**Machine Vision System Practical 5**

**Program for Edge Detection of a given object using image processing**

**in OpenCV using Python**

In this practical, we will learn about Gradients and understand three methods for Edge Detection of a given object/image in Open CV.

[Now, we could think of gradients as these edge like regions](javascript:;) [that are present in an image. Now, they're not the same thing, gradients and edges are](javascript:;) [completely different things from a mathematical point of view. But we can pretty much get](javascript:;) [away with thinking of gradients as edges from a programming perspective only.](javascript:;)

If the faces of object are smooth and homogeneous, then the intensity of the reflected light will be constant or at least will vary uniformly. However, at the boundary of the adjacent faces, the reflected light intensity will have a jump discontinuity. Therefore at the edge, the gradient (derivative) of light intensity will be infinite. Hence, the gradient operator can be used to locate the edge.

So basically, edge is present where gradient of light intensity is large. The algorithm or programme will go pixel by pixel to find whether it is an edge, once analysis is done it will show all the points/pixel that are present on edge.

So firstly, we will read an image and convert it into gray scale.

Code:

import cv2 as cv

import numpy as np

img = cv.imread('Photos/park.jpg')

cv.imshow('Park', img)

gray = cv.cvtColor(img, cv.COLOR\_BGR2GRAY)

cv.imshow('Gray', gray)

**1] Laplacian:**

The output of this method looks like a pencil shading off the given image. It shows the edges (laplacian edges) in the image like they are drawn over a chalkboard and the lightly smudged.

So we going to define a variable called ***lap*** and set this **equal** to ***cv.Laplacian(gray, cv.CV\_64F)***. This will take the source image, which is ***gray*** now, and it will take a D depth or data depth which is ***cv.CV\_64F***.

[The Laplacian method computes the gradients of this grayscale image. Generally](javascript:;) [this involves a lot of mathematics. But essentially, when we transition from black to white and](javascript:;) [white to black, that's considered a positive and a negative slope. Now, images itself cannot](javascript:;) [have negative pixel values. So what we do is we essentially compute the absolute value](javascript:;) [of that image. So all the pixel values of the image are converted to the ***absolute*** values.](javascript:;) [And then we convert that to a ***uint8,*** to an image specific datatype. So that's basically](javascript:;) happening in the third line of code.

The next line is to display the image named ***Laplacian***,by passing it the values of ***lap***.

Code:

# Laplacian

lap = cv.Laplacian(gray, cv.CV\_64F)

lap = np.uint8(np.absolute(lap))

cv.imshow('Laplacian', lap)

**2] Sobel**

This method is also known as Sobel Gradient Magnitude Representation. The use of this operator can be found in many advance cases.

Sobel computes the gradient in two directions, the x and y. So for this we have created two variable ***sobelx*** and ***sobely*** in line 2 and 3, and we have set them **equal** to ***cv.Sobel()***. In the brackets we have pass in the image i.e. ***gray***, the data depth i.e. ***cv.CV\_64F***, and we have set coordinate **x = 1** & **y = 0** for **sobelx** and **x = 0** & **y = 1** for **sobely.**

We can get a combined Sobel image, by combining **sobelx** and **sobely** using operator **Bitwise OR.** To do this in line 4, we have created a variable ***combined\_sobel*** and set it **equal** to ***cv.bitwise\_or()*** and we have passed in values of **sobelx** and **sobely** in the brackets.

Then we have used ***cv.imshow*** command to display all three images.

Code:

# Sobel

sobelx = cv.Sobel(gray, cv.CV\_64F, 1, 0)

sobely = cv.Sobel(gray, cv.CV\_64F, 0, 1)

combined\_sobel = cv.bitwise\_or(sobelx, sobely)

cv.imshow('Sobel X', sobelx)

cv.imshow('Sobel Y', sobely)

cv.imshow('Combined Sobel', combined\_sobel)

**3] Canny Algorithm**

Canny Algorithm or Canny Edge Detector is an advance algorithm involving many steps in it to create edge cascades. It is pretty famous in computer vision and involves steps like blurring, grading computations, and other steps like them and even sobel. It basically gives a, more clear version of sobel.

In this we have created a memory/variable name ***canny*** in line 2and set it **equal** to ***cv.Canny()***. In the brackets we have passed in the image i.e.  ***gray***, and we have set a threshold values of  **150** & **175**. Then in next line we have displayed the Canny Image.

Code:

# Canny Algorithm / Canny Edge Detector

canny = cv.Canny(gray, 150, 175)

cv.imshow('Canny', canny)

cv.waitKey(0)

**Code:**

import cv2 as cv

import numpy as np

img = cv.imread('../Resources/Photos/park.jpg')

cv.imshow('Park', img)

gray = cv.cvtColor(img, cv.COLOR\_BGR2GRAY)

cv.imshow('Gray', gray)

# Laplacian

lap = cv.Laplacian(gray, cv.CV\_64F)

lap = np.uint8(np.absolute(lap))

cv.imshow('Laplacian', lap)

# Sobel

sobelx = cv.Sobel(gray, cv.CV\_64F, 1, 0)

sobely = cv.Sobel(gray, cv.CV\_64F, 0, 1)

combined\_sobel = cv.bitwise\_or(sobelx, sobely)

cv.imshow('Sobel X', sobelx)

cv.imshow('Sobel Y', sobely)

cv.imshow('Combined Sobel', combined\_sobel)

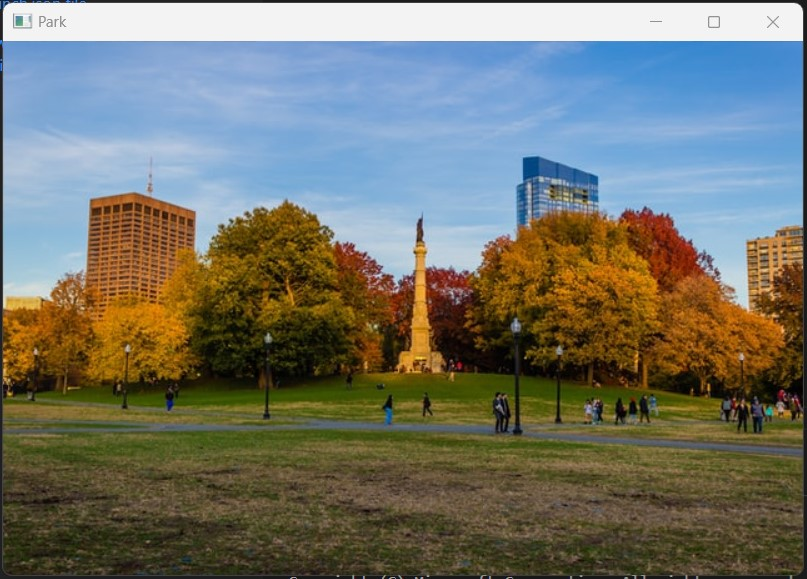
# Canny Algorithm / Canny Edge Detector

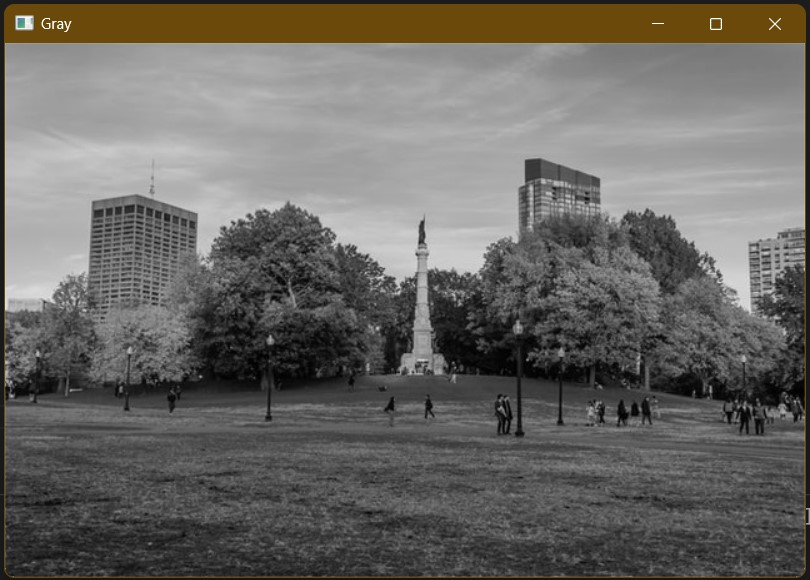
canny = cv.Canny(gray, 150, 175)

cv.imshow('Canny', canny)

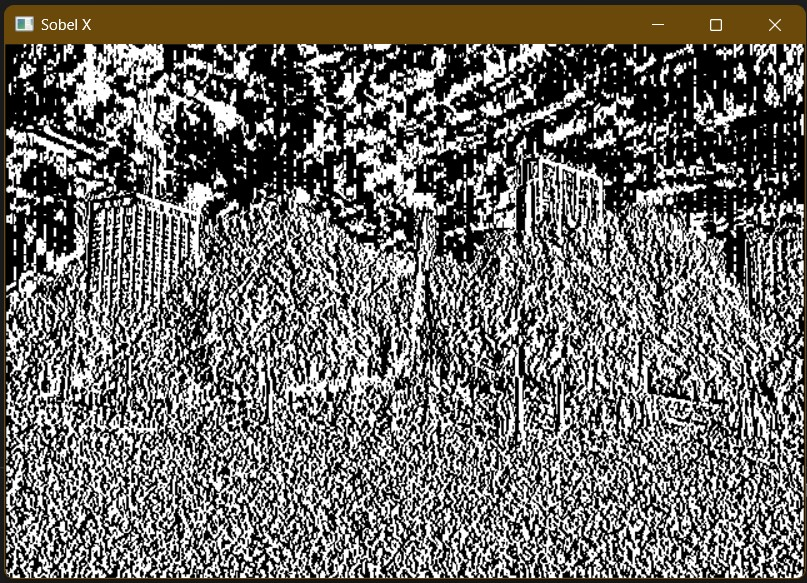
cv.waitKey(0)

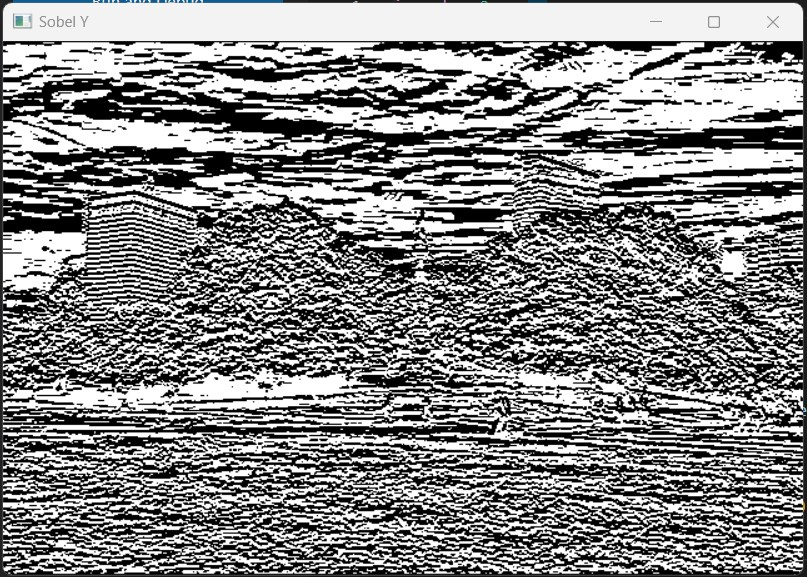
**Outputs:**

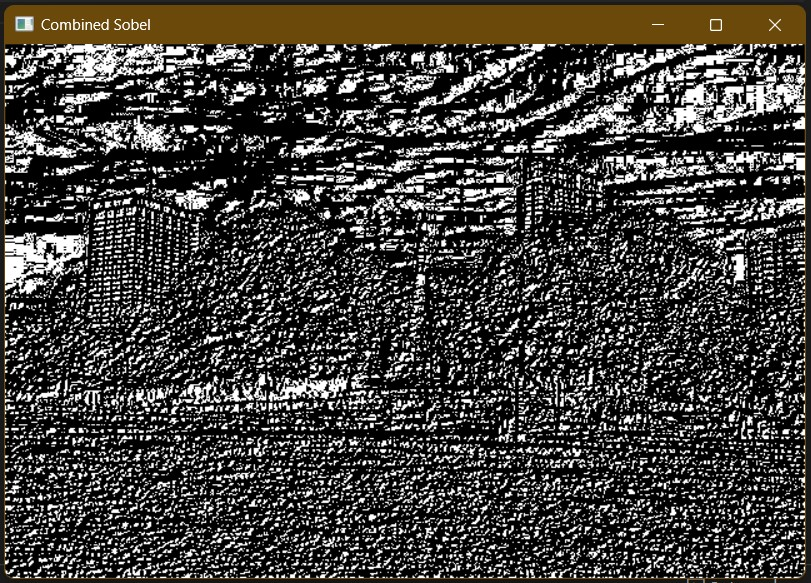
****

****

****

****

****

****

****