Implementing Canny Edge Detector and Harris Corner Detector

Harshit Malik
2017csb1078@iitrpr.ac.in

Indian Institute of Technolgy
Ropar, Punjab
India

Abstract

Edges and Corners plays a significant role in many Image Feature Extraction Algorithms and Computer Vision Applications. So it becomes essentional to detect corners and edges effeciently in an image. This paper presents the implemention of two such algorithm- **Canny Edge Detection** and **Harris Corner Detection**.

1 Canny Edge Detector

1.1 Introduction

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect edges in images. The stages involved are - smoothening, calculating gradients, nonmaximum suppression and thresholding with hysterysis. Canny Edge detector is based on the fact the sudden intensity changes accross the edge but however intensity does not change along the edge.

1.2 Methodology

1.2.1 Preprocessing and calculating smoothed gradients

First step is converting RGB image to gray scale. Edge detectors are prone to noise, so to avoid noise, input image is smoothened using gaussian filters. Now, gradient component Ix and Iy along x and y direction respectively are calculated at every single point in the image. But, to save compution, we can merge these two steps into one by convolving with the x and y derivatives of the Gaussian. This is done by convolving with standard 3*3 sobel filter. Now magnitude and direction of gradients are calculated as follow:

$$G = \sqrt{Ix^2 + Iy^2}$$
$$\theta = arctan(\frac{Iy}{Ix})$$

The magnitude of the gradient at a point determines if it possibly lies on an edge or not. A high gradient magnitude means the sudden intensity change - implying an edge. A low gradient implies no substantial changes. So it's not an edge. The direction of the gradient shows how the edge is oriented. Direction of edge is normal to the gradient direction.

Algorithm 1 convolveWithGaussianDerivative(img)

$sobel\_filter\_x \leftarrow \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 2 \end{bmatrix}$

$sobel\_filter\_y \leftarrow \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

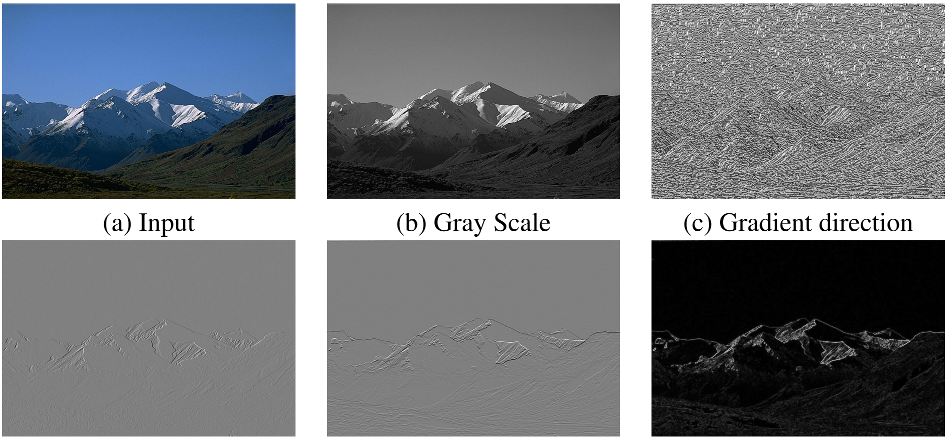
$I_x \leftarrow sobel\_filter\_x \star img$

$I_y \leftarrow sobel\_filter\_y \star img$

$G \leftarrow \sqrt{Ix^2 + Iy^2}$

$\theta \leftarrow arctan(\frac{Iy}{Ix})$

return G, θ



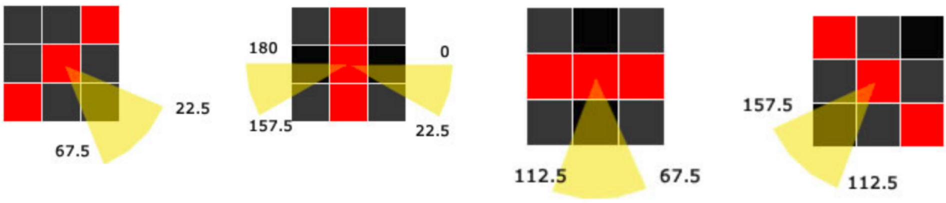
(a) Input (b) Gray Scale (c) Gradient direction

(d) X Gradient (e) Y Gradient (f) Gradient Magnitude

Figure 1: Mountains image gradient detection. Fig 1.(c) white region shows angle close to 180 degree and black region close to 0 degree.

1.2.2 Non-Maximum Suppression

This steps basically suppresses a pixel if it is not maximum along the gradient direction and this reducing the width of edges to one pixel. To perform this, gradient direction at each pixel is quantazied to four bins - 0°, 45°, 90° and 135°. And along these four direction lies exactly three pixels in the neighbourhood of a pixel and out of these three pixels, one with the maximum intensity is included in the edge, other two are suppressed to zero intensity. After this step we will get image with thin edges.

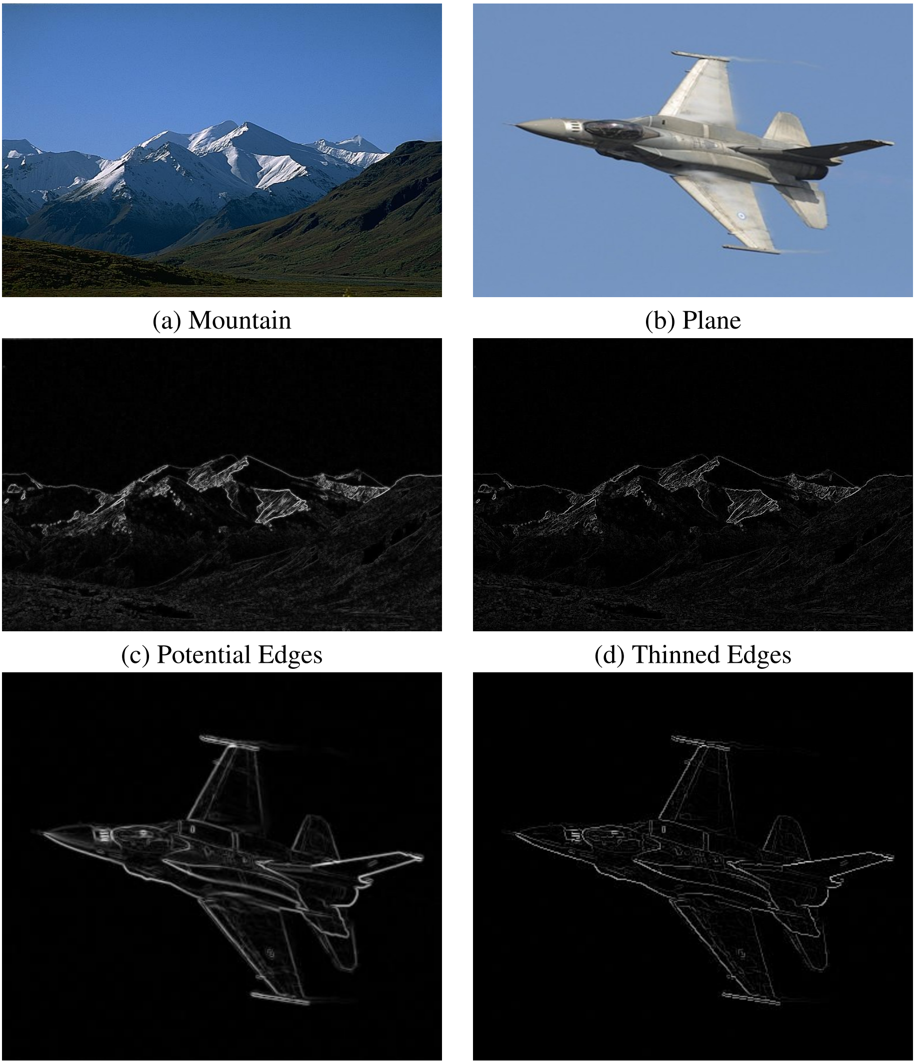


(a) (b) (c) (d)

Figure 2: These figures represent dividing gradient angle into bins and associated directed neighbourhood pixels. Red pixels represent edge direction and yellow region represent gradient direction. Black pixels represent pixel that falls along the gradient direction and thenon-maximum black pixels are suppresed to get the single red pixels for edges.

Algorithm 2 nonMaxSuppression(img, D)

$M \leftarrow img.height$
$N \leftarrow img.width$
$angle\_mat \leftarrow (D * 180)/\pi$
if $angle\_mat[x,y] < 0$ **then**
    $angle\_mat[x,y] \leftarrow angle\_mat[x,y] + 180$
end if
$i \leftarrow 1$
$j \leftarrow 1$
for $i < M$ **do**
    for $j < N$ **do**
        if $angle\_mat[x,y] \in 0°$ **then**
            $x \leftarrow img[i, j+1]$
            $y \leftarrow img[i, j-1]$
        else if $angle\_mat[x,y] \in 45°$ **then**
            $x \leftarrow img[i+1, j-1]$
            $y \leftarrow img[i-1, j+1]$
        else if $angle\_mat[x,y] \in 90°$ **then**
            $x \leftarrow img[i+1, j]$
            $y \leftarrow img[i-1, j]$
        else if $angle\_mat[x,y] \in 135°$ **then**
            $x \leftarrow img[i-1, j-1]$
            $y \leftarrow img[i+1, j+1]$
        end if
        if $img[i, j] > x, y$ **then**
            $result[i, j] = img[i, j]$
        else
            $result[i, j] = 0$
        end if
    end for
end for
return G, *result*



(a) Mountain (b) Plane

(c) Potential Edges (d) Thinned Edges

(e) Potential Edges (f) Thinned Edges

Figure 3: Figures depicting thinning of edges.

1.2.3 Thresholding with hysterysis

Now, we need mark pixels as edges based on some threshold on gradient magnitude. Two threshold T_{low} and T_{high} are picked. Iterate over whole image and mark edges as follow: if $G < T_{low}$, mark it as *'not edge'*, if $T_{low} < G < T_{high}$, mark it as *'weak edge'* and if $G > T_{high}$ as *'strong edge'*. Now strong edges are detected edges and we classify any weak edge as edge if any pixel in its neighbourhood is marked as strong edge.

Algorithm 3 $hystersisThresholding(G, T_{low}, T_{high})$

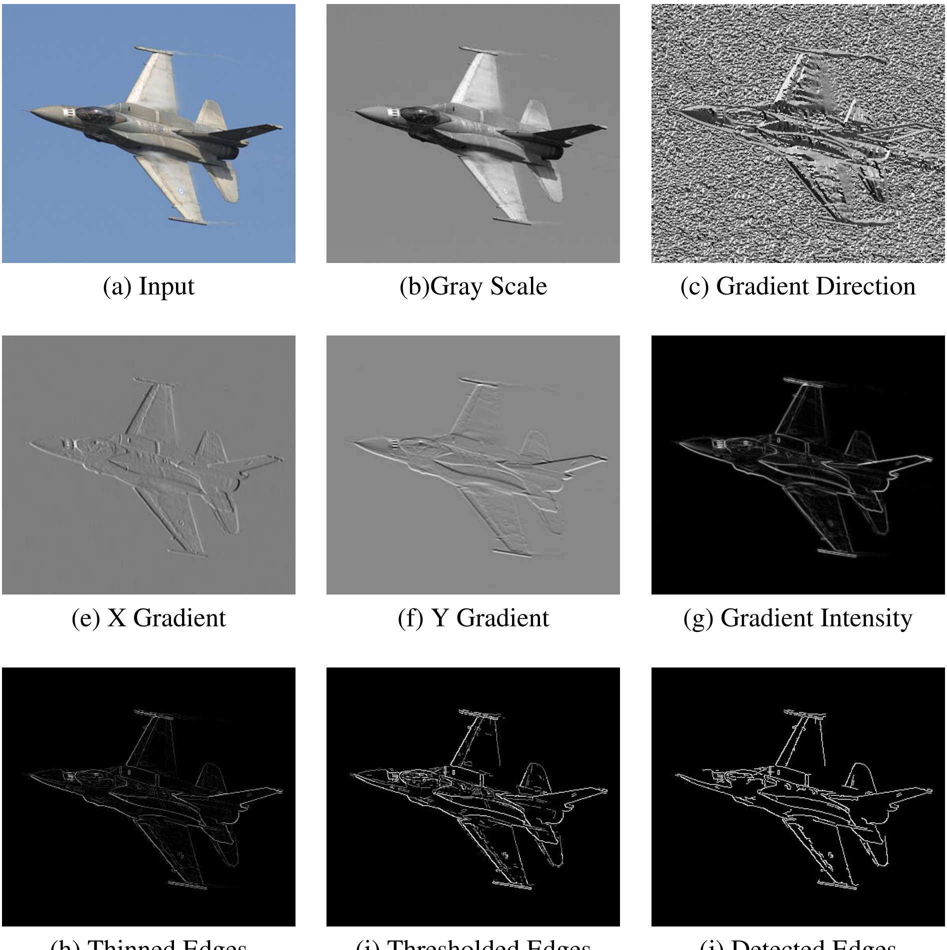
```
M ← G.height
N ← G.width
i ← 0
j ← 0
for i < M do
    for j < N do
        if G[x,y] >= T_high then
            output ← 1.0
        else if T_low <= G[x,y] < T_high then
            output[i, j] ← 0.5
        else
            output[i, j] ← 0.0
        end if
    end for
end for
i ← 0
j ← 0
for i < M do
    for j < N do
        if output[i, j] = 0.5 then
            if output[i ± 1, j ± 1] = 1.0 then
                output[i, j] ← 1.0
            else
                output[i, j] ← 0.0
            end if
        end if
    end for
end for
return output
```

(a) Thinned edges (b)Thresholding (c) Detected edges

Figure 4: Figures depicting hysterysis thresholding.

1.3 Observations and results

We ran Canny Edge Detector for all the six images - *bird.bmp*, *plane.bmp*, *dog.bmp*, *einstein.bmp*, *bicycle.bmp* and *toy_image.jpg* provided in the data folder, and the following outputs are obtained. With proper thresholds choosen, edges are detected quite accurately. More on choosing threshold is discussed in section 1.3.1 . Below figures from 5 to 10 shows the output image with all intermediate results. In figure (c), white region shows angle close to 180 degree and black region close to 0 degree.



(a) Input (b)Gray Scale (c) Gradient Direction

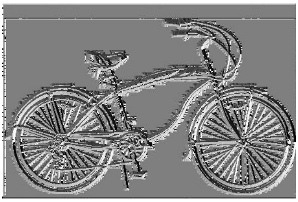
(e) X Gradient (f) Y Gradient (g) Gradient Intensity

(h) Thinned Edges (i) Thresholded Edges (j) Detected Edges

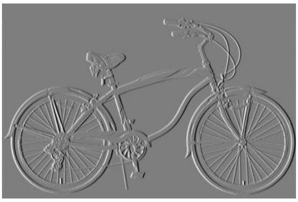
Figure 5: Plan Image

(a) Input

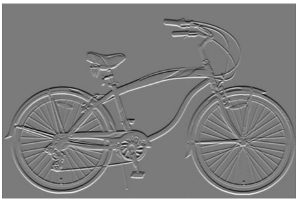
(b)Gray Scale



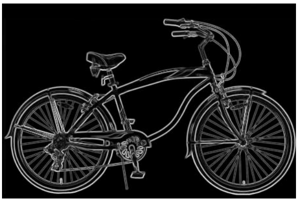
(c) Gradient Direction



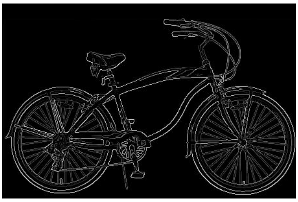
(e) X Gradient



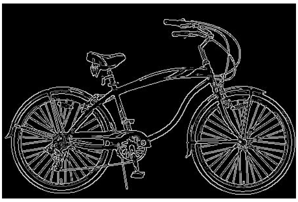
(f) Y Gradient



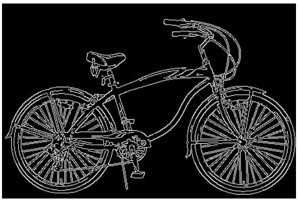
(g) Gradient Intensity



(h) Thinned Edges

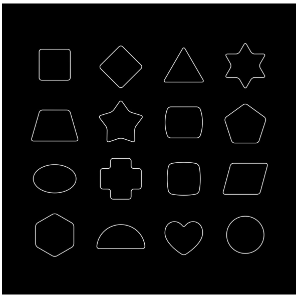


(i) Thresholded Edges

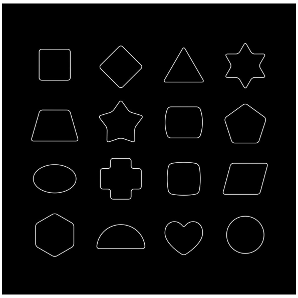


(j) Detected Edges

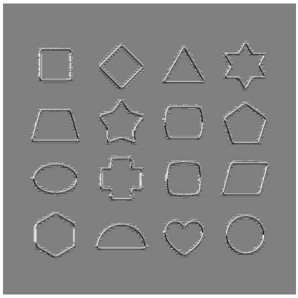
Figure 6: Bicycle Image



(a) Input

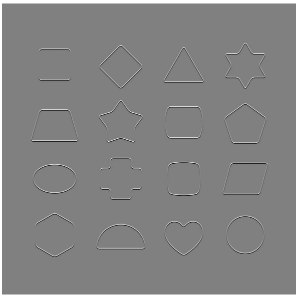


(b)Gray Scale

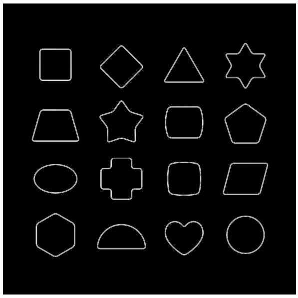


(c) Gradient Direction

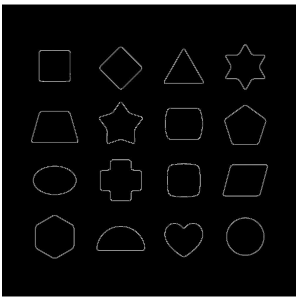
(e) X Gradient



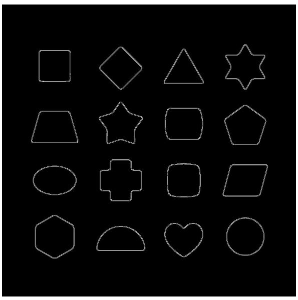
(f) Y Gradient



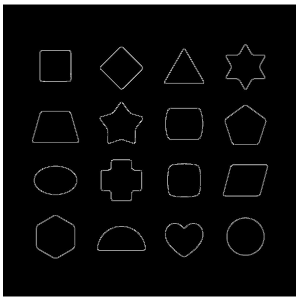
(g) Gradient Intensity



(h) Thinned Edges

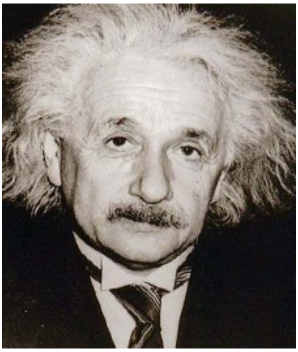


(i) Thresholded Edges

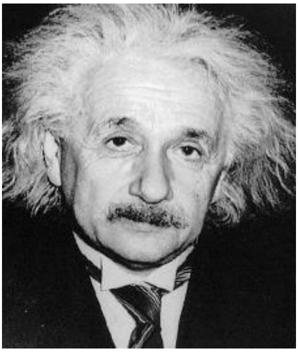


(j) Detected Edges

Figure 7: Toy Image

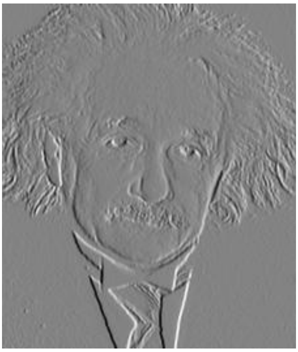


(a) Input

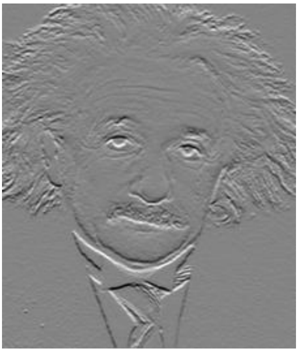


(b)Gray Scale

(c) Gradient Direction

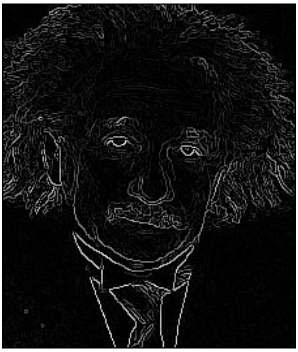


(e) X Gradient

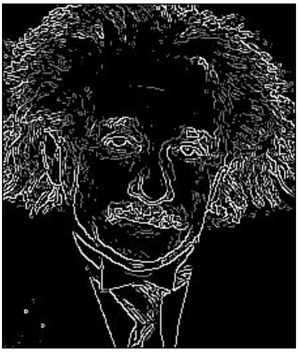


(f) Y Gradient

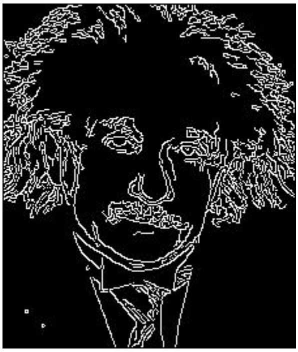
(g) Gradient Intensity



(h) Thinned Edges

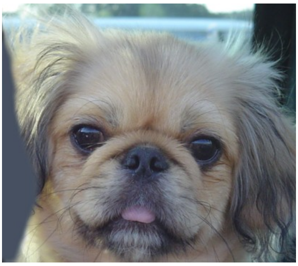


(i) Thresholded Edges



(j) Detected Edges

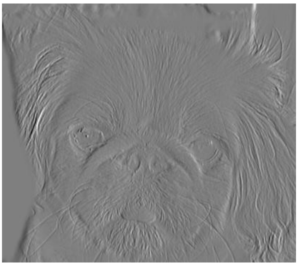
Figure 8: Einstein Image



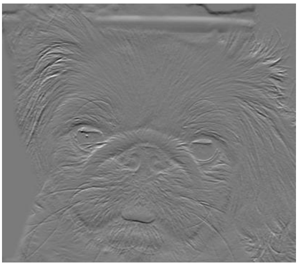
(a) Input

(b)Gray Scale

(c) Gradient Direction

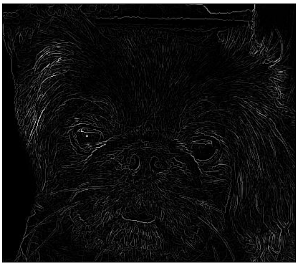


(e) X Gradient



(f) Y Gradient

(g) Gradient Intensity

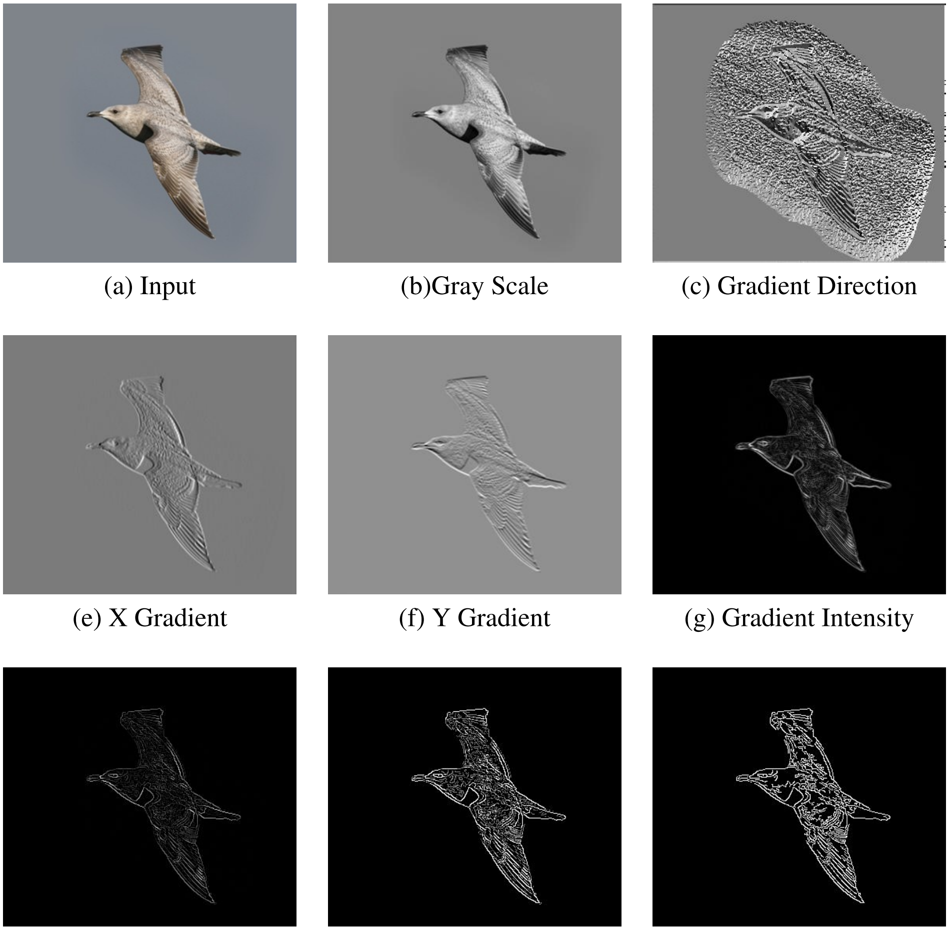


(h) Thinned Edges

(i) Thresholded Edges

(j) Detected Edges

Figure 9: Dog Image



(a) Input (b)Gray Scale (c) Gradient Direction

(e) X Gradient (f) Y Gradient (g) Gradient Intensity

(h) Thinned Edges (i) Thresholded Edges (j) Detected Edges

Figure 10: Bird Image

1.3.1 Thresholding Parameters

Canny Edge detector gives you freedom to select two independent parameters T_{low} and T_{high} to classify each pixel as not edge, weak edge and strong edge. Accuracy of this alogorithm depends largerly on these parameters value. If you select T_{low} large, many potential edges will be left out and if you select T_{high} small, many spurious edges will be detected due to noise in the image. It is recommended to select T_{low} small and T_{high} large so that there is significant difference between T_{low} and T_{high} so that many edges marked as weak. During hysterisis, only connnected edges will be marked as strong and spurious and wanted edges due to noise will be removed. Below image shows output for various threshold values.

(a) $T_{low} = 0.1, T_{high} = 0.5$

(b) $T_{low} = 0.25, T_{high} = 0.5$

(c) $T_{low} = 0.1, T_{high} = 0.25$

(e) $T_{low} = 0.2, T_{high} = 0.3$

(f) $T_{low} = 0.1, T_{high} = 0.1$

(g) $T_{low} = 0.5, T_{high} = 0.5$

Figure 11: Dog Image. Depecting the effects of changing threshold paramaters.

1.4 Insights and Take Aways

- Canny edge detector is invariant to rotation. Gradient magnitude remians same, only the gradient direction changes.

- Canny edger detector is invariant to linear transalation. Gradient magnitude and direction, both remian same.

- Canny edge detector is invaraint to intensity shift as first derivative is involved which remains same on linear shift.

- Canny edge detector is not invariant to scaling. Relative neighbourhood intensity shifts change on scaling and thus the gradient magnitude and direction.

- Canny edge detector is vulnerable to noise in the image. Gaussian smoothing is used to remove noise effects on edge detection. σ needs to be picked accordingly to remove niose effects and to avoid spurios and unwanted edge detection.

- Canny edge detector depends on two threshold parameter- T_{low} and T_{high}, which in a way control the accuracy and precision of the algorithm. These parameter varies significantly for different images. T_{low} avoids unwanted edge detection, T_{high} ensures no spurious edge detection due to noise and difference between T_{high} and T_{low} ensures only connected edge detection. See Section 1.3.1 .

2 Harris Corner Detection

2.1 Introduction

Harris Corner Detector is a corner detection operator that is commonly used in computer vision algorithms to extract corners and infer interest point of an image. Harris corner detector is based on the fact that around a corner, intensity changes along every direction. This is captured by considereng a window around a point and observing in intensity changes by moving window accross different direction.

2.2 Methodology

2.2.1 calculating smoothed gradients

Gradients, magnitude and direction, are calculated in the simalar way as mentioned in section 1.2.1 i.e by convolving 3*3 sobel filter.

Algorithm 4 smoothedGradients(img)

1: $sobel\_filter\_x \leftarrow \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 2 \end{bmatrix}$
2: $sobel\_filter\_y \leftarrow \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
3: $I_x \leftarrow sobel\_filter\_x \star img$
4: $I_y \leftarrow sobel\_filter\_y \star img$
5: $G \leftarrow \sqrt{Ix^2 + Iy^2}$
6: $\theta \leftarrow arctan(\frac{Iy}{Ix})$
7: **return** G, θ

2.2.2 Finding corners

For each pixel (x, y), look in a window of size 2m + 1 * 2m + 1 around the pixel (m = 4 used). Accumulate over this window the covariance matrix C, which contains the average of the products of x and y gradients as follow :

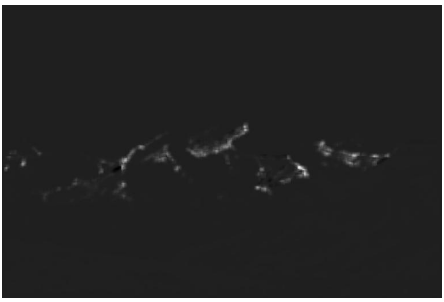
$$C = \frac{1}{2m+1} \sum_u \sum_v \begin{bmatrix} F_x^2 & F_x F_y \\ F_x F_y & F_y^2 \end{bmatrix} = \begin{bmatrix} < F_x^2 > & < F_x F_y > \\ < F_x F_y > & < F_y^2 > \end{bmatrix}$$

Here (u, v) are the coordinates within the window: $u = -m, \ldots, m$, and $v = -m, \ldots, m$, the brackets (< and >) denote a dot product operation. Now for each point calculate r score as follow: $r = Determinant(C) - k(Trace(C))^2$, where k is a small constant (k=0.04 used). Take a threshold T and and mark all points with r score > T as corner.

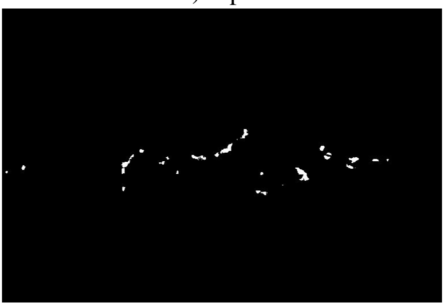
Algorithm 5 $getCorners(img, I_x, I_y, m, k, T)$

Declare empty corner list L
$M \leftarrow img.height$
$N \leftarrow img.width$
$I_{xx} \leftarrow I_x^2$
$I_{xy} \leftarrow I_x * I_y$
$I_{yy} \leftarrow I_y^2$
$i \leftarrow 0$
$j \leftarrow 0$
for $i < M$ **do**
  for $j < N$ **do**
    $S_{xx} \leftarrow sum(I_{xx}[y-m : y+m+1, x-m : x+m+1])$
    $S_{xy} \leftarrow sum(I_{xy}[y-m : y+m+1, x-m : x+m+1])$
    $S_{yy} \leftarrow sum(I_{yy}[y-m : y+m+1, x-m : x+m+1])$
    $det \leftarrow (S_{xx} * S_{yy}) - (S_{xy}^2)$
    $trace \leftarrow S_{xx} + S_{yy}$
    $r \leftarrow det - k * trace$
    if $r \geq T$ **then**
      add (i,j,r) to L
    end if
  end for
end for
return L

a) Input



(b) R values



(c) Corner points

(d) Corners

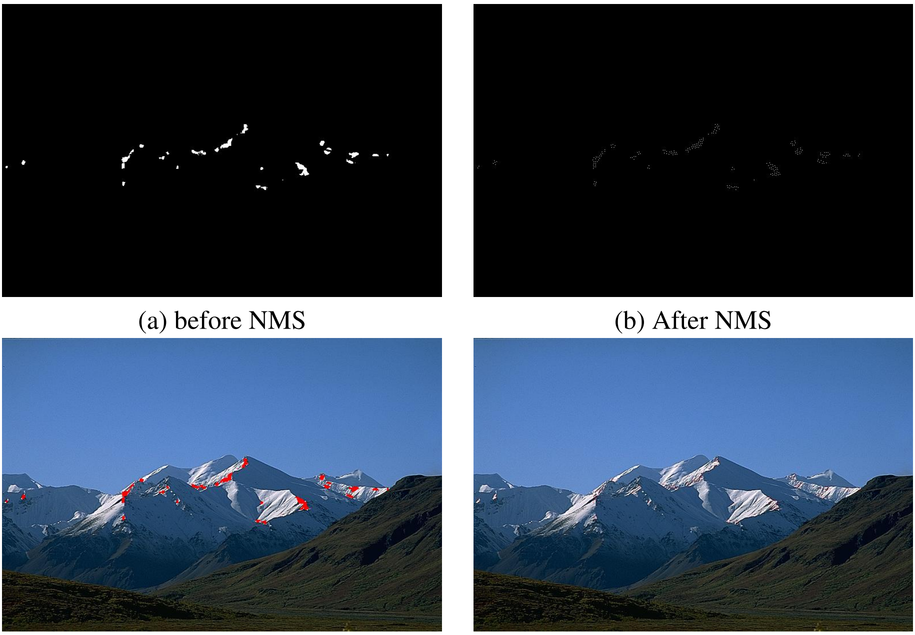
Figure 12: Figures depicting corner response and r values for each pixels.

2.2.3 Non-maximum Suppression

Reduce the thickness of corners using non-maximum suppression technique. Store all the detected corners along with their r score in a list. Sort this list based on r score. Now iterate this list from starting, for each point p, remove all points in the 8-connected neighborhood of p that occur later in the list.

Algorithm 6 $nonMaximalSuppression(L)$

Declare empty corner list S
$S \leftarrow L.sort\_on\_r()$
$i \leftarrow S.begin()$
for $i \neq S.end()$ **do**
    if $S.find([i.x \pm 1, i.y \pm 1])$ **then**
        $S.remove([i.x \pm 1, i.y \pm 1])$
    end if
end for
return S



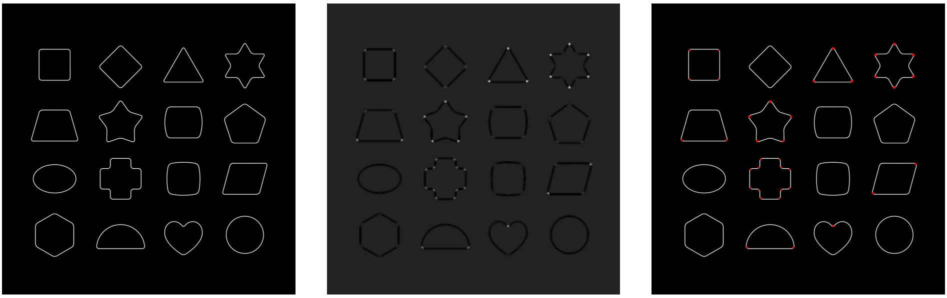
(a) before NMS (b) After NMS

(c) before NMS (d) After NMS

Figure 13: Figure depecting action of non-maximum supression algorithm.

2.3 Observations and Results

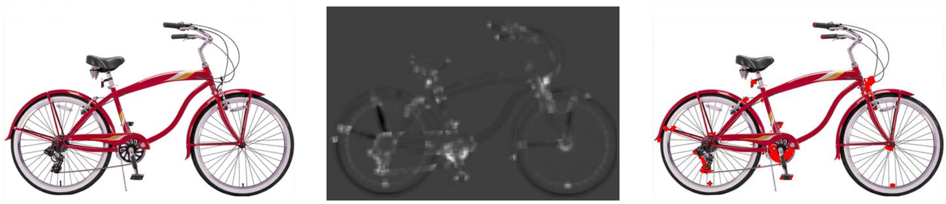
We ran Harris Corner Detector for all the six images - $bird.bmp$, $plane.bmp$, $dog.bmp$, $einstein.bmp$, $bicycle.bmp$ and $toy\_image.jpg$ provided in data folder, and the following outputs are obtained. With proper thresholds choosen, corners are detected quite fairly. Below figures from 14 to 19 shows the output image with all intermediate results.



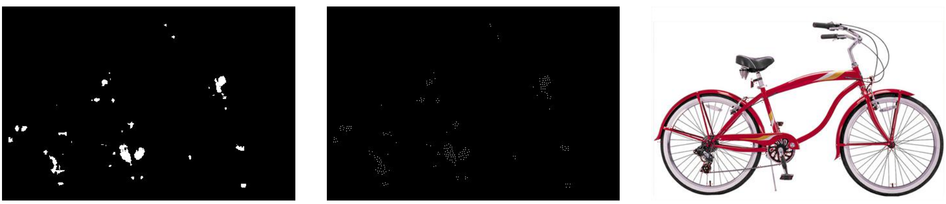
(a) Input (b) R Values (c) Corners

(a) Before NMS (b) After NMS (c) After NMS

Figure 14: Toy Image

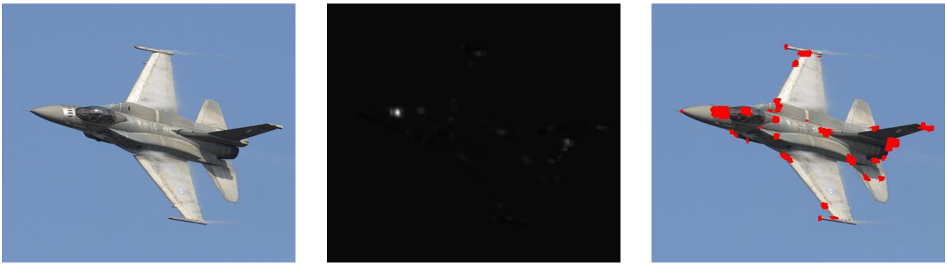


(a) Input (b) R Values (c) Corners

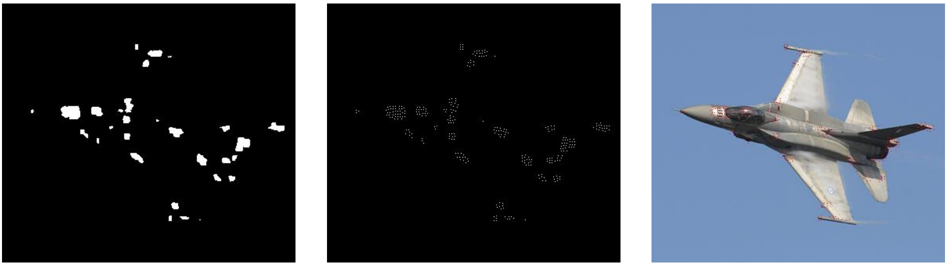


(a) Before NMS (b) After NMS (c) After NMS

Figure 15: Bicycle Image

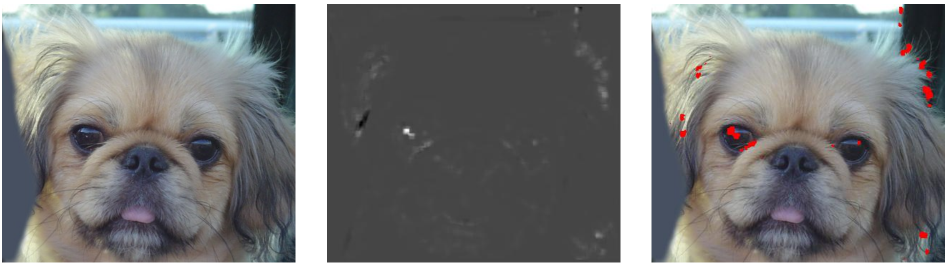


(a) Input (b) R Values (c) Corners

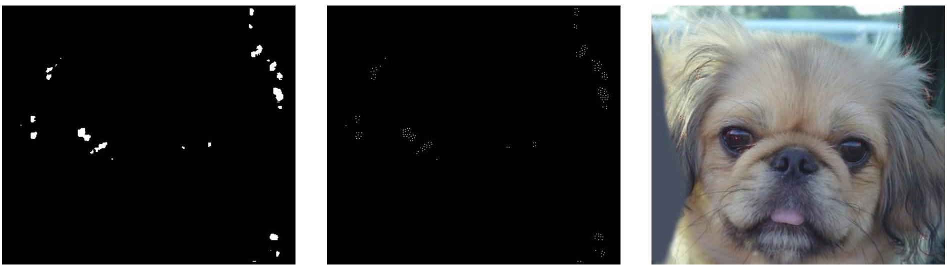


(a) Before NMS (b) After NMS (c) After NMS

Figure 16: Plane Image



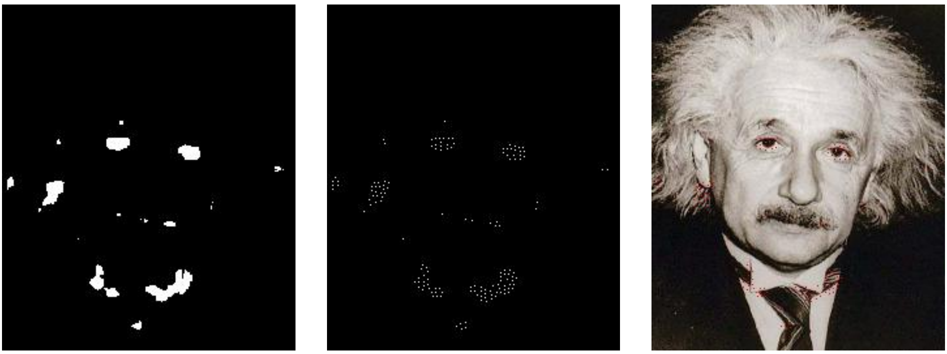
(a) Input (b) R Values (c) Corners



(a) Before NMS (b) After NMS (c) After NMS

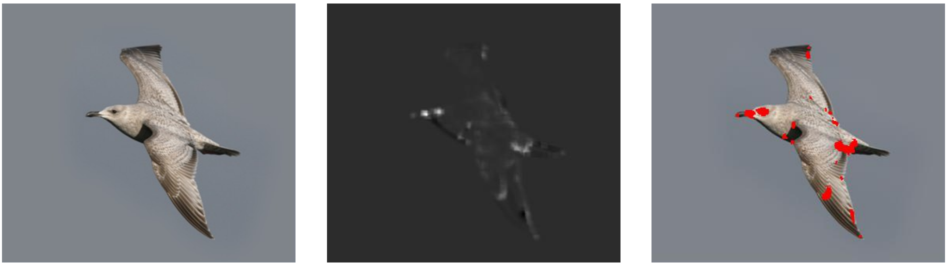
Figure 17: Dog Image

(a) Input (b) R Values (c) Corners

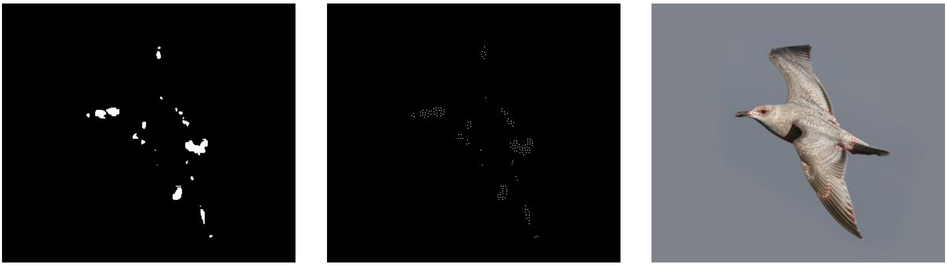


(a) Before NMS (b) After NMS (c) After NMS

Figure 18: Einstein Image



(a) Input (b) R Values (c) Corners

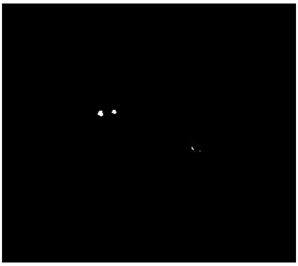


(a) Before NMS (b) After NMS (c) After NMS

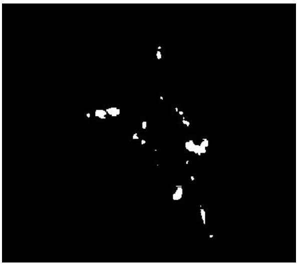
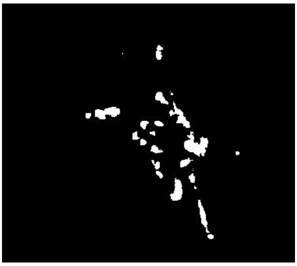
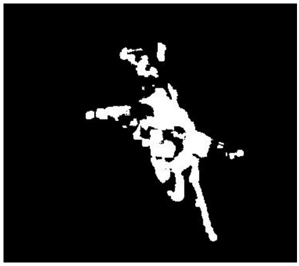
Figure 19: Bird Image

2.3.1 Thresholding Parameter

Harris Corner detector gives you freedom to select a independent thresholding parameters T to classify each pixel into corner and not corner. Accuracy of this alogorithm depends largerly on this parameters value. If you select value of T to be small, you will get many unwanted corners due to noise and if T is larger, many soft corners will be ignored. Generally, T is defined $\alpha * r\_score.max()$ as whereα is choosen between 0.001 to 0.2 . T values varies significantly from image to timage, so it is defined relatively with max value. Below images output at various thresholding values.

(a) $T = 0.5$ (b) $T = 0.4$ (c) $T = 0.25$

(e) $T = 0.1$ (f) $T = 0.05$ (g) $T = 0.01$

Figure 20: Bird Image

(a) $T = 0.5$ (b) $T = 0.4$ (c) $T = 0.25$

(e) $T = 0.1$ (f) $T = 0.05$ (g) $T = 0.01$

Figure 21: Bird Image

Threshold Values	Number of Corner Pixels Detected	
	Before NMS	After NMS
T = 0.50	67	12
T = 0.40	126	21
T = 0.25	321	40
T = 0.10	1087	144
T = 0.05	2726	333
T = 0.01	9015	933

Figure 22: Table showing the number of corner pixels detected after thresholdng

2.4 Insights and Take Aways

- Harris corner detector is invariant to rotation. R score of any pixel remains same after rotation.

- Harris corner detector is invariant to linear transalation. R score of any pixel remains same after transalation.

- Harris corner detector is invaraint to intensity shift as first derivative is involved which remains same on linear shift.

- 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.

- Harris corner detector is vulnerable to noise in the image. Gaussian smoothing is used to remove noise effects on corner detection. σ needs to be picked accordingly to remove niose effects.

- 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.

- Apart from edges, R score also gives the information about edges. If R is latge and negative, it shows presence og an edge. See below figure 23.

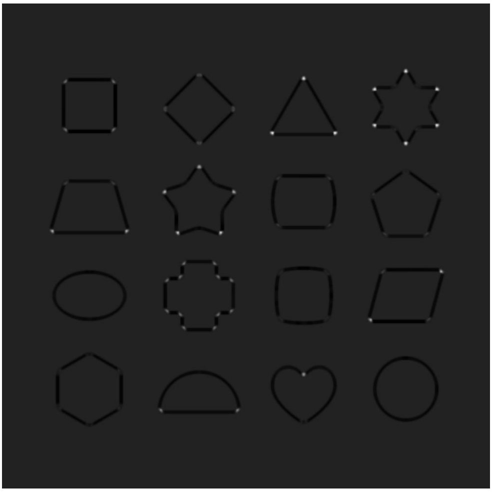


Figure 23: White area represents high positive R value, thus the corners. Black region represents large negative R values, thus the edges. Gray region represnts small absolute R values, thus the flat region.