PS3_2: EDGE DETECTION

CODE:

## Input Images

In [1]:
```
#C:/Users/parth/OneDrive/Desktop//cheerios.png
#C:/Users/parth/OneDrive/Desktop//gear.png
#C:/Users/parth/OneDrive/Desktop//circuit.png
#C:/Users/parth/OneDrive/Desktop//professor.png
```

## Importing Required Libraries

In [1]:
```
import numpy as np
import cv2 as cv
```

## Reading the image, converting to Binary and resizing to fit the screen

In [2]:
```
img1= input("Enter an Image for Smoothening and Sharpening process: ")
img = cv.imread(img1)

scale_percent = 50

#calculate the 50 percent of original dimensions
width = int(img.shape[1] * scale_percent / 100)
height = int(img.shape[0] * scale_percent / 100)

# dsize
dsize = (width, height)
img = cv.resize(img, dsize)
gray = cv.cvtColor(img, cv.COLOR_BGRA2GRAY)
(thresh, blackAndWhiteImage) = cv.threshold(gray, 93, 255, cv.THRESH_BINARY)
print(np.shape(blackAndWhiteImage))
```

```
Enter an Image for Smoothening and Sharpening process: C:/Users/parth/OneDrive/Desktop//professor.png
(676, 933)
```

## Defining the Sobel Kernels of Vertical and Horizontal Direction

In [3]:
```
kernelX = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
kernelY = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
print(np.shape(kernelX))
print(np.shape(kernelY))
```

```
(3, 3)
(3, 3)
```

## Defining the Convolution function for adding Sobel Filter

In [4]:
```
def convolution(kernel, img):
    a,b = np.shape(img)

    l = []

    for i in range (a-2):
        for j in range (b-2):
            temp = img[i:i+3, j:j+3]
            c = np.multiply(kernel, temp)
            m = np.sum(c)
            l += [m]

            #l = np.array(l)
    l = np.array(l)
    l = np.reshape(l, (a-2, b-2))
    #print (len(l))
```

```
    return l
```

## Initializing the X and Y Sobel Filters

In [5]:
```
X = convolution(kernelX, blackAndWhiteImage)
Y = convolution(kernelY, blackAndWhiteImage)
```

## Combining the X anf Y filters, normalizing and then changing the edges to black

In [6]:
```
Grad = np.sqrt(np.square(X) + np.square(Y))
Grad *= 255.0 / Grad.max()
Grad1 = 255 - Grad
(thresh, Grad_Img) = cv.threshold(Grad1, 115, 255, cv.THRESH_BINARY)
```

## Displaying the original and Sobel Filtered Image

In [7]:
```
cv.imshow("Original Image", img)
#cv.imshow("Sobel Filtered Image", output)
cv.imshow("Sobel Filtered Image",Grad_Img)
cv.waitKey(0)
cv.destroyAllWindows()
```

## Canny Edge Detection

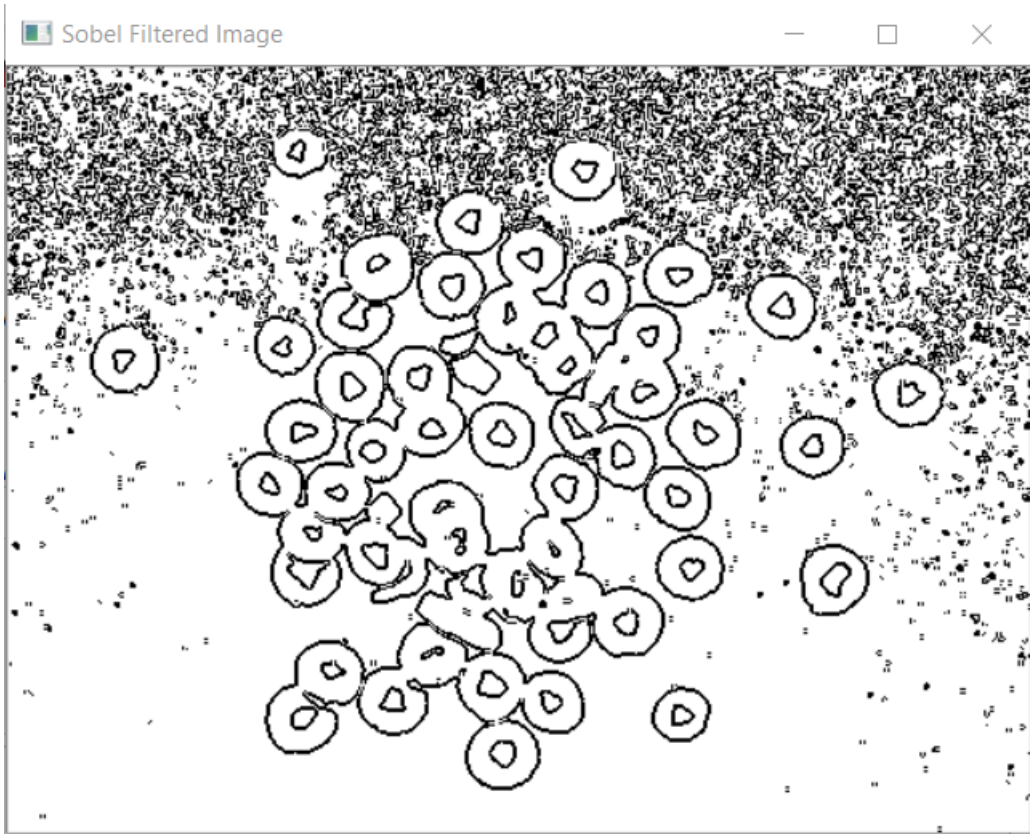
In [5]:
```
def Canny(Lower = 0):
    Lower = cv.getTrackbarPos('Lower', 'Canny Edge Detection')
    Higher = cv.getTrackbarPos('Higher', 'Canny Edge Detection')
    apertureSize = int(cv.getTrackbarPos('apertureSize','Canny Edge Detection' ))

    '''Adding a if loop for keeping Aperture Size between 3, 5 and 7'''

    if ((apertureSize > 3 and apertureSize < 5) or apertureSize == 3):
        apertureSize = 3
    elif ((apertureSize > 5 and apertureSize < 7) or apertureSize == 5):
        apertureSize = 5
    else:
        apertureSize = 7

    '''Adding if loop for L2gradient to keep only True and False Conditions'''

    L2gradient = cv.getTrackbarPos('L2gradient', 'Canny Edge Detection')
    if L2gradient == 0:
        L2gradient = True
    else:
        L2gradient = False
    Canny = cv.Canny(img, Lower, Higher, apertureSize = apertureSize, L2gradient = L2gradient)
    Canny = 255 - Canny #Changing the edges to black and background to white
    cv.imshow('Canny Edge Detection', Canny)


img = cv.GaussianBlur(img, (5,5), 0)
cv.namedWindow('Canny Edge Detection')
```

```
'''Generating Trackbars'''

cv.createTrackbar('Lower', 'Canny Edge Detection', 0, 255, Canny)
cv.createTrackbar('Higher', 'Canny Edge Detection', 0, 255, Canny)
cv.createTrackbar('apertureSize', 'Canny Edge Detection', 3, 7, Canny)
cv.createTrackbar('L2gradient', 'Canny Edge Detection', 0, 1, Canny)
Canny(0)

cv.waitKey(0)
cv.destroyAllWindows()
```

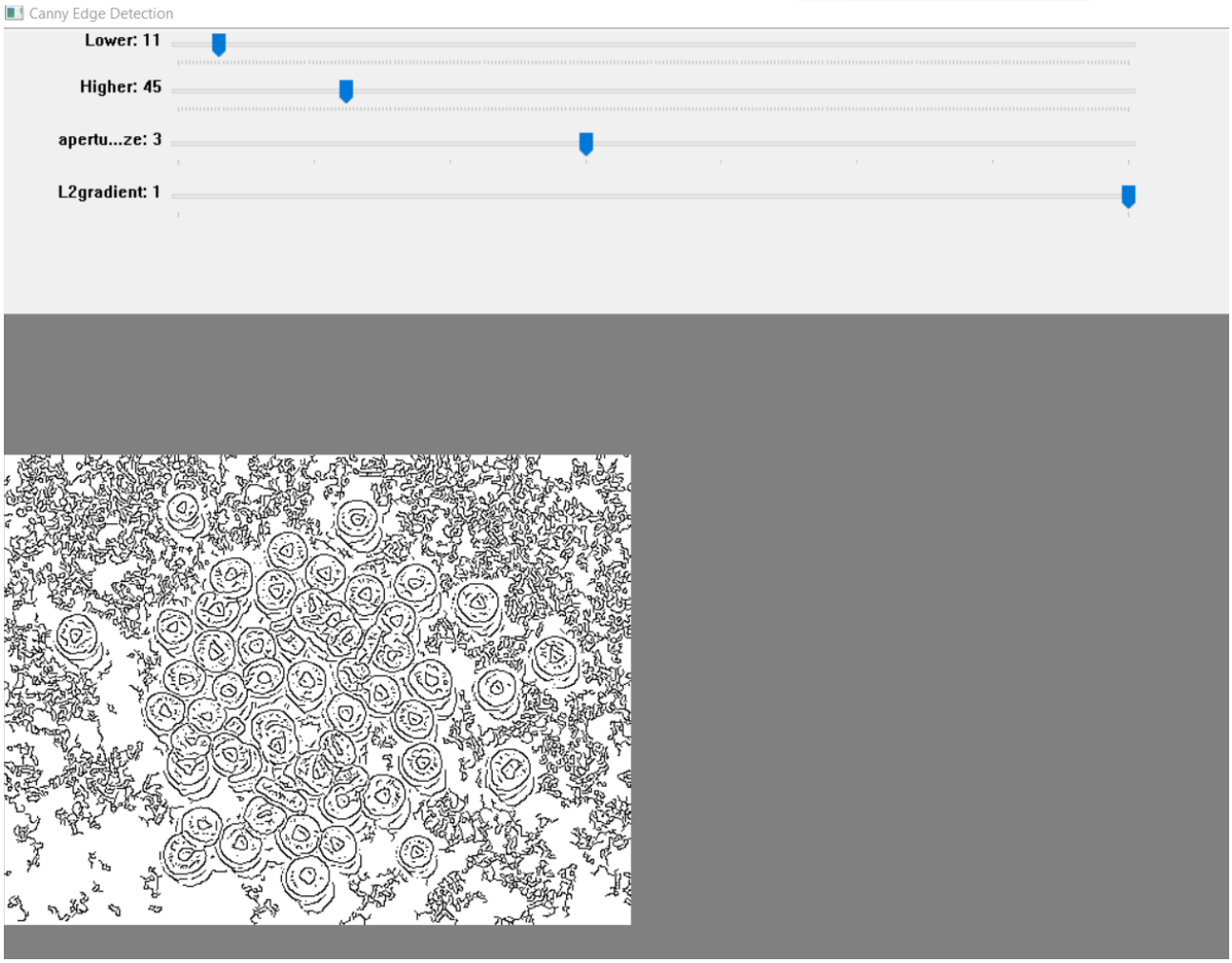
OUTPUT:

Cheerios.png:

Sobel Output:



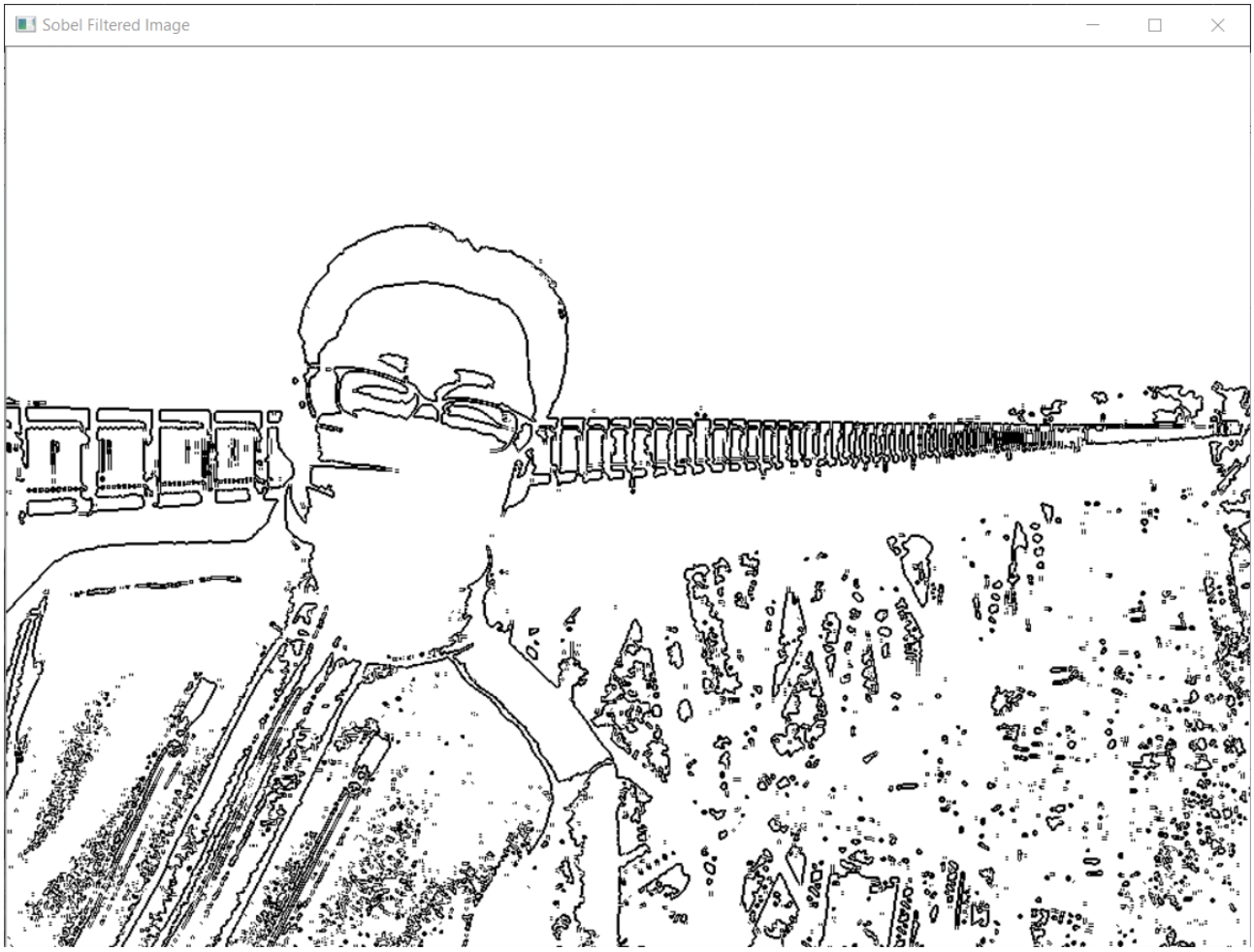
Canny Output:




As seen in both the output images, the details on the cheerios are distinctly visible by canny edge detection for all the threshold values.

Professor.png:

Sobel Output:

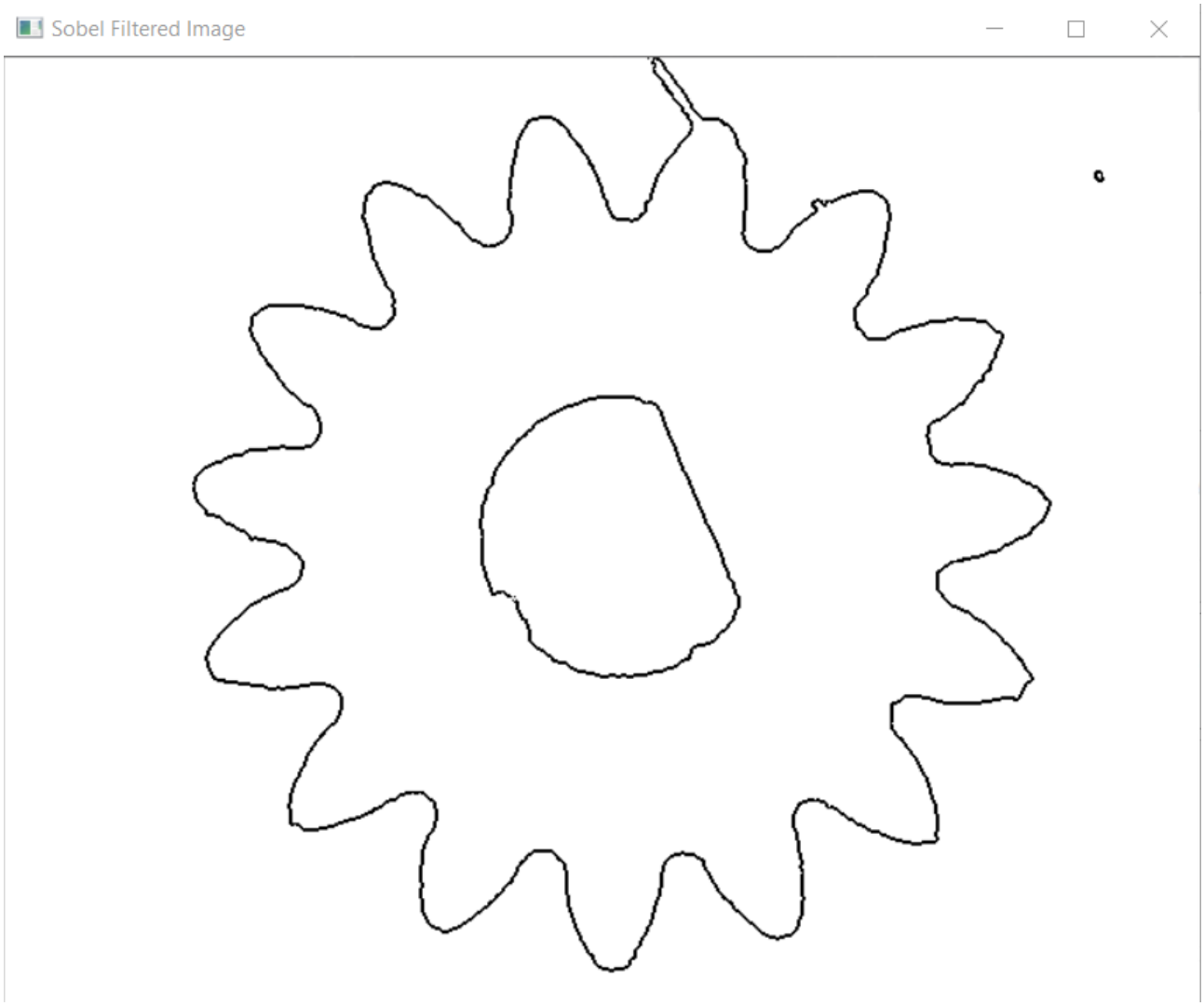


Canny Output:

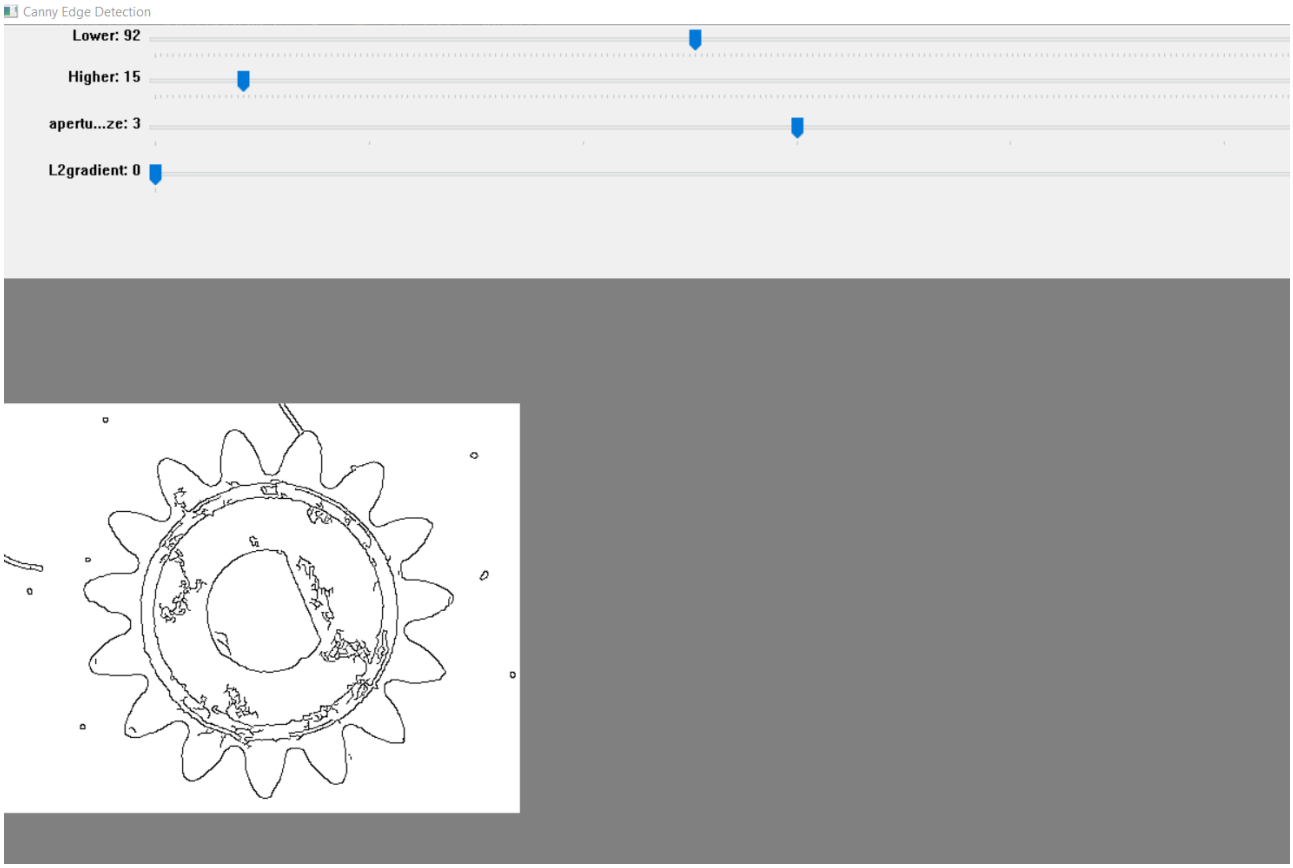

The bridge edges are visible for specific adjustments made on the trackbar which gives a detailed edge detection.

Gear.png:

Sobel Output:



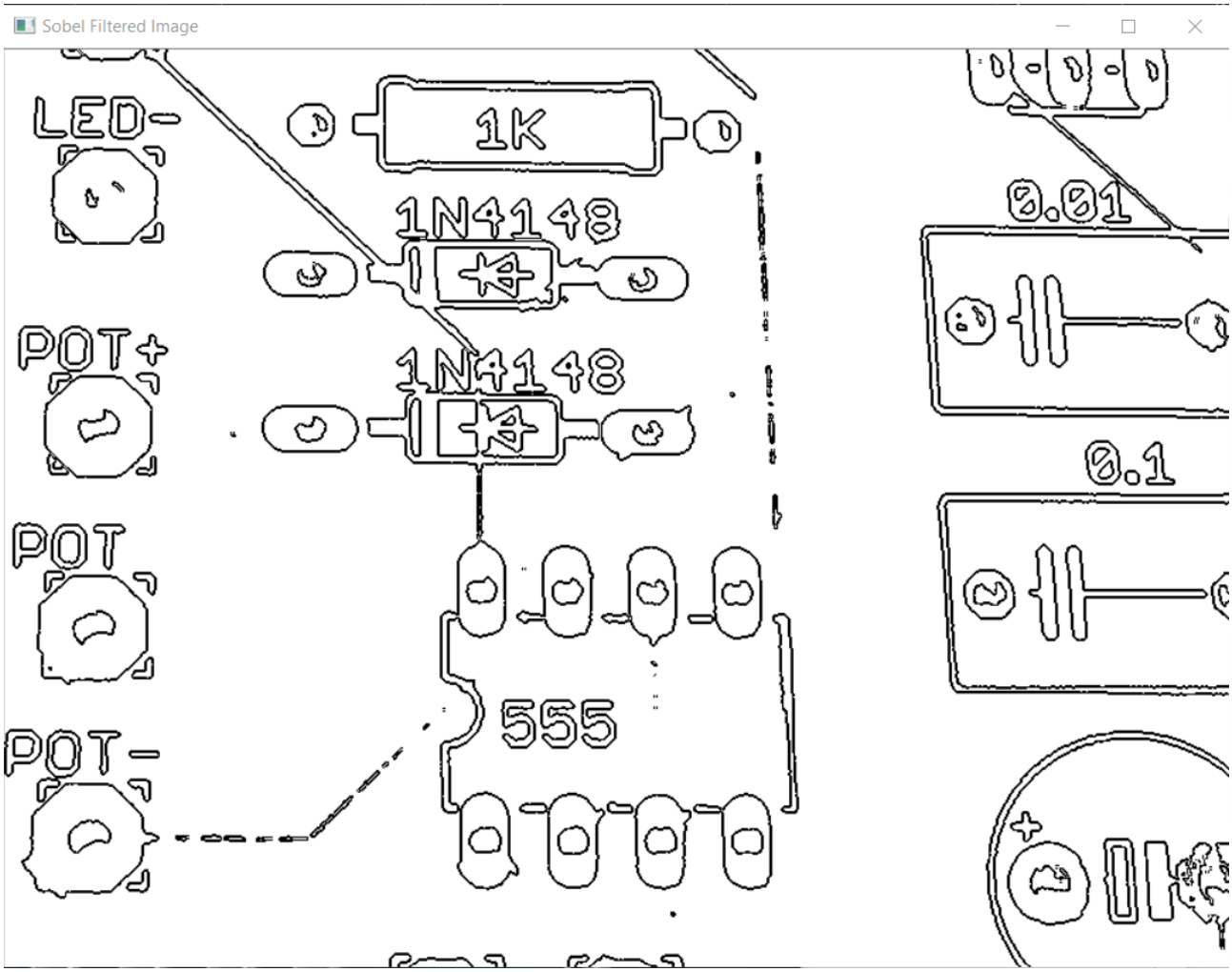
Canny Output:



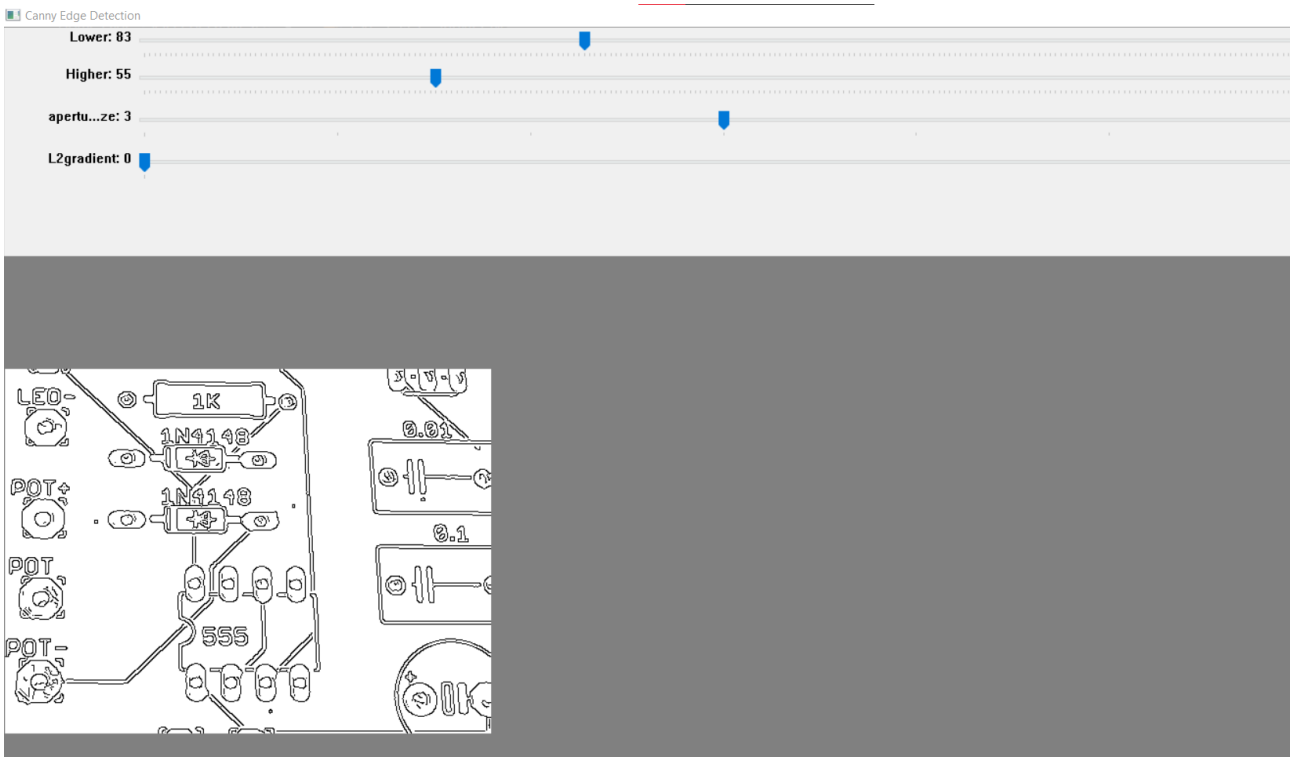

This minute bright spots in the vicinity of the gear are also caught in canny edge detection while they are not in Sobel Filter.

Circuit.png:

Sobel Output:



Canny Output:




Here, sobel filter does a good job of marking all the edges distinctly as is done by canny edge detection because of the convenience of track bar added to it.

---

Operating System: Windows 10

IDE: Jupyter Notebook

Number of Hours Spent: 8.5