INDEX

PS3-1:

Code screenshots:

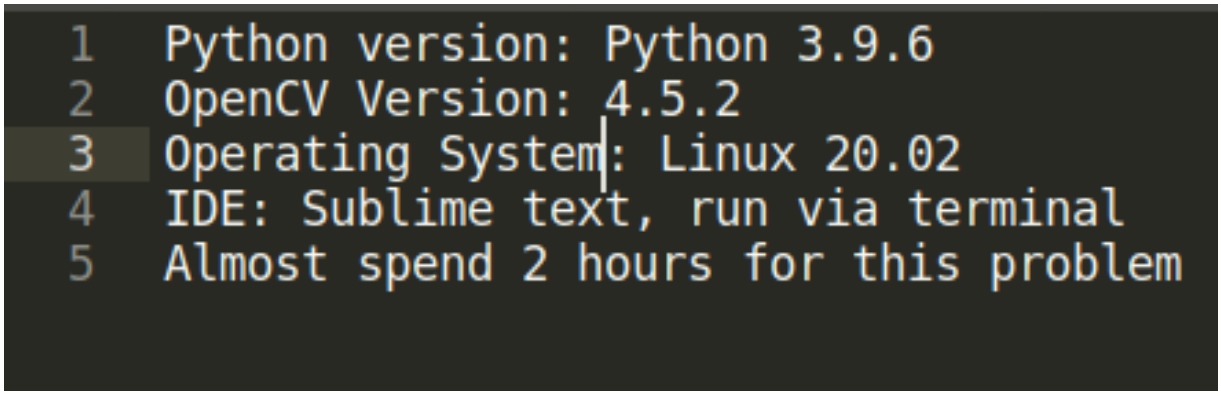
```python
import numpy as np
import cv2
import os


#A class that stores all the images including input image, smoothed image,
#sharped image and final improved image. The class also has functions that
#apply different filters for smoothing and sharpening according to choice
class Solution():
    #constructor that declares and assigns the input image
    def __init__(self,filename):
        self.input_image = cv2.imread(filename)

    #This function smoothes the image according to choice and assigns it to
    #the member variable self.smoothed_image
    def smooth(self,smooth_type,image):
        #uses gaussian filter of kernel 3
        if (smooth_type == 'GaussianBlur'):
            self.smoothed_image = cv2.GaussianBlur(image,(3,3),sigmaX=2)

        #applies median filter of size 3
        elif (smooth_type == 'medianBlur'):
            self.smoothed_image = cv2.medianBlur(image,3)

        #applies boxFilter of size 3, -1 indicates input and output of same dimensions
        elif (smooth_type == 'boxFilter'):
            self.smoothed_image = cv2.boxFilter(image,-1,(3,3))

        #applies the normal blur which takes average around the kernel of size(3,3)
        elif (smooth_type == 'blur'):
            self.smoothed_image = cv2.blur(image,(3,3))

        #applies the bilateral filter of diameter 15, sigma of color space & coordinate
        #space respectively
        elif (smooth_type == 'bilateralFilter'):
            self.smoothed_image = cv2.bilateralFilter(image,15,75,75)

        elif (smooth_type == 'unsharpMasking'):
            temp_img = cv2.GaussianBlur(image,(3,3),sigmaX=1)
            self.smoothed_image = cv2.addWeighted(temp_img,-.5,image,1.7,0)

        else :
            raise Exception("Invalid choice")
```

```
    #This function sharps the image according to the sharpening matrix of choice
    #and assigns the sharped image to self.sharp_image
    def sharp(self,sharp_matrix,image):

        #sharps by taking the sharp_matrix and image as input
        self.sharp_image = cv2.filter2D(image,-1,sharp_matrix)


def main():

    #For image 1, pcb.png
    filename = "/home/akshay/Downloads/CV/ps-3-q/ps3-images/pcb.png"
    #Creating an object for the pcb image
    image_improve_pcb = Solution(filename)
    #smoothing it using median filter
    image_improve_pcb.smooth('medianBlur',image_improve_pcb.input_image)
    #sharpening the smoothed it using the folowing matrix
    sharp_matrix = np.asarray([[-1,-1,-1],[-1,9,-1],[-1,-1,-1]])
    image_improve_pcb.sharp(sharp_matrix,image_improve_pcb.smoothed_image)

    #writing the image
    first_name = os.path.basename(filename)
    first_name = first_name.split('.',1)[0]
    write_filename = first_name + "-improved.png"

    if(cv2.imwrite(write_filename,image_improve_pcb.sharp_image)):
        print("Successfully saved:",write_filename)
    else:
        print("Save Unsuccessfull")
    #printing the input as well as improved image
    cv2.imshow("input image pcb",image_improve_pcb.input_image)
    cv2.imshow("final improved image pcb",image_improve_pcb.sharp_image)


    #for image 2 golf.png
    filename = "/home/akshay/Downloads/CV/ps-3-q/ps3-images/golf.png"
    #Creating an object for the pcb image
    image_improve_golf = Solution(filename)
    #smoothing it using median filter
    image_improve_golf.smooth('medianBlur',image_improve_golf.input_image)
    #sharpening the smoothed it using the folowing matrix
    sharp_matrix = np.asarray([[0,-1,0],[-1,5,-1],[0,-1,0]])
    image_improve_golf.sharp(sharp_matrix,image_improve_golf.smoothed_image)

    #writing the image
```

```
first_name = os.path.basename(filename)
first_name = first_name.split('.',1)[0]
write_filename = first_name + "-improved.png"

if(cv2.imwrite(write_filename,image_improve_golf.sharp_image)):
    print("Successfully saved: ",write_filename)
else:
    print("Save Unsuccessfull")

#printing the input as well as improved image
cv2.imshow("input image golf",image_improve_golf.input_image)
cv2.imshow("final improved image golf",image_improve_golf.sharp_image)


#for image 3 pots.png
filename = "/home/akshay/Downloads/CV/ps-3-q/ps3-images/pots.png"
#Creating an object for the pots image
image_improve_pots = Solution(filename)
#smoothing it using median filter
#image_improve_pots.smooth('unsharpMasking',image_improve_pots.input_image)
#sharpening the smoothed it using the folowing matrix
sharp_matrix = np.asarray([[-1,-1,-1],[-1,9,-1],[-1,-1,-1]])
image_improve_pots.sharp(sharp_matrix,image_improve_pots.input_image)
image_improve_pots.smooth('unsharpMasking',image_improve_pots.sharp_image)

#writing the image
first_name = os.path.basename(filename)
first_name = first_name.split('.',1)[0]
write_filename = first_name + "-improved.png"

if(cv2.imwrite(write_filename,image_improve_pots.smoothed_image)):
    print("Successfully saved: ",write_filename)
else:
    print("Save Unsuccessfull")

#printing the input as well as improved image
cv2.imshow("input image pots",image_improve_pots.input_image)
cv2.imshow("final improved image pots",image_improve_pots.smoothed_image)


#for image 4 rainbow.png
filename = "/home/akshay/Downloads/CV/ps-3-q/ps3-images/rainbow.png"
#Creating an object for the rainbow image
image_improve_rainbow = Solution(filename)
#smoothing it using median filter
```

```
    image_improve_rainbow.smooth('bilateralFilter',image_improve_rainbow.input_image)
    #sharpening the smoothed it using the folowing matrix
    sharp_matrix = np.asarray([[0,-1,0],[-1,5,-1],[0,-1,0]])
    image_improve_rainbow.sharp(sharp_matrix,image_improve_rainbow.smoothed_image)

    #writing the image
    first_name = os.path.basename(filename)
    first_name = first_name.split('.',1)[0]
    write_filename = first_name + "-improved.png"

    if(cv2.imwrite(write_filename,image_improve_rainbow.sharp_image)):
        print("Successfully saved: ",write_filename)
    else:
        print("Save Unsuccessfull")


    #printing the input as well as improved image
    cv2.imshow("input image rainbow",image_improve_rainbow.input_image)
    cv2.imshow("final improved image rainbow",image_improve_rainbow.sharp_image)


if __name__ == '__main__':
    main()
    cv2.waitKey(0)
```

Screenshot of Readme.txt

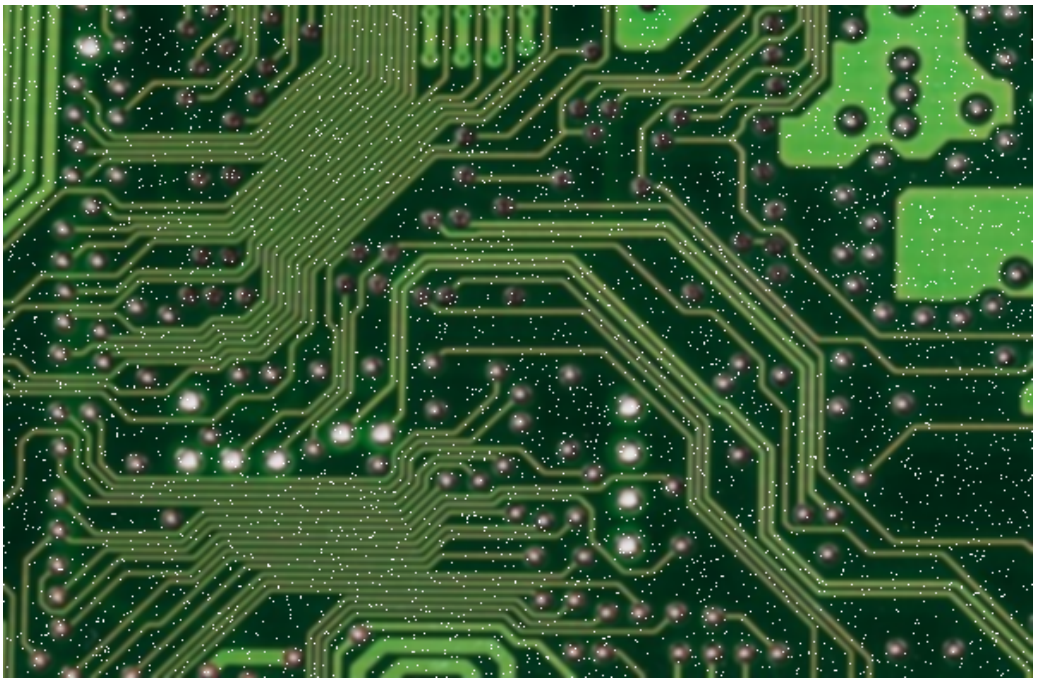


Output Images

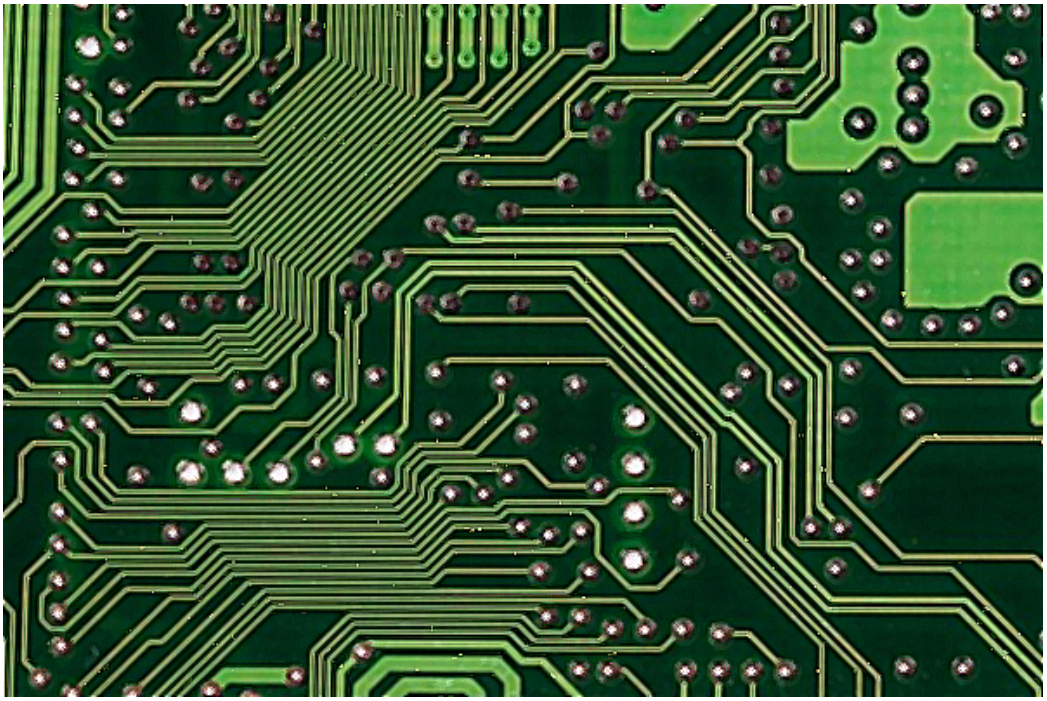
For Image 1 ***pcb.png****:*

Filters Applied

- *MedianBlur of ksize=3: medianblur removes salt and pepper noise*
- *Sharpening Matrix = [[-1,-1,-1]*
  [-1,9,-1]
  [-1,-1,-1]]



pcb.png



pcb-improved.png

For image 2: **golf.png**

Filters Applied

- *MedianBlur of ksize=3: medianblur removes salt and pepper noise*
- *Sharpening Matrix* = [[0,-1,0]
  [-1,5,-1]
  [0,-1,0]]

golf.png

below golf-improved.png

For image 3: **pots.png**

Filters Applied (in order)

- *Sharpening Matrix* = [[-1,-1,-1]
  [-1,9,-1]
  [-1,-1,-1]]: applied to sharpening filter first because input image was blurred already
- Unsharping kernel
  - First a Gaussian filter of ksize = (3,3) and sigmaX=2
  - Weighted Adding using the sharpened image and blurred image from the previous step with a parameter of weight -0.5 and 2.0 respectively for blurred and sharpened image

pots.png

pots-improved.png

For image 4: **rainbow.png**

Filters Applied

- *bilateralFilter of diameter 15, sigma of color space & coordinate space 75,75 respectively. A bilateral filter maintains the edge information.*
- *Sharpening Matrix* = [[0,-1,0]
  [-1,5,-1]
  [0,-1,0]]

rainbow.png

rainbow-improved.png

PS3-2

Screenshot of ***codes for Sobel filter****:*

```
#Applying sobel filter without cv2
import numpy as np
import cv2
import os


class SobelEdgeDetection():
    def __init__(self,filename):
        #constructor takes filename and assigns the orginal image and displays,
        #converts the original image to gray to apply sobel in only one channel
        self.__filename = filename
        self.__original_image = cv2.imread(self.__filename)
        self.__gray_image = cv2.cvtColor(self.__original_image,cv2.COLOR_BGR2GRAY)
        n_gray_image = np.float64(self.__gray_image)
        #padding with 1 as i am using 3*3 kernel, padding is done to work with border
        #elements
        self.__padded_image = np.pad(n_gray_image,((1,1),(1,1)),constant_values=0)
        self.__gradient_y_image = np.zeros_like(n_gray_image)
        self.__gradient_x_image = np.zeros_like(n_gray_image)
        self.__gradient_image = np.zeros_like(n_gray_image)


    #function that does the filter 2d operation in for loops
    def convol(self):
        for i in range(self.__gray_image.shape[0]):
            for j in range(self.__gray_image.shape[1]):
                #multiplying element wise and adding, by selecting a 3*3 from the padded image
                self.__gradient_y_image[i,j] = np.sum(self.__padded_image[i:i+3,j:j+3]*self.__filter_y)
                self.__gradient_x_image[i,j] = np.sum(self.__padded_image[i:i+3,j:j+3]*self.__filter_x)
                #final gradient is taken as sqrt of sum of squares of x and y
                self.__gradient_image[i,j] = np.sqrt(np.power(self.__gradient_x_image[i,j],2) + np.power(self.__gradient_y_image[i,j],2

    def sobel(self,filter_x,filter_y):
        #assigns the value of filter to the member variables
        self.__filter_y = filter_y
        self.__filter_x = filter_x
        #calls the convolution function
        self.convol()
        #ALl the images are in float, taking their absolute and converting them into uint8
        self.__gradient_x_image = np.uint8(np.absolute(self.__gradient_x_image))
        self.__gradient_y_image = np.uint8(np.absolute(self.__gradient_y_image))
        self.__gradient_image = np.uint8(self.__gradient_image)

    def flip_image(self):
        #edges generated is white on black, making it black on white according to the question
        self.__flipped_image = np.uint8(np.where(self.__gradient_image > 127,0,255))

    def display_final(self):
        cv2.imshow("input image",self.__original_image)
        cv2.imshow("x",self.__gradient_y_image)
        cv2.imshow("y",self.__gradient_x_image)
        cv2.imshow("final_edge_image",self.__gradient_image)
        cv2.imshow("final_edge_image_flipped",self.__flipped_image)

    #finding the first name of the loaded image
```

```
    def __find_write_filename(self):
        first_name = os.path.basename(self.__filename)
        first_name = first_name.split('.',1)[0]
        return first_name

    def write_image(self):
        self.__write_filename = self.__find_write_filename() + "-sobel.png"
        if(cv2.imwrite(self.__write_filename,self.__flipped_image)):
            print("Successfully saved")
        else:
            print("Unsuccessfull")

if __name__ == "__main__":
    #defining the 3*3 kernel for sobel filetring in y and x respectively
    sobel_filter_x = np.asarray([[-1.,0,1.],[-2.,0,2.],[-1.,0,1.]])
    sobel_filter_y = np.asarray([[1.,2.,1.],[0.,0.,0.],[-1.,-2.,-1.]])
    filename = "/home/akshay/Downloads/CV/ps-3-q/ps3-images/professor.png"

    #creating the object
    edge_detection = SobelEdgeDetection(filename)
    #This operation takes some time while running
    edge_detection.sobel(sobel_filter_x,sobel_filter_y)
    #makes black edges on white background
    edge_detection.flip_image()
    edge_detection.display_final()
    edge_detection.write_image()

    cv2.waitKey(0)
```

Screenshot of canny codes

```
import cv2
import numpy as np
import os
#callback of trackbar
def nothing(x):
    pass

#reading the input image

filename = "/home/akshay/Downloads/CV/ps-3-q/ps3-images/gear.png"
input_image = cv2.imread("/home/akshay/Downloads/CV/ps-3-q/ps3-images/gear.png")

cv2.namedWindow('image',  cv2.WINDOW_KEEPRATIO)
# create trackbars for color change
cv2.createTrackbar('aperture','image',1,2,nothing)
#0-False,1-True for L2norm
cv2.createTrackbar('L2norm','image',0,1,nothing)
cv2.createTrackbar('threshold1','image',0,255,nothing)
cv2.createTrackbar('threshold2','image',0,255,nothing)
#aperture size limited in (3,5,7) in Canny function
#so the result from trackbar aperture is modified as
#2*aperture+3

threshold1,threshold,L2,aperture = 0,0,0,0
#Infinite loop activated whenever a trackbar value is changed
while(1):
    # get current positions of four trackbars
    threshold1 = cv2.getTrackbarPos('threshold1','image')
    threshold2 = cv2.getTrackbarPos('threshold2','image')
    aperture = cv2.getTrackbarPos('aperture','image')
    L2 = bool(cv2.getTrackbarPos('L2norm','image'))
    #applying cv2 canny
    new_image = cv2.Canny(input_image,threshold1,threshold2,apertureSize=2*aperture+3,L2gradient=L2)
    # flipping
    new_image = np.uint8(np.where(new_image>127,0,255))
    cv2.imshow("image",new_image)
    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break
cv2.destroyAllWindows()
#saving the image
first_name = os.path.basename(filename)
first_name = first_name.split('.',1)[0]
write_filename = first_name + "-canny.png"
```

```
print(threshold1,threshold2,aperture,L2)
if(cv2.imwrite(write_filename,new_image)):
    print("Successfully saved")
else:
    print("Save Unsuccessfull")
```

Screenshot of readme.txt:

```
Python version: Python 3.9.6
OpenCV Version: 4.5.2
Operating System: Linux 20.02
IDE: Sublime text, run via terminal
Almost spend 4 hours for this problem
```

Output Images

Canny Edge Detection

Components in Trackbar:

1. Threshold1: range 0-255
2. Threshold2: range 0-255
3. Aperture: range 0-2
    a. The output from the trackbar of the aperture is multiplied by 2 and 3 is added.
    b. So 0,1,2 becomes 3,5,7 which are the acceptable range in open cv
4. L2norm: Either 0 0r 1, 0 is False, 1 is True

NB. For aperture=2(ie ksize=7), you must adjust threshlod1 and threshold2 before changing the aperture to 2. I don't know the error here, but when you set the aperture to the last value ie the limit(here 2), the change in threshold does not change the edges. But for all other aperture values ie(0,1) you can adjust the aperture and threshold values in any order.

Output Images from Canny and Sobel:

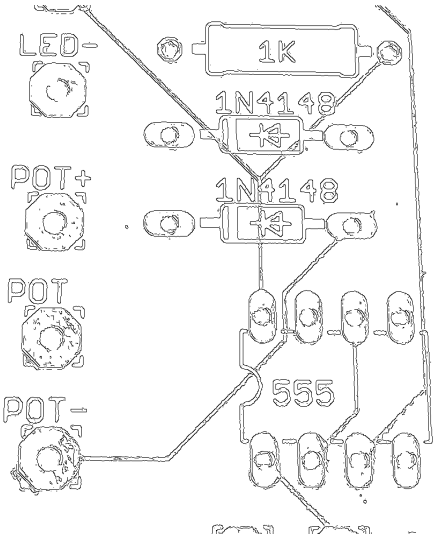
Value of parameters used for circuit.png in canny:

Threshold1: 71
Threshold2: 101
*Aperature: 0 (ie 2*0+3=3 kernel size)*
L2norm: True

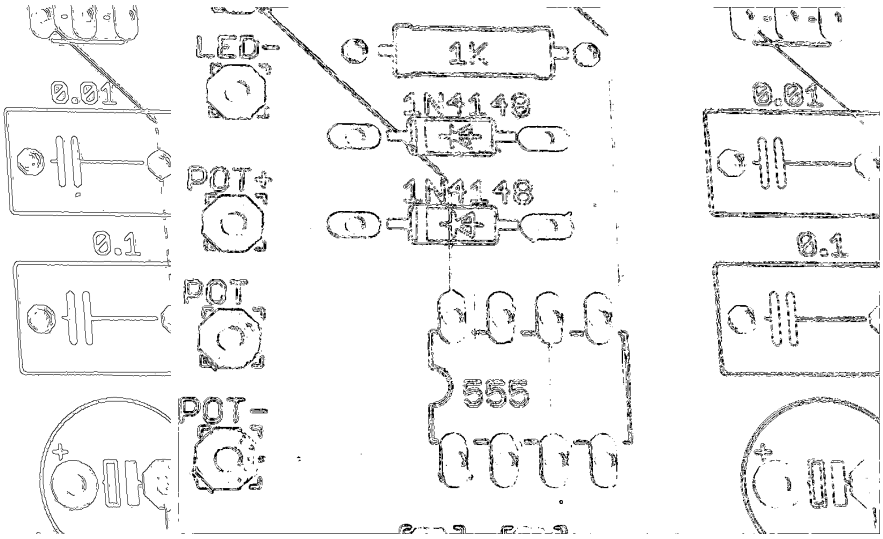
Value of parameters used for the circuit.png in sobel:

sobel_filter_x=[[-1.,0,1.],[-2.,0,2.],[-1.,0,1.]]
sobel_filter_y =[[1.,2.,1.],[0.,0.,0.],[-1.,-2.,-1.]]
Used L2 norm to combine the two.

Results: Well-defined thin edges in Canny while the edges in Sobel in thick and invisible in some areas. But even the smallest edges are visible in canny.png. All the alphabets are numbers in canny are easily readable.



circuit-canny.png



circuit-sobel.png

CHEERIOS.PNG

Value of parameters used for cheerios.png in canny:

Threshold1: 137

Threshold2: 255

*Aperature: 0 (ie 2*0+3=3 kernel size)*

L2norm: True

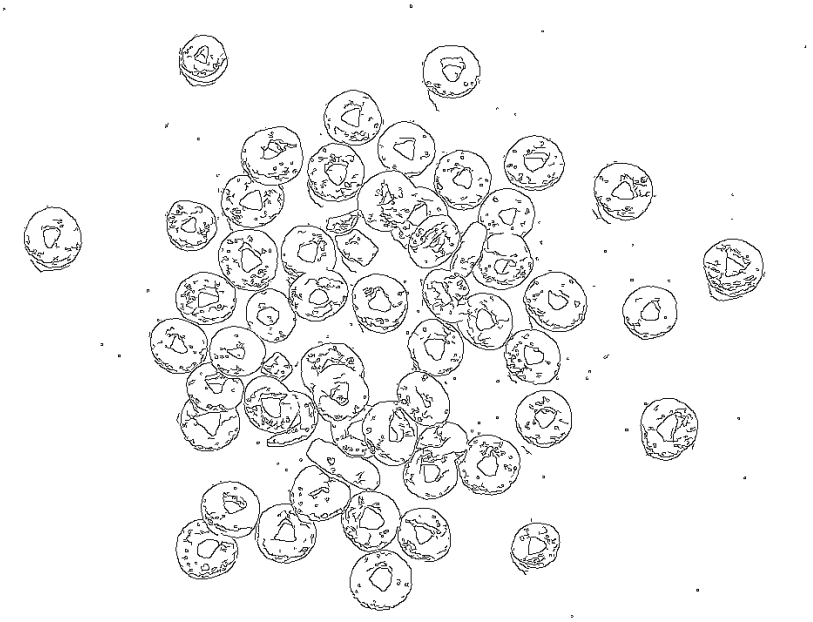
Value of parameters used for the cheerios.png in sobel:

sobel_filter_x=[[-1.,0,1.],[-2.,0,2.],[-1.,0,1.]]

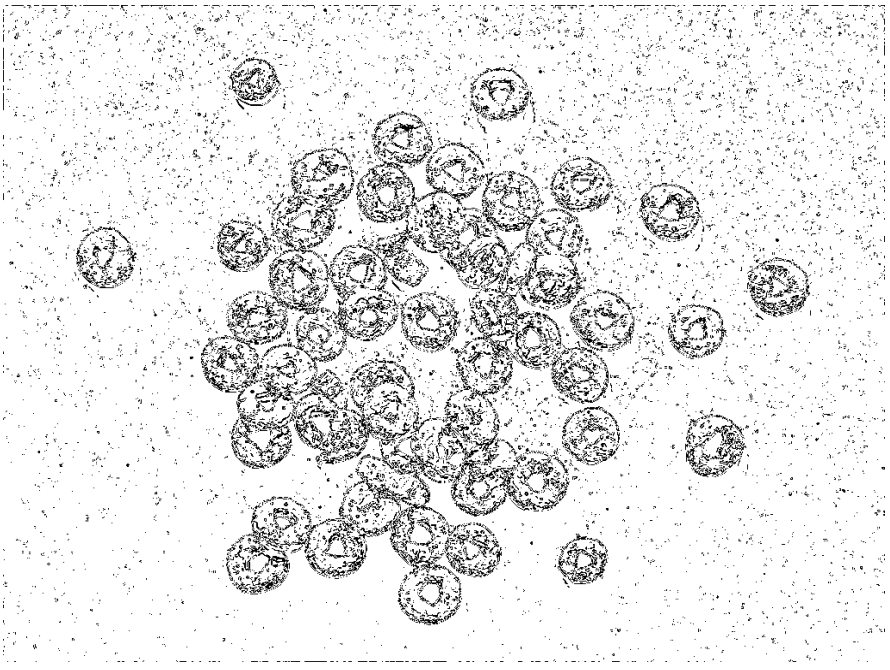
sobel_filter_y =[[1.,2.,1.],[0.,0.,0.],[-1.,-2.,-1.]]

Used L2 norm to combine the two.

Results: well-defined edges in canny.png while Sobel contains fake edges which occur outside the part where cheerios are present



cheerios-canny.png



cheerios-sobel.png

PROFESSOR.PNG

Value of parameters used for professor.png in canny:

Threshold1: 70

Threshold2: 87

*Aperature: 0 (ie 2*0+3=3 kernel size)*

L2norm: False

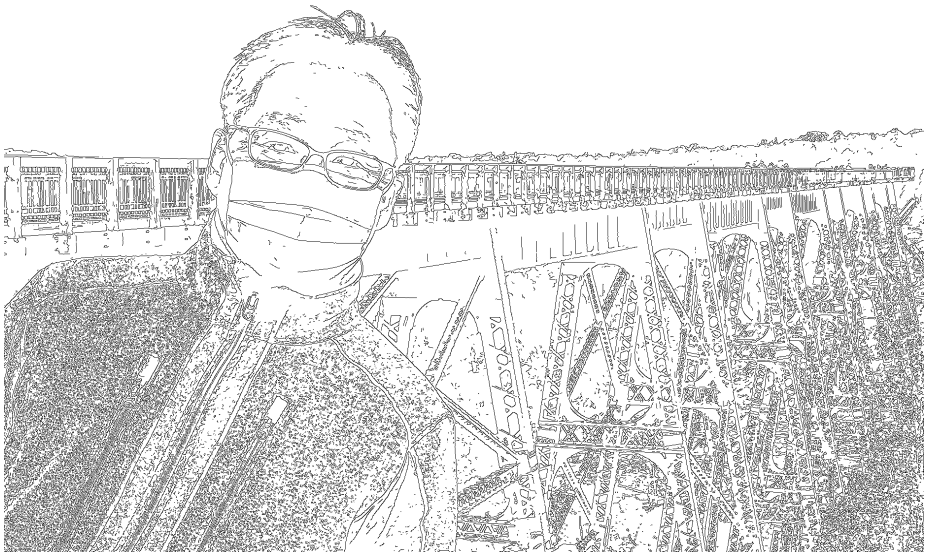
Value of parameters used for the professor.png in sobel:

sobel_filter_x=[[-1.,0,1.],[-2.,0,2.],[-1.,0,1.]]

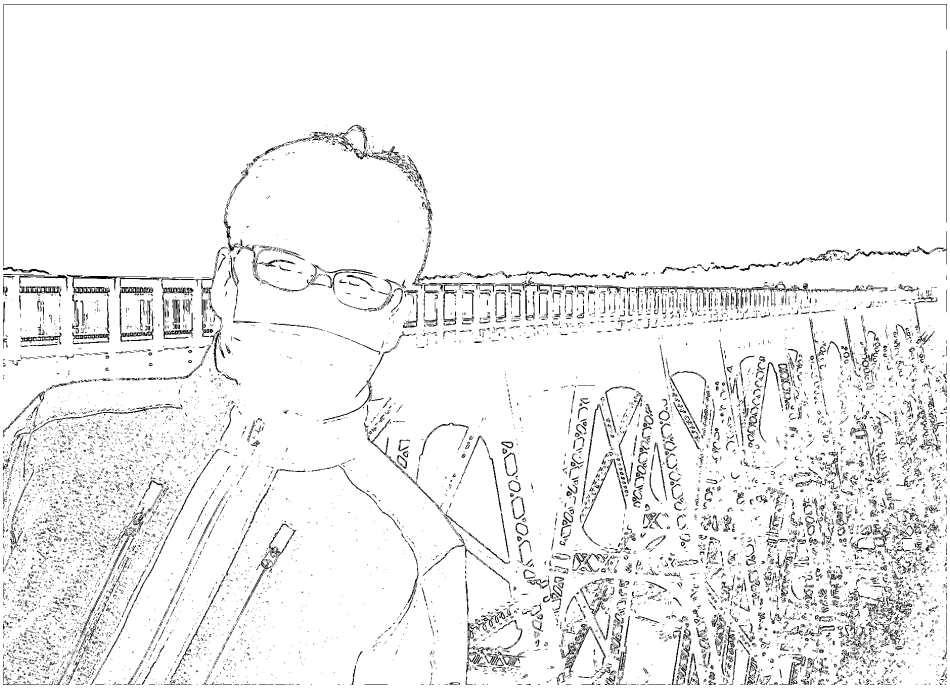
sobel_filter_y =[[1.,2.,1.],[0.,0.,0.],[-1.,-2.,-1.]]

Used L2 norm to combine the two.

Results: The canny edge detection provides more detailed edges including the hair, eyes, and masks part than the Sobel image



professor-canny.png



professor-sobel.png

GEAR.PNG

Value of parameters used for gear.png in canny:

Threshold1: 60

Threshold2: 104

*Aperature: 0 (ie 2*0+3=3 kernel size)*

L2norm: True

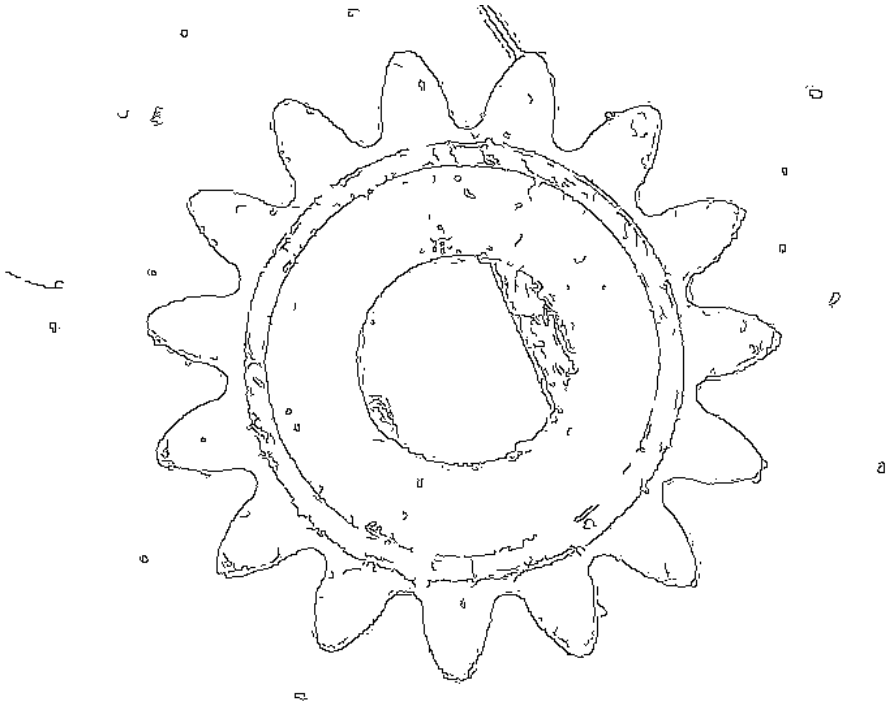
Value of parameters used for the gear.png in sobel:

sobel_filter_x=[[-1.,0,1.],[-2.,0,2.],[-1.,0,1.]]

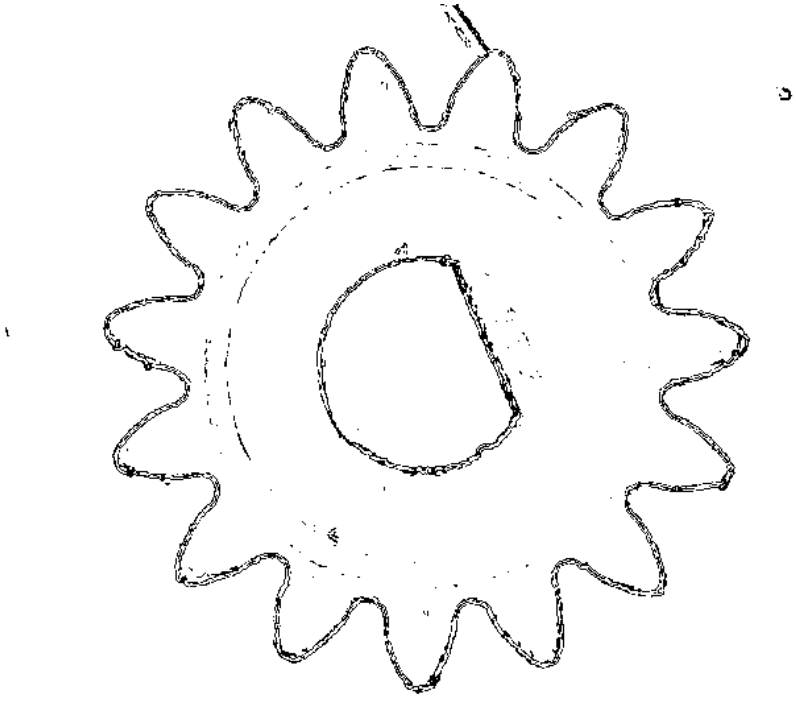
sobel_filter_y =[[1.,2.,1.],[0.,0.,0.],[-1.,-2.,-1.]]

Used L2 norm to combine the two.

Results: Canny provides crisp and clear edges than thick and uncertain edges in Sobel. But in this case Sobel edges are also not bad.



gear-canny.png



gear-sobel.png