19CSE413
DIGITAL IMAGE PROCESSING

CB.EN.U4CSE20309
AVANTIKA BALAJI

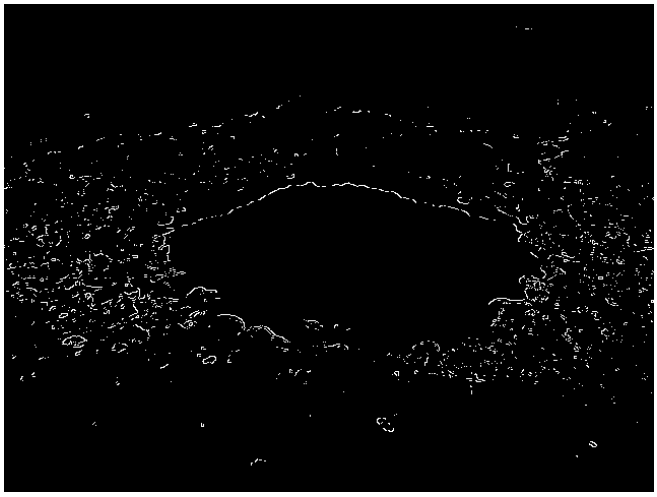
EVALUATION - 3

EDGE DETECTORS

1. CANNY EDGE DETECTOR

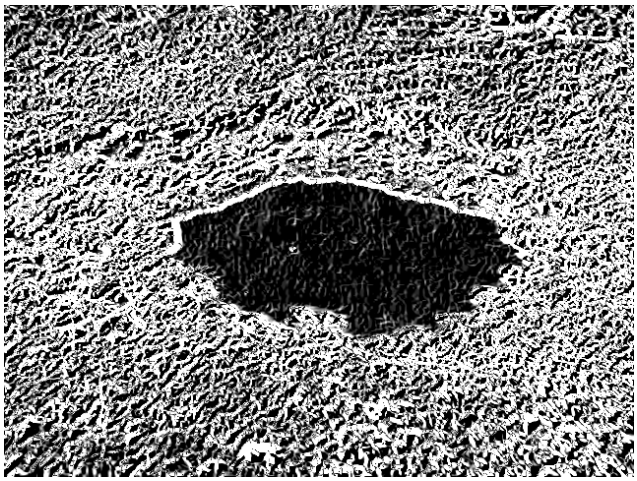
It is a multi-stage algorithm with steps involved:
1. Removal of noise in input image using a Gaussian filter.
2. To find intensity gradient of the image - Compute the derivative of Gaussian filter
3. Suppress the non-max edge contributor pixel points. This way we remove unwanted pixels which is not a part of the edge
4. Use Hysteresis Thresholding method to determine the true edges in the image.

The pothole can be easily distinguished from the background because the optimal detector eliminates the possibility of false positives and false negatives. There is also the advantage of good localisation: the detected edges are as close to true edges as possible.



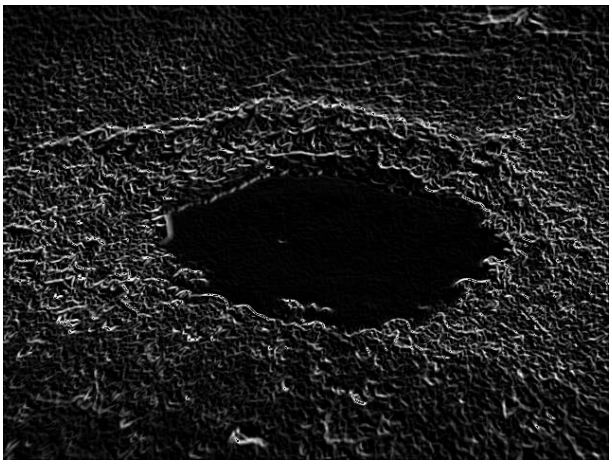
2. SOBEL EDGE DETECTOR

For edge detection, we use matrix to calculate areas of different intensities of an image. And wherever we find extreme differences in the intensities of the pixel, it means we have found the edge of an object. Here the image is processed separately in X and Y directions first, then the sum of the X and Y edges are taken which forms the combined output image. This works best when we first convert the image from RGB to standardised Grayscale since the Luminance plane of grayscale has far more important and distinct edge information. Then we do kernel convolution using a 3x3 matrix. So we basically run a 3x3 matrix kernel over all the pixels in the image. At every iteration, we measure the change in the gradient of the pixels that fall within this 3x3 kernel. The greater the change in pixel intensity, the more significant the edge there is.



3. **PREWITT EDGE DETECTOR**

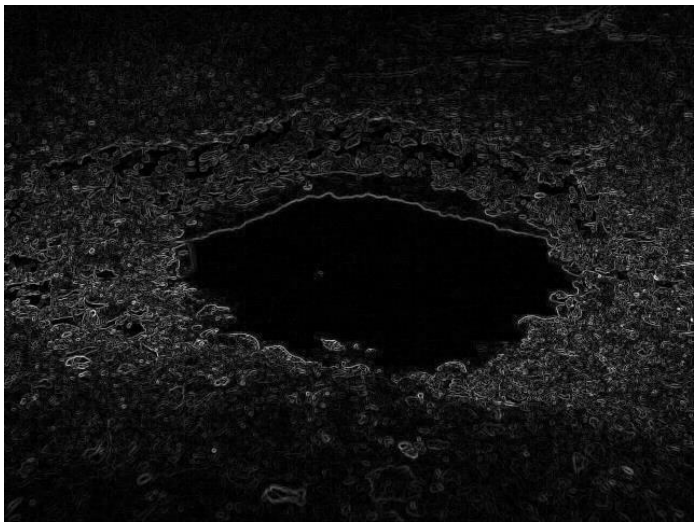
Prewitt mask is a first-order derivative mask. This also works along the X and Y-axis separately. Wherever there is a sudden change in pixel intensities, an edge is detected by the mask. This edge can be calculated by using differentiation. The edge is represented by the local maxima or local minima.



4. **ROBERT EDGE DETECTOR**

It thus highlights regions of high spatial frequency which often correspond to edges. As a differential operator, which approximates the gradient of an image through discrete differentiation which is achieved by computing the sum of the squares of the differences between diagonally adjacent pixels

.

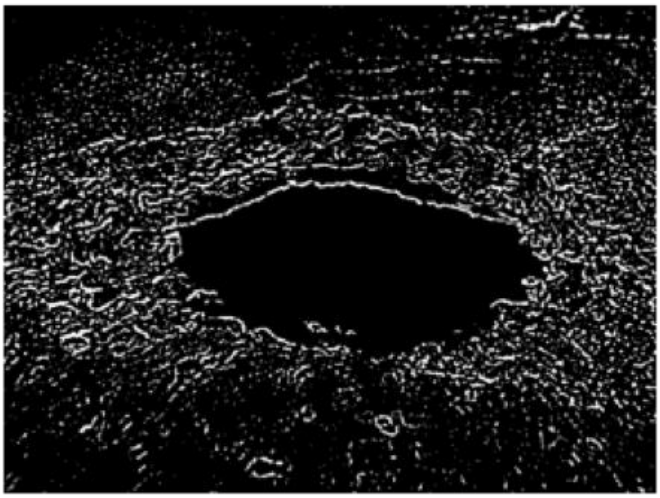


5. **LAPLACIAN OF GAUSSIAN - MARR-HILDRETH EDGE DETECTOR**

In Marr-Hildreth we first, we use a Gaussian filter on the noisy image to smoothen it instead of just simple averaging and then subsequently use the Laplacian filter for edge detection. The image is first filtered with the Laplacian of the Gaussian filter matrix, which is computed using the standard deviation value input for Marr-Hildreth edge detection. The standard deviation

value determines the filter matrix width and controls the amount of smoothing produced by the Gaussian component. The Gaussian filtering's Laplacian will smooth the image and enhance the edges. Edge localization is performed after filtering by locating zero crossings at each pixel in all directions.

LOG



CODE:

```
#ROBERT EDGE DETECTION
from scipy import ndimage
img = cv2.imread("6.jpg",0).astype('float64')
img/=255.0
roberts_cross_v = np.array( [[1, 0 ],
             [0,-1 ]] )

roberts_cross_h = np.array( [[ 0, 1 ],
             [ -1, 0 ]] )

vertical = ndimage.convolve( img, roberts_cross_v )
horizontal = ndimage.convolve( img, roberts_cross_h )

edged_img = np.sqrt( np.square(horizontal) + np.square(vertical))
edged_img*=255
cv2_imshow(edged_img)
```

```
img = cv2.imread('6.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
img_gaussian = cv2.GaussianBlur(img,(3,3),0)

#canny
img_canny = cv2.Canny(img,450,450)



#sobel
img_sobelx = cv2.Sobel(img_gaussian,cv2.CV_8U,1,0,ksize=5)
img_sobely = cv2.Sobel(img_gaussian,cv2.CV_8U,0,1,ksize=5)
img_sobel = img_sobelx + img_sobely


#prewitt
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
img_prewittx = cv2.filter2D(img_gaussian, -1, kernelx)
img_prewitty = cv2.filter2D(img_gaussian, -1, kernely)



cv2_imshow( img)
cv2_imshow(img_canny)
cv2_imshow(img_sobelx)
cv2_imshow( img_sobely)
cv2_imshow( img_sobel)
cv2_imshow(img_prewittx)
cv2_imshow( img_prewitty)
cv2_imshow(img_prewittx + img_prewitty)


cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
blur = cv2.GaussianBlur(img, (7, 7), 0)

laplacian = cv2.Laplacian(blur, cv2.CV_64F)

# Apply a threshold to the Laplacian image to identify the edges
_, thresh = cv2.threshold(laplacian, 5, 255, cv2.THRESH_BINARY)
```

```
# Display the resulting edge-detected image
plt.imshow(thresh, cmap='gray')
plt.axis('off')
plt.title(f"LOG")
```

```
from math import log10, sqrt

def PSNR(original, compressed):
 mse = np.mean((original - compressed) ** 2)
 if(mse == 0): # MSE is zero means no noise is present in the signal .
       # Therefore PSNR have no importance.
   return 100
 max_pixel = 255.0
 psnr = 20 * log10(max_pixel / sqrt(mse))
 return psnr

print(f"PSNR value of Canny {PSNR(img,img_canny)} dB")
print(f"PSNR value of Robert {PSNR(img,edged_img)} dB")
print(f"PSNR value of Sobel {PSNR(img,img_sobel)} dB")
print(f"PSNR value of Prewitt {PSNR(img,img_prewittx + img_prewitty)} dB")
print(f"PSNR value of LOG {PSNR(img,thresh)} dB")
```

```
print(f"SSIM value of Canny {calculate_ssim(img,img_canny)} ")
print(f"SSIM value of Robert {calculate_ssim(img,edged_img)} ")
print(f"SSIM value of Sobel {calculate_ssim(img,img_sobel)} ")
print(f"SSIM value of Prewitt {calculate_ssim(img,img_prewittx +
img_prewitty)} ")
print(f"SSIM value of LOG {calculate_ssim(img,thresh)} ")
```