pixel look in that window. Here we have used m=4. Accumulate over the window to find the covariance matrix C ,for each pixel which will contain the average of Fx and Fy in that window.
As depicted here:

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

where u and v are the coordinates within the window $\in$ (-m,m) *i.e* -4 to 4. and Fx, Fy are the gradients of the pixel at the location x+u,y+v. Where x,y is the location of the central

[7] Gradient of the gaussian filter then covolve it with the image

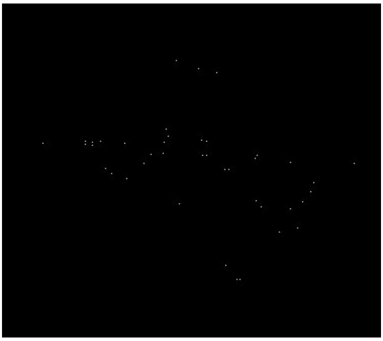
pixel *i.e.* over which the window's response is being recorded.

Then Compute the 2 eigen values of the matrix and find the minimum of the 2. Set that minimum value to e .Set a relevant Threshold value **T**. If e $\geq$ T save that into a list **L**. These pixels represent the corners of the image.

Figure 8: Corners of the image

NMS

Sort the List **L** in decreasing order of their corresponding e. Then simply remove the neighboring 8 pixels which occur later in the List L for each pixel.[8]

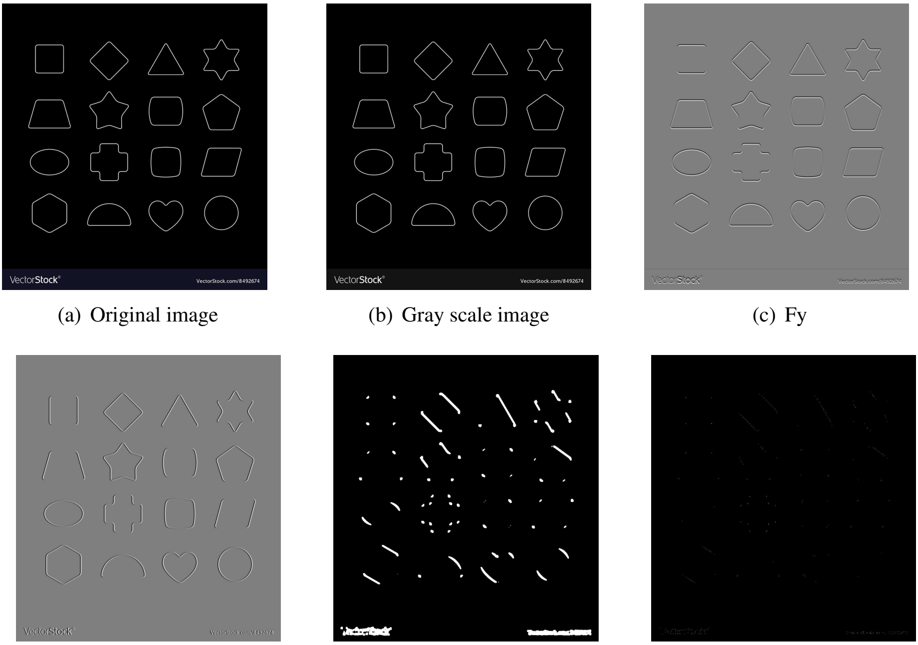


(a) Corners after NMS

(b) Superimposition of the NMS corner image

Figure 9: Final Result of Harris corner detection output

[8] You will have to make a stack to store the elements you want to delete before instantly deleting them from L, to delete all the neighboring pixels



(a) Original image (b) Gray scale image (c) Fy

(d) Fx (e) Corners (f) Final Output

Figure 10: Some more Results

Algorithm 3 Harris corner detection

Require: $F_x, F_y, T(Thresholdvalue)$
  L = [] ▷ Empty list which will store all the corner pixels
  for each pixel p $\in$ image **do** ▷ x,y is position of the pixel p in the image
    $F_x^2, F_y^2, F_xF_y \leftarrow$ 0,0,0
    for u $\leftarrow$ -m to m **do**
      for v $\leftarrow$ -m to m **do**
        $F_x^2 \leftarrow F_x^2 + F_x|x+u, y+v|_x|x+u, y+v|$
        $F_y^2 \leftarrow F_y^2 + F_y|x+u, y+v|_y|x+u, y+v|$
        $F_xF_y \leftarrow F_xF_y + F_x|x+u, y+v|_y|x+u, y+v|$
      end for
    end for
    C = $\begin{bmatrix} F_x^2/(2m+1)^2 & F_xF_y/(2m+1)^2 \\ F_xF_y/(2m+1)^2 & F_y^2/(2m+1)^2 \end{bmatrix}$
    $e \leftarrow$ Minimum of the 2 eigen values of C
    if $e \geq T$ **then**
      add p to L
    end if
  end for
  Sort L in decreasing order of their corresponding e values.
  Stack = [] ▷ This will store the pixels to be deleted from L
  for each pixel p $\in$ L **do**
    if any of the 8 neighbouring pixels of p occur later in the list L **then**
      add those neighbouring pixels to Stack ▷ Make sure each neighbouring pixel is added to the stack only once
    end if
  end for
  for each pixel p$\in$ Stack **do**
    Delete p that in the list L
  end for

3 Results and Observations

Canny Edge Detector:

- If we choose a very high T_high value then none of the edge pixels will be labelled as strong edged pixels. As well as if we choose T_low too low then we will have no change in the output.[9]
- If we pick the correct range *i.e.* T_low and T_high. we will get an output depicting only the relevant edges.

Harris Corner Detector:

- Similar to Canny Edge Detector. If we choose the T value to be too high, no value gets added to list L. And if too low we wont get any useful information.
- Since the corners are too thick to tell where exactly is the corner occurring, after NMS we get a more precise output.

4 Conclusion and Takeaway

Canny Edge Detector:

- Balance of threshold values is important, as we don't want to lose vital information of the image.
- In order to avoid to remove the "thick" lines, we apply NMS, which will reduce the width of the lines to one pixel.
- We can apply a gradient filter like sobel, laplacian etc, and perhaps get a better result.

Harris Corner Detector:

- The windows dimensions are important when determining the corners. The algorithm is susceptible to scale change.
- We need to determine by experimentation, the appropriate threshold value in order to ignore noise and not lose vital information present in the image.

References

[1] Wikipedia. Canny edge detector. *en.wikipedia.org/wiki/Canny*$_{e}dge_{d}etector$, .

[2] Wikipedia. Harris corner detector. *en.wikipedia.org/wiki/Harris*$_{C}orner_{D}etector$, .

[9] The noise will also come in the output, rendering the hysteresis step useless.