## Sobel Filter

A Sobel filter, is a type of filter that is used for edge detection in images. It is defined by a specific matrix or window size, such as 3x3 or 5x5, and is applied to the image to highlight the edges by calculating the gradient intensity.

The python code for it:

### *import the libraries*

```
from scipy import ndimage, datasets
>>> import matplotlib.pyplot as plt
>>> import numpy as np
```

*load the image*

```
>>> ascent = datasets.ascent().astype('int32')
```

*Appling sobel filter by computing the vertical and `horizontal gradient of the image*

```
>>> sobel_h = ndimage.sobel(ascent, 0)  # horizontal gradient
>>> sobel_v = ndimage.sobel(ascent, 1)  # vertical gradient
*calculating the edge magnitude*
>>> magnitude = np.sqrt(sobel_h**2 + sobel_v**2)
>>> magnitude *= 255.0 / np.max(magnitude)  # normalization
```

*plotting the results*

```
>>> fig, axs = plt.subplots(2, 2, figsize=(8, 8))
>>> plt.gray()  # show the filtered result in grayscale
>>> axs[0, 0].imshow(ascent)
>>> axs[0, 1].imshow(sobel_h)
>>> axs[1, 0].imshow(sobel_v)
>>> axs[1, 1].imshow(magnitude)
>>> titles = ["original", "horizontal", "vertical", "magnitude"]
>>> for i, ax in enumerate(axs.ravel()):
...     ax.set_title(titles[i])
...     ax.axis("off")
>>> plt.show()
```

……………………………………………………………………………………………………………………

## Laplacian filter

The Laplacian filter is used for detection of edges in an image. It highlights areas in which intensity changes rapidly producing a picture of all the edges in an image.It is a standard Laplacian of Gaussian convolution. It designed to measure changes in intensity without being overly sensitive to noise. The function produces a peak at the start of the change in intensity and then at the end of the change.

Because the Laplacian of Gaussian produces a fairly wide convolution for a small radius it become quite computationally expensive as radius is increased

- First, we load the original image.
- As Laplacian is prone to be affected by noise, so best strategy is to remove unwanted noise by applying a Gaussian blur and then convert the original image to grayscale
- After performing Gaussian blur, convert the image to grayscale.
- On a Grayscale image, apply a Laplacian operator and display the image output.

```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
# Read the original image
img = cv2.imread('elon.jpg')
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(RGB_img)
src = cv2.GaussianBlur(RGB_img, (3, 3), 0)
# converting to gray scale
gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
edges = cv2.Laplacian(gray, -1, ksize=3,
scale=1,delta=0,borderType=cv2.BORDER_DEFAULT)
plt.imshow(edges)
```

## Canny Edge Detector

It is a popular and widely used edge detection technique that aims to identify and extract the edges of objects within an image and its algorithm is a multistage process that helps to identify the edges in an image by reducing noise and preserving important edge features.

The Canny edge detection algorithm is composed of 5 steps:

1) Noise reduction
2) Gradient calculation
3) Non-maximum suppression
4) Double threshold
5) Edge Tracking by Hysteresis

The python code for it:

import numpy as np

import matplotlib.pyplot as plt

from scipy import ndimage as ndi

```
from skimage.util import random_noise

from skimage import feature

# Generate noisy image of a square

image = np.zeros((128, 128), dtype=float)

image[32:-32, 32:-32] = 1

image = ndi.rotate(image, 15, mode='constant')

image = ndi.gaussian_filter(image, 4)

image = random_noise(image, mode='speckle', mean=0.1)

# Compute the Canny filter for two values of sigma

edges1 = feature.canny(image)

edges2 = feature.canny(image, sigma=3)

# display results

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(8, 3))

ax[0].imshow(image, cmap='gray')

ax[0].set_title('noisy image', fontsize=20)

ax[1].imshow(edges1, cmap='gray')

ax[1].set_title(r'Canny filter, $\sigma=1$', fontsize=20)

ax[2].imshow(edges2, cmap='gray')

ax[2].set_title(r'Canny filter, $\sigma=3$', fontsize=20)

for a in ax:

   a.axis('off')

fig.tight_layout()

plt.show()
```

…………………………………………………………………………………………………………………………………………

## Contours in Image processing

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition, for better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.

In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

The python code to find the counters in a binary image:

```
import numpy as np
import cv2 as cv
im = cv.imread('test.jpg')
assert im is not None, "file could not be read, check with os.path.exists()"
imgray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(imgray, 127, 255, 0)
im2, contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
```

See, there are three arguments in **cv.findContours()** function, first one is source image, second is contour retrieval mode, third is contour approximation method. And it outputs a modified image, the contours and hierarchy. contours is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.