CVE Assignment 3 Report

Question 1 (PS3-1)

Algorithm

The objective of this problem is to gain familiarity with the different blurring and filtering functions available in OpenCV. Test images are given with the aim to improve their quality and remove any noise from the images.

For this purpose, a menu-based selection was created, taking in input from the user depending on the operation which may be needed depending on the given image. After applying each filter, the image is finally saved to a file, or the entire process can be restarted depending on the command passed. At the end, a list of all blurs and filters used are displayed, for reproducibility.

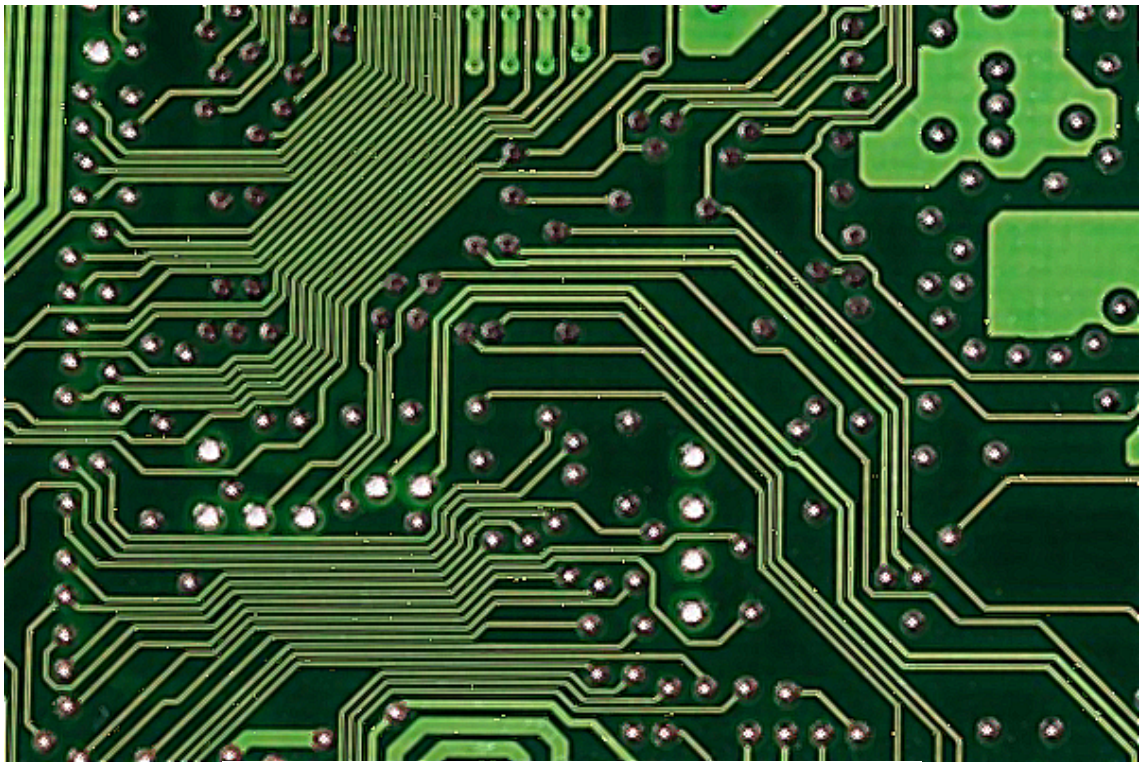
The menu features the following options

1. Box blur: Method is used to blur an image using the normalized box filter
2. Gaussian blur: Any sharp edges in images are smoothed while minimizing too much blurring
3. Median blur: Takes median of all pixels under kernel area. Effective in reducing salt-and-pepper noise
4. Bilateral blur: A bilateral filter is used for smoothening images and reducing noise, while preserving edges
5. Sharpen: convolves an image with the kernel meant for increasing image sharpness
6. Save: Saves the current image to a designated file
7. Quit: Stop the process with no saves

Any other options display an error and a prompt to enter another option.

Results

golf-improved.png
['median blur (kernel size = 5)', 'save']



pcb-improved.png
['median blur (kernel size = 3)', 'sharpen' ,'save']

pots-improved.png
['bilateral (kernel size = 5)', 'sharpen' ,'save']

rainbow-improved.png
['bilateral (kernel size = 9)', 'sharpen' ,'save']

Source Code

```
import numpy as np
import cv2 as cv

f = input("Filename : ")
img = cv.imread(f)
f = f.split('.')
ch = "Choice"
func_list = []

while ch.lower() != "save":
    ch = input("Enter filter : ")
    cv.destroyAllWindows()
    cv.imshow("Original Image", img)
    cv.waitKey(100)
    if ch.lower() == "blur":
        k = int(input("Kernel Size : "))
        img = cv.blur(img, (k,k))
        func_list.append(ch+" ( kernel size = "+str(k)+" )")
        # method is used to blur an image using the normalized box
filter
    elif ch.lower() == "gaussian blur":
        k = int(input("Kernel Size : "))
        img = cv.GaussianBlur(img, (k,k), cv.BORDER_DEFAULT)
        func_list.append(ch+" ( kernel size = "+str(k)+" )")
        #any sharp edges in images are smoothed while minimizing too
much blurring.
    elif ch.lower() == "median blur":
        k = int(input("Kernel Size : "))
        img = cv.medianBlur(img, k)
        func_list.append(ch+" ( kernel size = "+str(k)+" )")
        #Takes median of all pixels under kernel area. Effective in
reducing salt-and-pepper noise.
    elif ch.lower() == "bilateral":
        d1 = int(input("Kernel Dia : "))
        img = cv.bilateralFilter(img, d1, 75, 75)
        func_list.append(ch+" ( kernel size = "+str(d1)+" )")
        #A bilateral filter is used for smoothening images and reducing
noise, while preserving edges.
    elif ch.lower() == "sharpen":
        K = np.array([[-1,-1,-1],[-1,9,-1],[-1,-1,-1]])
        img = cv.filter2D(img, ddepth=-1, kernel=K)
        func_list.append(ch)
        #convolves an image with the kernel
```

```
    elif ch.lower() == "save":
        cv.imwrite(f[0]+"-improved."+f[1], img)
        func_list.append(ch)
    elif ch.lower() == "quit":
        break
    else:
        print("No such options!")
        continue
    cv.imshow(ch+" applied", img)
    cv.waitKey(100)

print("Filters Applied : ")
print(func_list)
cv.destroyAllWindows()
```

Question 2 (PS3-2)

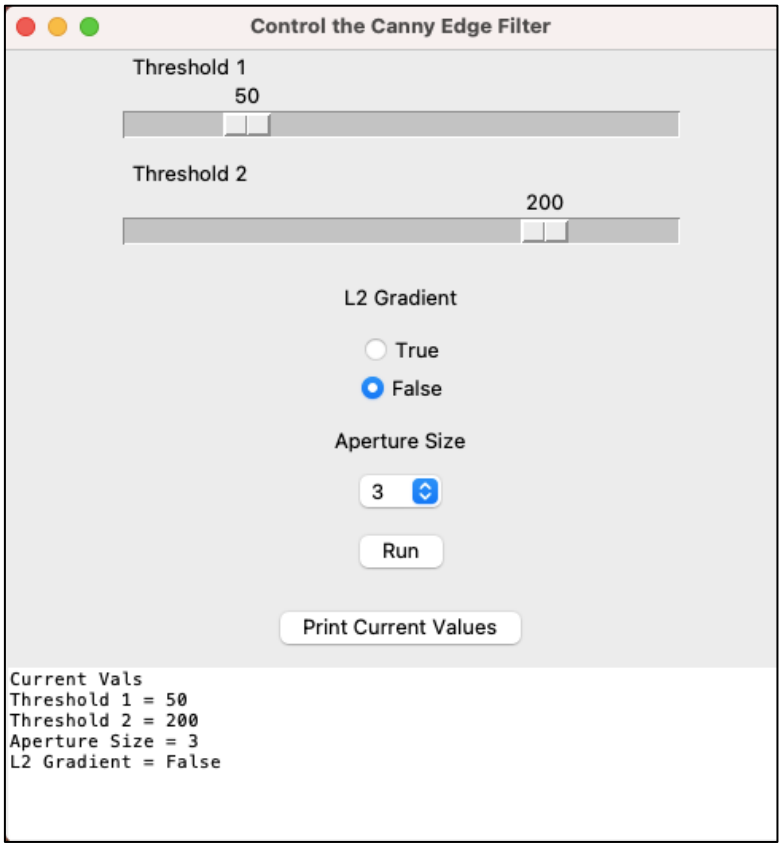
Algorithm

The objective of this problem was to detect the edges in the given images using two methods primarirly: the Sobel filter and the Canny edge filter. The Sobel filter was designed manually, while the Canny Edge filter makes use of a GUI to vary the parameters in the Canny edge function and display an image accordingly.

Applying the Sobel Filter:

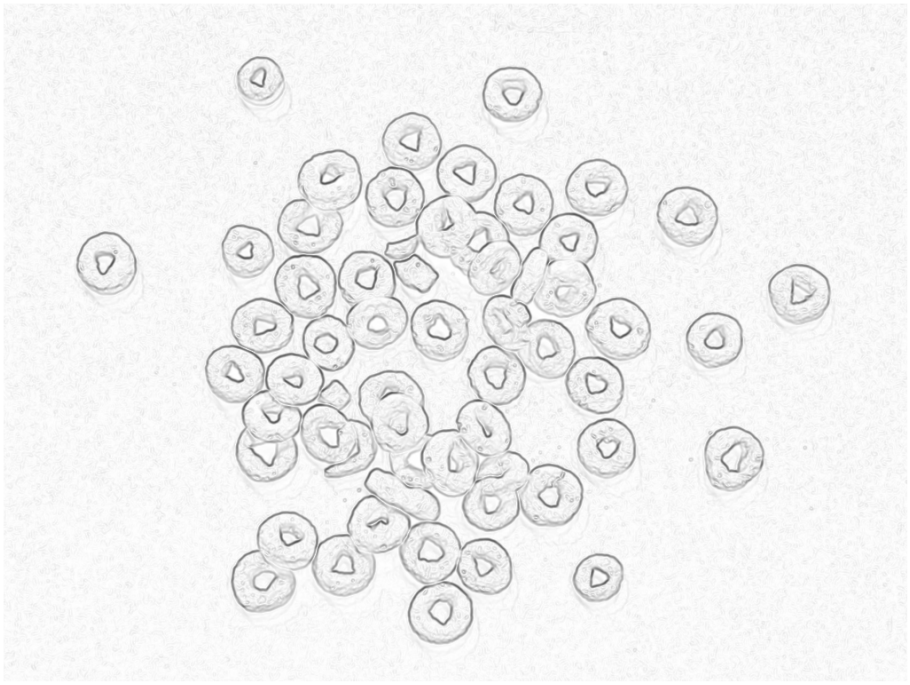
1. Pad the image
2. Define the Gx and Gy matrices for the convolution process
3. Run a loop, slicing the image into the required sizes, multiplying the slices with Gx and Gy, and normalize the values
4. Determine the intensity of the pixel at that point using values calculated
5. Display the image and save it

The GUI developed for the Canny edge control is displayed below. It utilizes scale sliders to modify the thresholds, a drop-down menu to select the Aperture Size for the filter, and a Radiobutton to switch the L2 Gradient value between True and False.

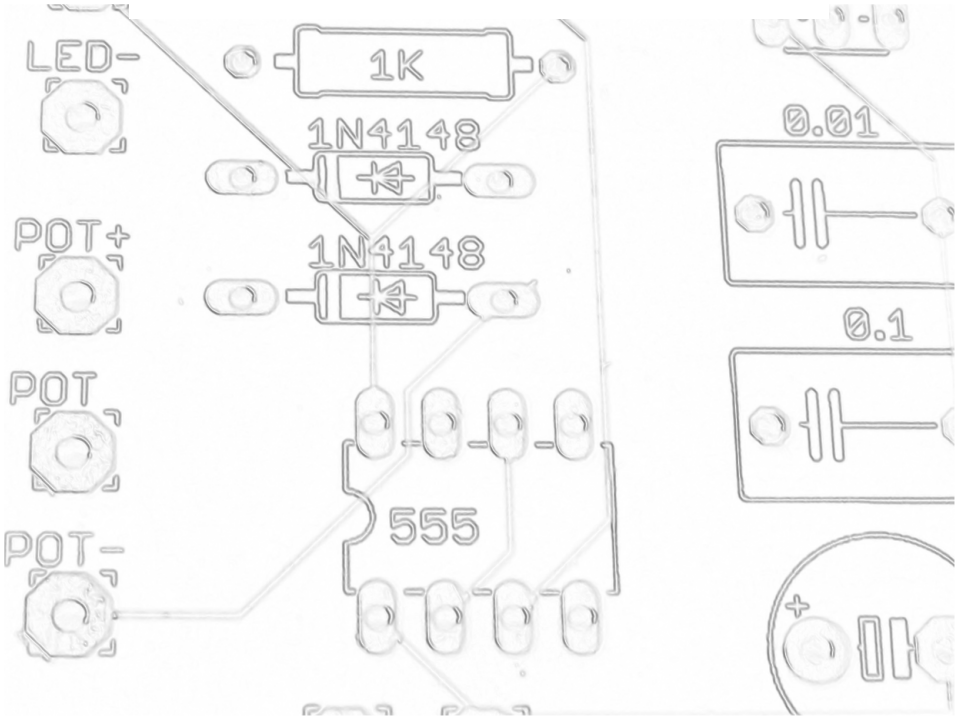


Control GUI

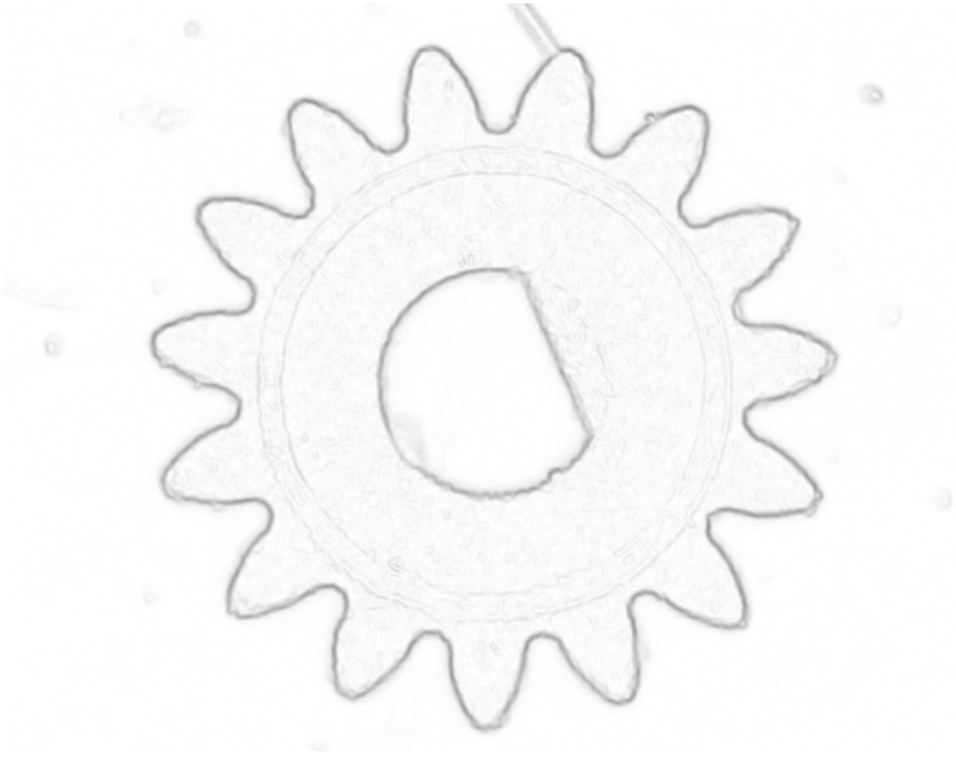
Sobel Filter Results



cheerios-sobel.png



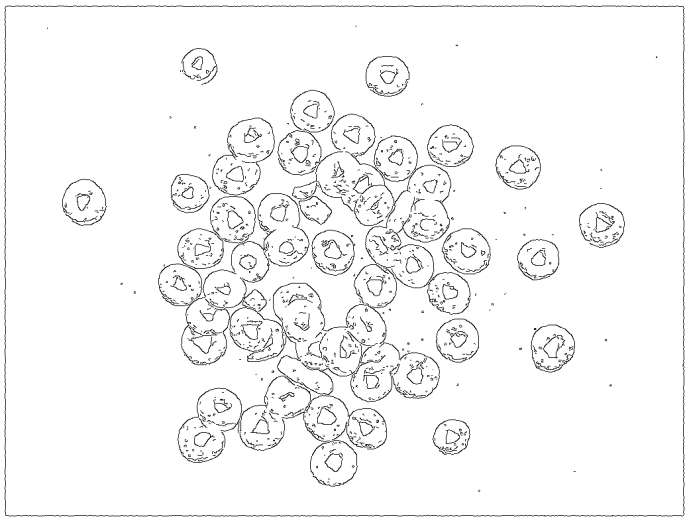
circuit-sobel.png

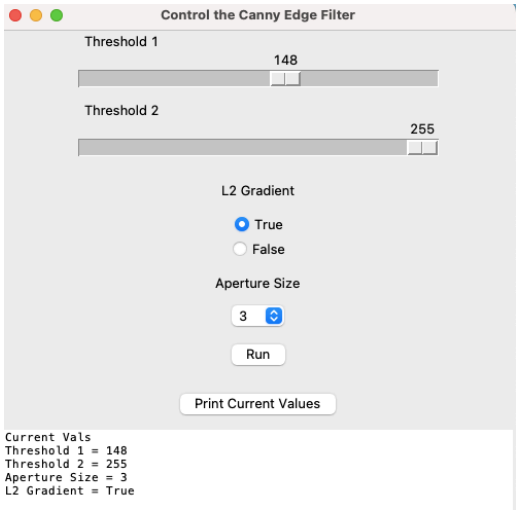


gear-sobel.png

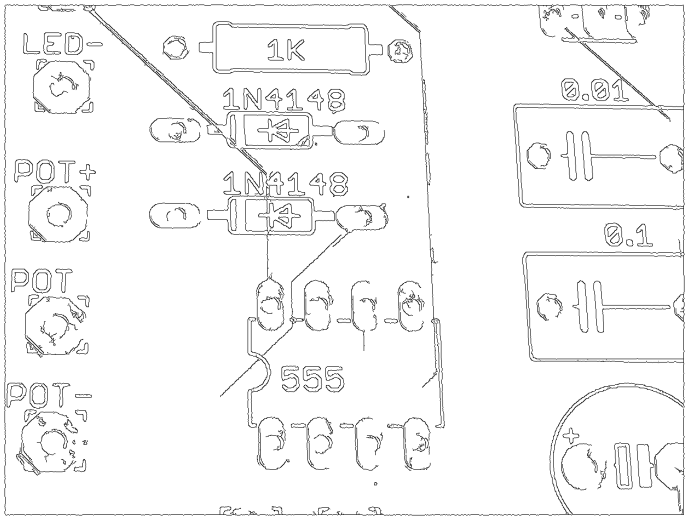
professor-sobel.png

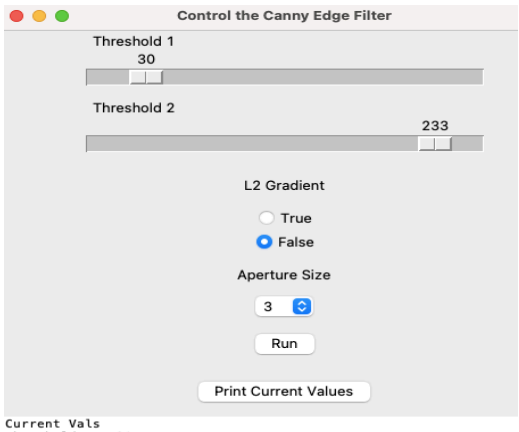
Canny Edge Input and Output



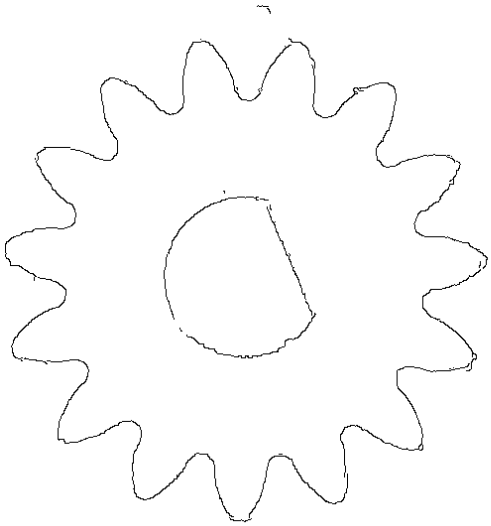


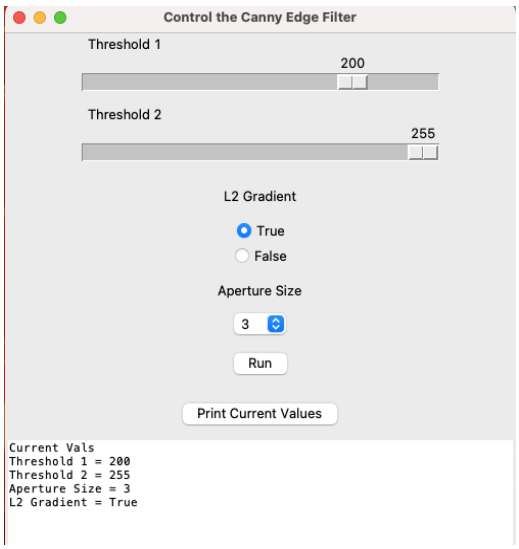
cheerios-canny.png



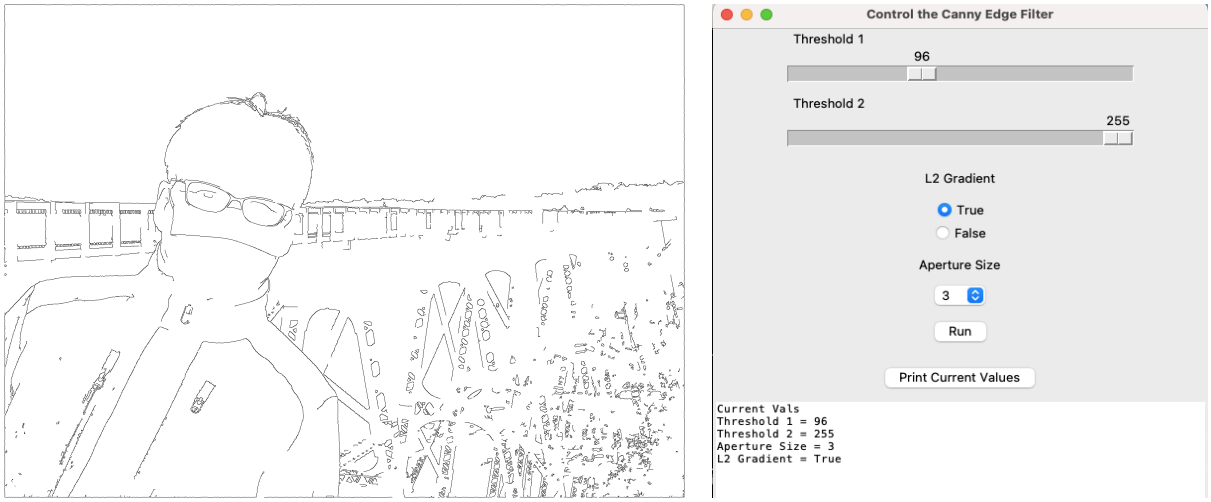


circuit-canny.png





gear-canny.png



professor-canny.png

Source Code

```
from ast import Lambda
import numpy as np
import cv2 as cv
from tkinter import *

def sobel (img,f):
    Gx = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
    Gy = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    sobel_img = np.zeros(shape=img.shape, dtype="uint8")
    img = cv.GaussianBlur(img, (3,3), cv.BORDER_DEFAULT)
    for i in range(3,img.shape[0] - 2):
        for j in range(3,img.shape[1] - 2):
            x = np.sum(Gx * img[i-1:i+2 , j-1:j+2])/4  # horizontal
            y = np.sum(Gy * img[i-1:i+2 , j-1:j+2])/4  # vertical
            sobel_img[i,j] = np.sqrt(x**2 + y**2)
    sobel_img = cv.bitwise_not(sobel_img)
    cv.imshow("Sobel Filtered Image", sobel_img/np.amax(sobel_img))
    cv.waitKey(0)
    cv.destroyAllWindows()
    f = f.split('.')
    cv.imwrite(f[0]+"-sobel."+f[1], sobel_img, )

def GUI(img,f):
    root = Tk()
    root.title("Control the Canny Edge Filter")
    root.geometry("500x500")
    s1 = IntVar()
    s2 = IntVar()
    s1.set(50)
    s2.set(200)
    l2 = BooleanVar()
    l2.set(False)
    apert = IntVar()
    apert.set(3)
    Scale(root, from_=0, to=255, label="Threshold 1", length=350 ,
variable=s1, orient=HORIZONTAL).pack()
    Scale(root, from_=0, to=255, label="Threshold 2", length=350
,variable=s2, orient=HORIZONTAL).pack(pady=10)
    Label(root, text="L2 Gradient").pack(pady=10)
    Radiobutton(root, text = "True", variable=l2, value=True).pack()
    Radiobutton(root, text = "False", variable=l2, value=False).pack()
    Label(root, text="Aperture Size").pack(pady=10)
```

```
    apertmenu  = OptionMenu(root, apert, 3,5,7)
    apertmenu.pack()
    execute = Button(root,text="Run", command= lambda : canny(img,
s1.get(), s2.get(), apert.get(), l2.get(),f))
    execute.pack(pady=10)
    vals = Text(root)
    valP = Button(root, text="Print Current Values", command = lambda:
vals.replace("1.0" ,"end",f"Current Vals\nThreshold 1 =
{s1.get()}\nThreshold 2 = {s2.get()}\nAperture Size = {apert.get()}\nL2
Gradient = {l2.get()} " ))
    valP.pack(pady=10)
    vals.pack()
    root.mainloop()

def canny(img,th1,th2,apert,l2,f):
    cv.destroyAllWindows()
    dst = cv.Canny(img, th1, th2, apertureSize=apert, L2gradient=l2)
    dst = cv.bitwise_not(dst)
    cv.imshow("Canny Filtered Image", dst)
    cv.waitKey(100)
    f = f.split('.')
    cv.imwrite(f[0]+"-canny."+f[1], dst)

f = input("Enter filename : ")
img = cv.imread(f)
img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
pad_img = np.pad(img, [(2,2),(2,2)], mode='constant',
constant_values=0)
ch = input(f"Sobel or Canny?\n")
if ch.lower() == "sobel":
    sobel(img,f)
elif ch.lower() == "canny":
    GUI(pad_img,f)
else:
    print("Invalid Input")
```

System Specifications

Operating System: macOS Monterey Version 12.5.1
Hardware: MacBook Air 2017 (Intel Core i5)
Python: Conda environment utilizing Python 3.9.1
IDE: Visual Studio Code