Filters and edge detection

Sobel Filter:

Sobel Filters use matrix multiplication to transform the image into a matrix of positive and negative values representing black and white pixels respectively.

There are 2 types of Sobel kernels used in this process:

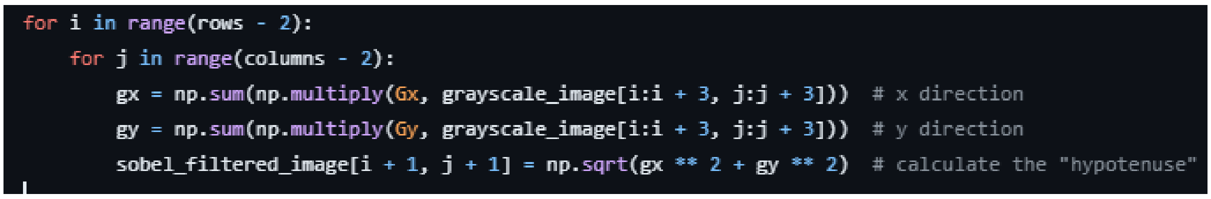
)x-direction Sobel kernel Gx

$$\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$$

)y-direction Sobel kernel Gy

$$\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

Each of these kernels are multiplied by a region of the grayscale image to produce its respective kernels and the outputs are used to obtain the Magnitude matrix



```
for i in range(rows - 2):
    for j in range(columns - 2):
        gx = np.sum(np.multiply(Gx, grayscale_image[i:i + 3, j:j + 3]))  # x direction
        gy = np.sum(np.multiply(Gy, grayscale_image[i:i + 3, j:j + 3]))  # y direction
        sobel_filtered_image[i + 1, j + 1] = np.sqrt(gx ** 2 + gy ** 2)  # calculate the "hypotenuse"
```

from which we calculate the threshold value and apply it onto the Magnitude matrix obtaining the final filtered image.

Laplacian filter:

Images as are as a function F with respect to X and Y and the second derivative of F is computed. The points at the X axis on the second derivative graph are called “zero crossings” and represent the location of the edges on the original image both horizontally and vertically. A Laplacian operator is then computed using the sum of the second derivatives that when applied to the image, produces what is called a Laplacian image. The

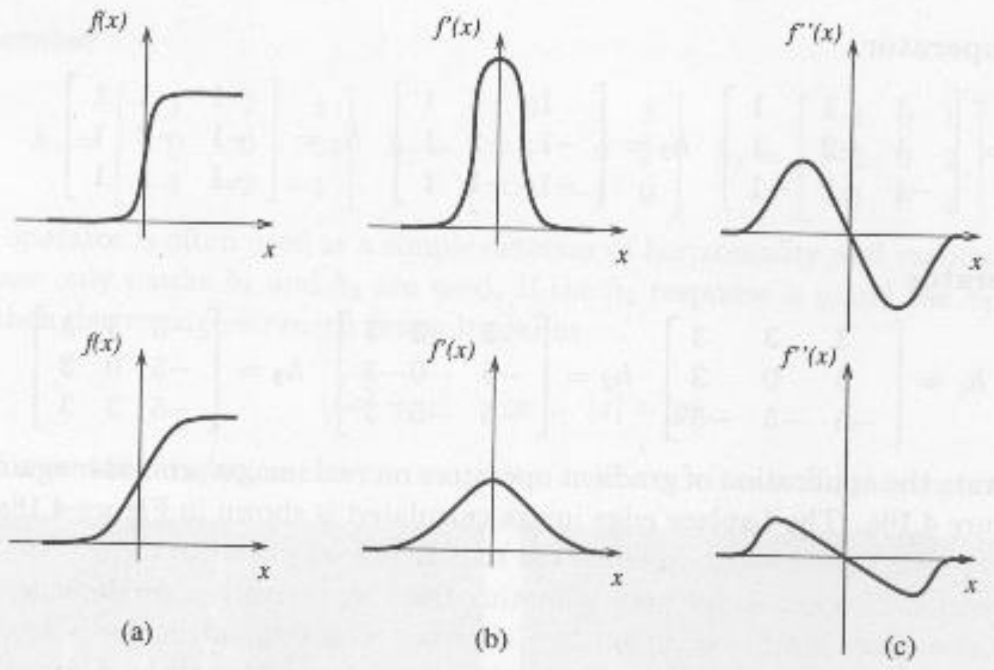
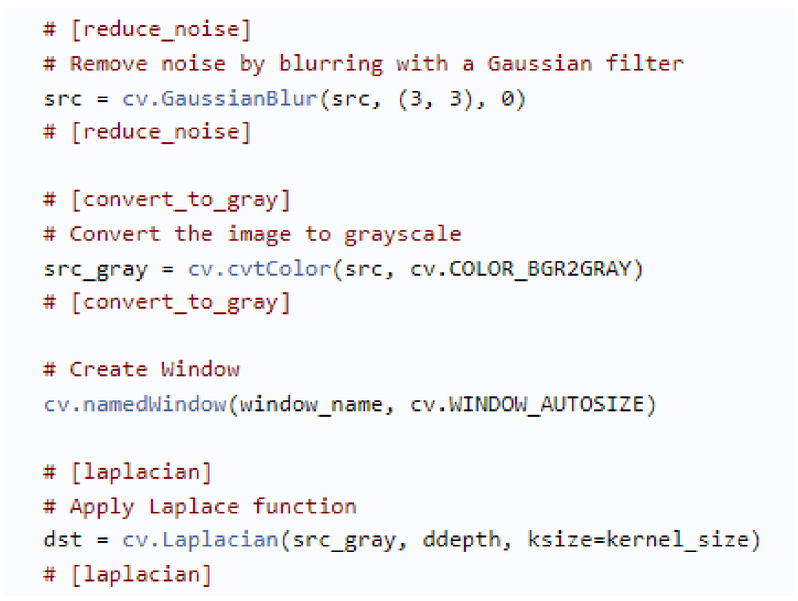


Figure 4.20: *1D edge profile of the zero-crossing.*

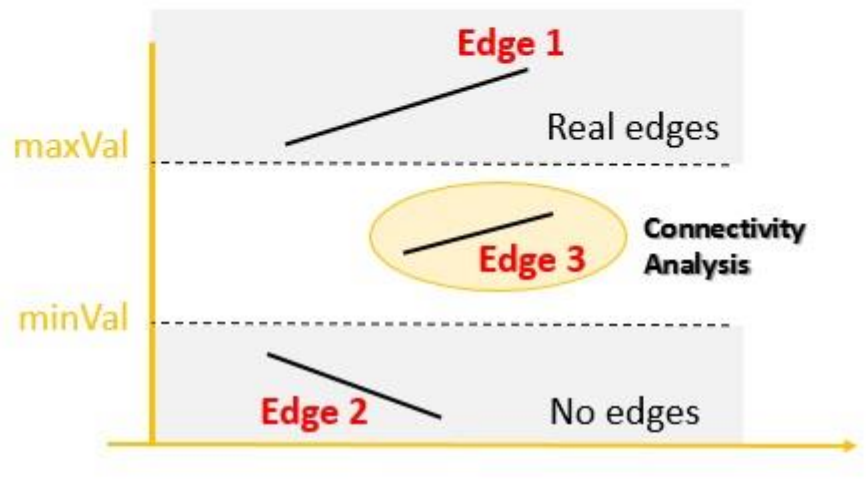
second derivative zero crossing function is then applied to the Laplacian image so that any pixel below the zero crossing is black and any pixel above is white. (*NOTE: gaussian blur is often applied before Laplacian edge detection to reduce noise and filter off less prominent often internal edges)*

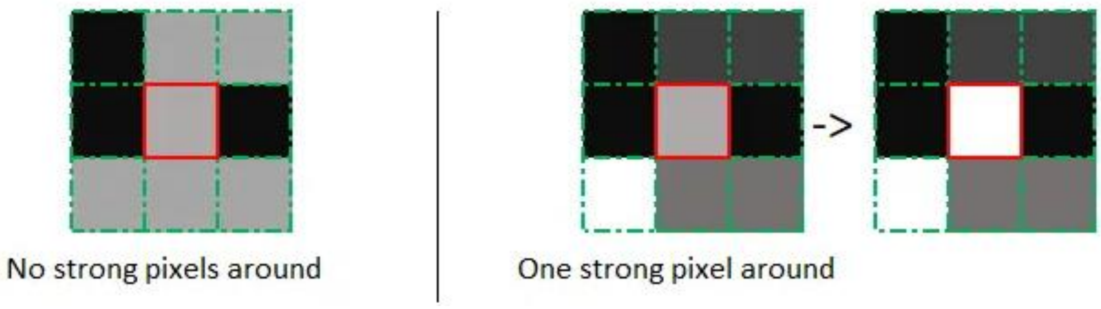


```
# [reduce_noise]
# Remove noise by blurring with a Gaussian filter
src = cv.GaussianBlur(src, (3, 3), 0)
# [reduce_noise]

# [convert_to_gray]
# Convert the image to grayscale
src_gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
# [convert_to_gray]

# Create Window
cv.namedWindow(window_name, cv.WINDOW_AUTOSIZE)

# [laplacian]
# Apply Laplace function
dst = cv.Laplacian(src_gray, ddepth, ksize=kernel_size)
# [laplacian]
```

Canny edge detection:

This technique can be interpreted as sort of an upgrade to the Sobel filter, it works by thinning all the edges until they are 1 pixel wide and applying 2 levels of thresholding. Canny edge detection figures out the direction the edge is moving, calculates the magnitude matrix of each pixel along the edge and compares it to the magnitude of pixels on a parallel line to both the left and right of the edge. Next we apply thresholding by setting 2 values MAXVAL, and MINVAL. Anything above the MAXVAL is automatically considered an edge, anything below the MINVAL is automatically discarded, and anything between the 2 values is checked via the first technique to see if it connects to a point with a value above MAXVAL.





```python
def hysteresis(img, weak, strong=255):
    M, N = img.shape
    for i in range(1, M-1):
        for j in range(1, N-1):
            if (img[i,j] == weak):
                try:
                    if ((img[i+1, j-1] == strong) or (img[i+1, j] == strong) or (img[i+1, j+1] ==
                        or (img[i, j-1] == strong) or (img[i, j+1] == strong)
                        or (img[i-1, j-1] == strong) or (img[i-1, j] == strong) or (img[i-1, j+1]
                        img[i, j] = strong
                    else:
                        img[i, j] = 0
                except IndexError as e:
                    pass
    return img
```

Contours in image processing:

Contours can be defined as a kind of string that wraps around the perimeter of an object, they can be used for corner and shape detection as well as motion detection using active contours or “snakes”. Contours can be found by applying gaussian filter on the image and then using thresholding to make it easier to detect changes in intensity or edges in the image. We can further simplify the edges by interpreting a line of continuous points along the edge as just its end point to save data.

```python
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)
```