In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

In [64]:
```
image_raw = "image.png"
image = cv2.imread(image_raw, cv2.IMREAD_GRAYSCALE)
image.shape
```

Out[64]: (489, 493)

Defining functions that can be used later repeadely

In [65]:
```
def convolve2d(image, kernel):

    image = np.array(image)
    kernel = np.array(kernel)

    kernel_height, kernel_width = kernel.shape
    image_height, image_width = image.shape

    result = np.zeros((image_height - kernel_height + 1, image_width - ke

    for i in range(image_height - kernel_height + 1):
        for j in range(image_width - kernel_width + 1):

            img_slice = image[i:i+kernel_height, j:j+kernel_width]
            value = np.sum(img_slice * kernel)
            result[i, j] = value

    return result


def apply_operator(image, Gx, Gy):
    Gx = np.array(Gx)
    Gy = np.array(Gy)

    Gx_result = convolve2d(image, Gx)
    Gy_result = convolve2d(image, Gy)

    magnitude = np.sqrt(Gx_result**2 + Gy_result**2)
    return magnitude



def plot_stuff(img,img_den, operator_name):
    plt.subplot(1, 2, 1)
    plt.imshow(img, cmap='gray')
    plt.title(f'{operator_name} without Denoising')
    plt.subplot(1, 2, 2)
    plt.imshow(img_den, cmap='gray')
    plt.title(f'{operator_name} with Denoising')
```

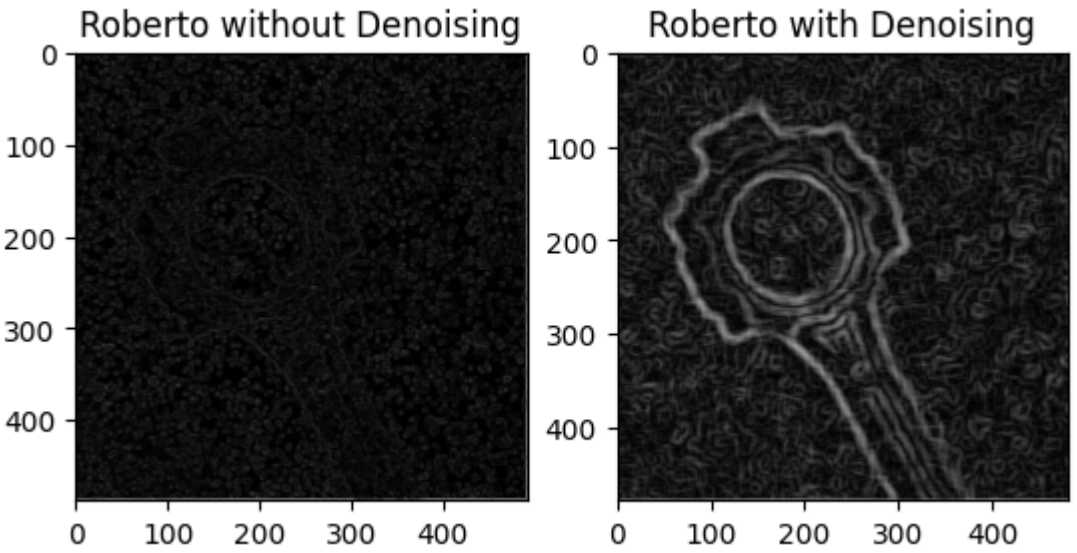
Removing Noise

Using Gaussian Filter

In [ ]:
```python
# implement gaussian filter

def gaussian_filter(size, sigma):
    kernel = np.zeros((size, size))
    center = size // 2

    for i in range(size):
        for j in range(size):
            x = i - center
            y = j - center
            kernel[i, j] = np.exp(-(x**2 + y**2) / (2 * sigma**2))

    kernel /= np.sum(kernel)
    return kernel
gaussian_kernel = gaussian_filter(11, 9)

denoised_image = convolve2d(image, gaussian_kernel)
```

Roberto

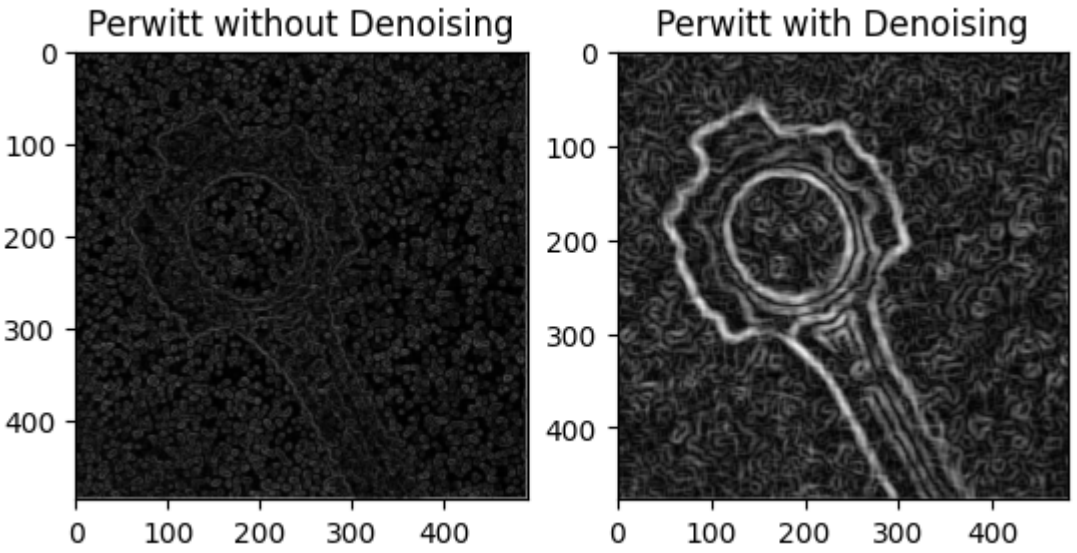
In [67]:
```python
roberto_edges = apply_operator(
    image,
    [[1, 0], [0, -1]], [[0, 1], [-1, 0]]
)
roberto_edges_denoised = apply_operator(
    denoised_image,
    [[1, 0], [0, -1]], [[0, 1], [-1, 0]]
)

plot_stuff(roberto_edges, roberto_edges_denoised, "Roberto")
```



Perwitt

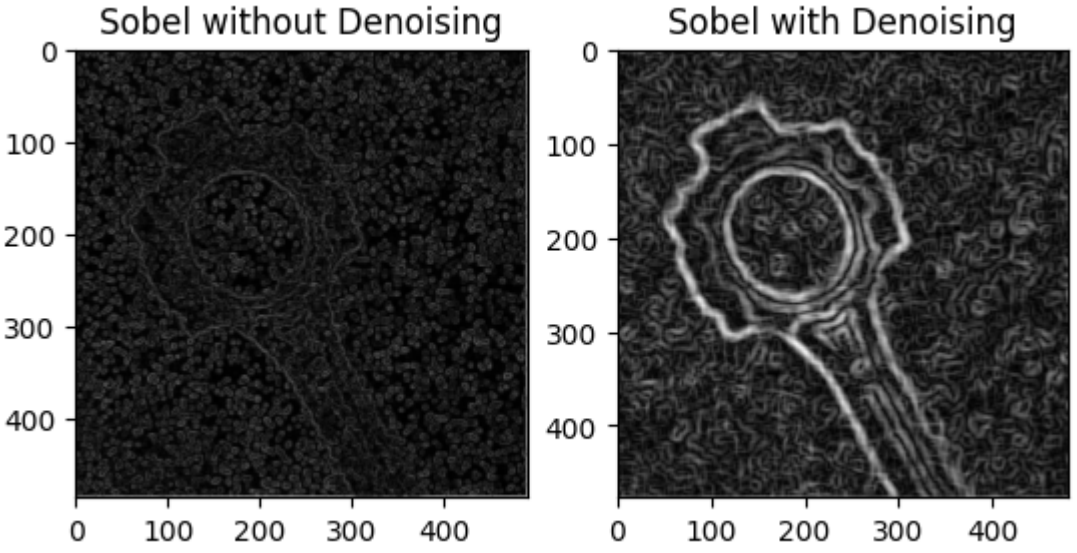
In [68]:
```python
perwitt_edges = apply_operator(
    image,
    [[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]], [[1, 1, 1], [0, 0, 0], [-1, -1,
)
perwitt_edges_denoised = apply_operator(
    denoised_image,
    [[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]], [[1, 1, 1], [0, 0, 0], [-1, -1,
```

```
)

plot_stuff(perwitt_edges, perwitt_edges_denoised, "Perwitt")
```



Sobel

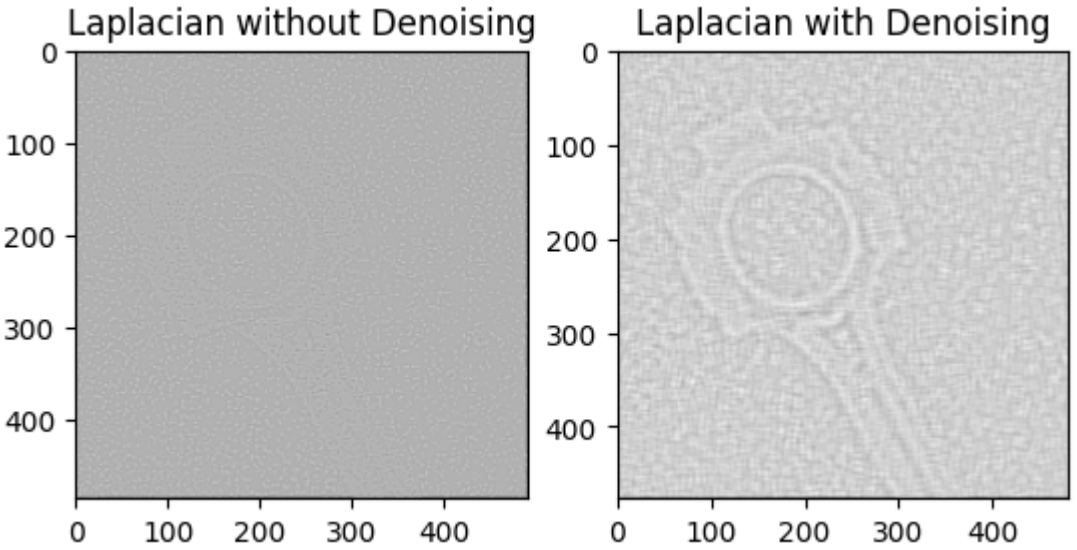
In [69]:
```
sobel_edges = apply_operator(
    image,
    [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], [[1, 2, 1], [0, 0, 0], [-1, -2,
)
sobel_edges_denoised = apply_operator(
    denoised_image,
    [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], [[1, 2, 1], [0, 0, 0], [-1, -2,
)

plot_stuff(sobel_edges, sobel_edges_denoised, "Sobel")
```



Laplacian

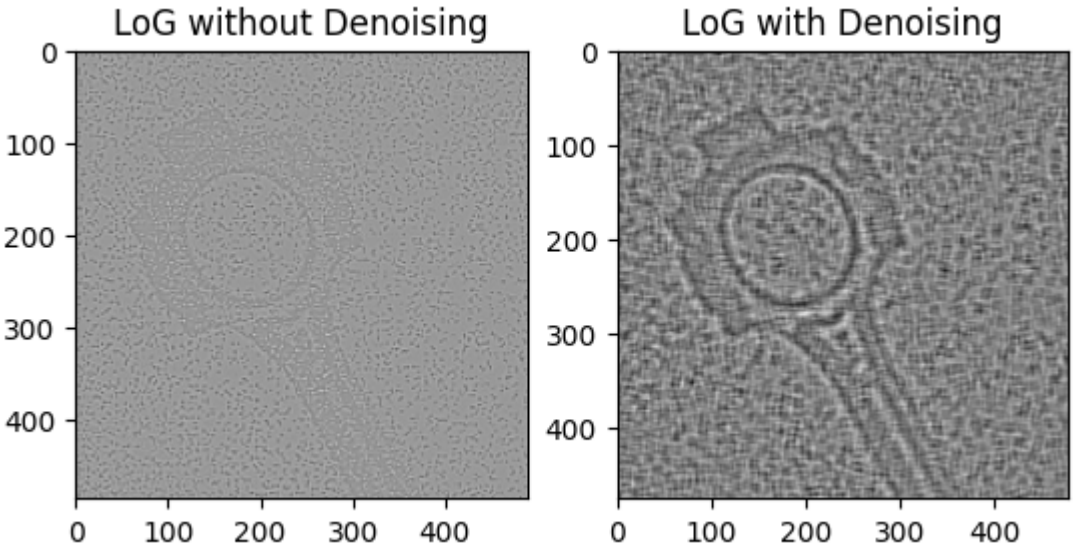
In [70]:
```
# implement laplacian operator
laplacian_edges = convolve2d(
    image,
```

```
    [[0, 1, 0], [1, -4, 1], [0, 1, 0]]
)
laplacian_edges_denoised = convolve2d(
    denoised_image,
    [[0, 1, 0], [1, -4, 1], [0, 1, 0]]
)

plot_stuff(laplacian_edges, laplacian_edges_denoised, "Laplacian")
```



Laplacian + Gaussian

In [71]:
```
# implement laplacian with gaussian (LoG) operator
log_edges = convolve2d(
    image,
    [[0, 0, -1, 0, 0], [0, -1, -2, -1, 0], [-1, -2, 16, -2, -1], [0, -1,
)
log_edges_denoised = convolve2d(
    denoised_image,
    [[0, 0, -1, 0, 0], [0, -1, -2, -1, 0], [-1, -2, 16, -2, -1], [0, -1,
)

plot_stuff(log_edges, log_edges_denoised, "LoG")
```



Canny Edge Detection:

In [ ]:
```
def sobel_gradients(image):

    Kx = np.array([[-1,0,1],
                   [-2,0,2],
                   [-1,0,1]])

    Ky = np.array([[1,2,1],
                   [0,0,0],
                   [-1,-2,-1]])

    Gx = convolve2d(image, Kx)
    Gy = convolve2d(image, Ky)

    magnitude = np.sqrt(Gx**2 + Gy**2)
    direction = np.arctan2(Gy, Gx)

    return magnitude, direction


def non_max_suppression(mag, angle):

    H, W = mag.shape
    output = np.zeros((H,W))

    # Convert the tan angle to degrees and adjust to [0, 180]
    angle = angle * 180.0 / np.pi
    angle[angle < 0] += 180

    for i in range(1, H-1):
        for j in range(1, W-1):

            q = 0
            r = 0

            # angle 0 - Horizontal
            if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180):
                q = mag[i, j+1]
                r = mag[i, j-1]

            # angle 45 - Diagonal right
            elif (22.5 <= angle[i,j] < 67.5):
                q = mag[i+1, j-1]
                r = mag[i-1, j+1]

            # angle 90 - Vertical
            elif (67.5 <= angle[i,j] < 112.5):
                q = mag[i+1, j]
                r = mag[i-1, j]

            # angle 135 - Diagonal left
            elif (112.5 <= angle[i,j] < 157.5):
                q = mag[i-1, j-1]
                r = mag[i+1, j+1]

            if mag[i,j] >= q and mag[i,j] >= r:
                output[i,j] = mag[i,j]
```

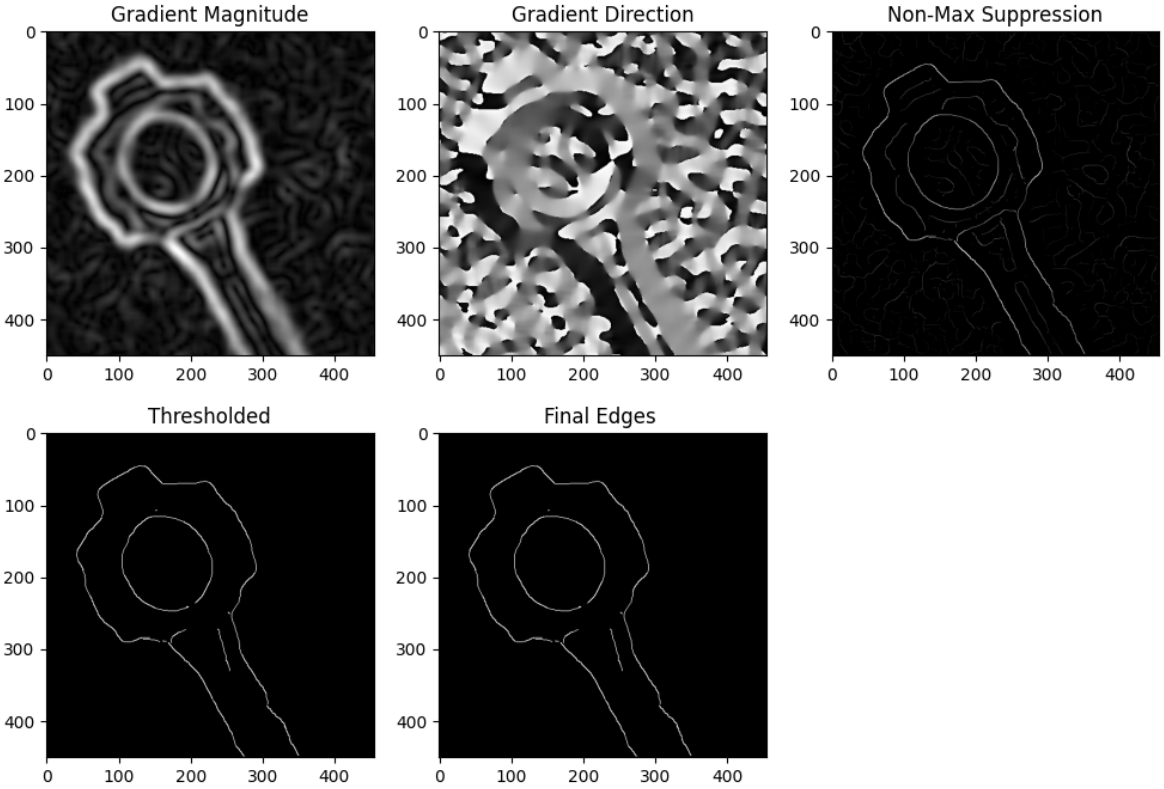
```
            else:
                output[i,j] = 0

    return output

def threshold(img, low_ratio=0.1, high_ratio=0.5):

    high = img.max() * high_ratio
    low = high * low_ratio

    H, W = img.shape
    res = np.zeros((H,W))

    strong = 255
    weak = 240

    strong_i, strong_j = np.where(img >= high)
    weak_i, weak_j = np.where((img <= high) & (img >= low))

    res[strong_i, strong_j] = strong
    res[weak_i, weak_j] = weak


    return res, weak, strong

def hysteresis(img, weak, strong=255):

    H, W = img.shape

    for i in range(1, H-1):
        for j in range(1, W-1):

            if img[i,j] == weak:

                if ((img[i+1, j-1] == strong) or (img[i+1, j] == strong)
                    or (img[i, j-1] == strong) or (img[i, j+1] == strong)
                    or (img[i-1, j-1] == strong) or (img[i-1, j] == stron
                    img[i,j] = strong
                else:
                    img[i,j] = 0
    return img

def canny_edge(image):

    smoothed = convolve2d(image, gaussian_filter(19,9))

    mag, direction = sobel_gradients(smoothed)

    thin = non_max_suppression(mag, direction)

    thresh, weak, strong = threshold(thin)

    final = hysteresis(thresh, weak, strong)

    return {
        'magnitude': mag,
        'direction': direction,
        'thin': thin,
        'thresholded': thresh,
        'final': final
```

```
    }
```

In [73]:
```
def median_filter(image, size=3):

    pad = size//2
    padded = np.pad(image, pad)
    output = np.zeros_like(image)

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            region = padded[i:i+size, j:j+size]
            output[i,j] = np.median(region)

    return output

smoothed = convolve2d(image, gaussian_filter(19,9))
denoised = median_filter(image, 13)


canny_results = canny_edge(smoothed)
```

In [74]:
```
plt.figure(figsize=(12, 8))
plt.subplot(2, 3, 1)
plt.imshow(canny_results['magnitude'], cmap='gray')
plt.title('Gradient Magnitude')
plt.subplot(2, 3, 2)
plt.imshow(canny_results['direction'], cmap='gray')
plt.title('Gradient Direction')
plt.subplot(2, 3, 3)
plt.imshow(canny_results['thin'], cmap='gray')
plt.title('Non-Max Suppression')
plt.subplot(2, 3, 4)
plt.imshow(canny_results['thresholded'], cmap='gray')
plt.title('Thresholded')
plt.subplot(2, 3, 5)
plt.imshow(canny_results['final'], cmap='gray')
plt.title('Final Edges ')
```

Out[74]:
```
Text(0.5, 1.0, 'Final Edges ')
```


Gradient Magnitude
Gradient Direction
Non-Max Suppression
Thresholded
Final Edges
0
100
200
300
400