**Assignment 2**

**Canny Edge Detector**

By – Harsh Sharma

**Code:**

1. **Include code with comments in the report.**

from scipy import misc

from scipy import ndimage

import numpy as np

import matplotlib.pyplot as plt

import imageio

import warnings

warnings.filterwarnings('ignore')

#loading the image

lion = imageio.imread("daft.jpg")

plt.imshow(lion, cmap = plt.get\_cmap('gray'))

plt.show()

Two people wearing black and gold helmets

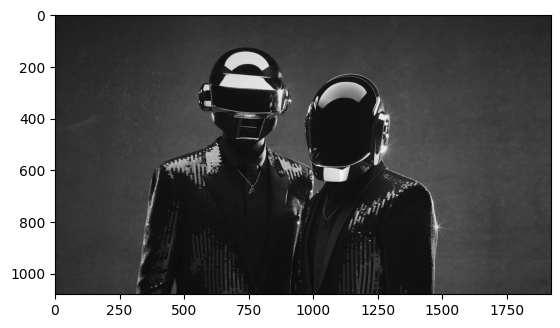
Description automatically generated with low confidence

# Convert color image to grayscale to help

lion\_gray = np.dot(lion[...,:3], [0.299, 0.587, 0.114])

plt.imshow(lion\_gray, cmap = plt.get\_cmap('gray'))

plt.show()

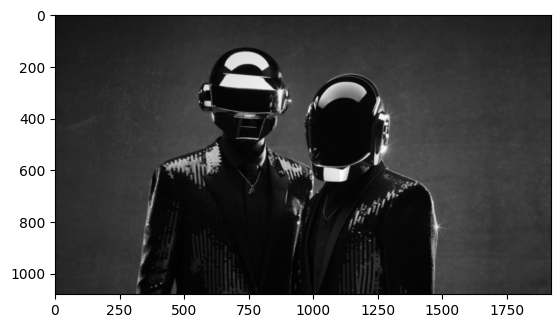


# Blur the grayscale image

lion\_gray\_blurred = ndimage.gaussian\_filter(lion\_gray, sigma=1.4)

plt.imshow(lion\_gray\_blurred, cmap = plt.get\_cmap('gray'))

plt.show()



# Apply Sobel Filter using the convolution operation

def SobelFilter(img, direction):

if(direction == 'x'):

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

Res = ndimage.convolve(img, Gx)

if(direction == 'y'):

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

Res = ndimage.convolve(img, Gy)

return Res

# Normalize the pixel array, so that values are <= 1

def Normalize(img):

img = img/np.max(img)

return img

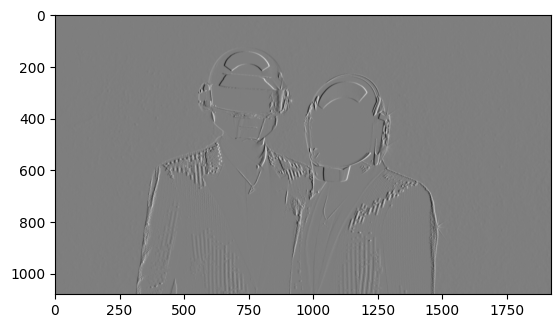
# Apply Sobel Filter in X direction

gx = SobelFilter(lion\_gray\_blurred, 'x')

gx = Normalize(gx)

plt.imshow(gx, cmap = plt.get\_cmap('gray'))

plt.show()



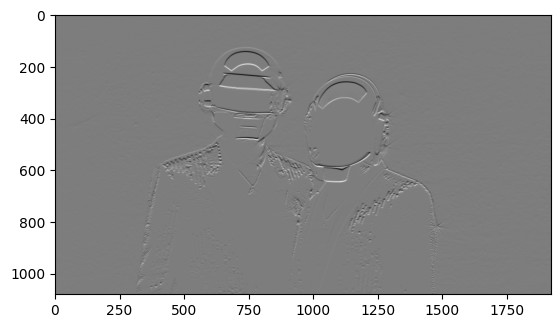
# Apply Sobel Filter in Y direction

gy = SobelFilter(lion\_gray\_blurred, 'y')

gy = Normalize(gy)

plt.imshow(gy, cmap = plt.get\_cmap('gray'))

plt.show()



# Apply the Sobel Filter using the inbuilt function of scipy, this was done to verify the values obtained from above

dx = ndimage.sobel(lion\_gray\_blurred, axis=1) # horizontal derivative

dy = ndimage.sobel(lion\_gray\_blurred, axis=0) # vertical derivative

# Plot the derivative filter values obtained using the inbuilt function

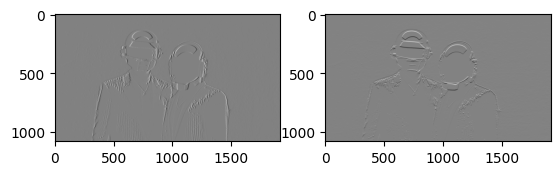
plt.subplot(121)

plt.imshow(dx, cmap = plt.get\_cmap('gray'))

plt.subplot(122)

plt.imshow(dy, cmap = plt.get\_cmap('gray'))

plt.show()



Gradient

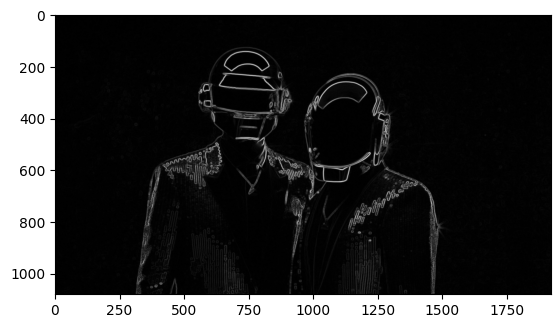
# Calculate the magnitude of the gradients obtained

Mag = np.hypot(gx,gy)

Mag = Normalize(Mag)

plt.imshow(Mag, cmap = plt.get\_cmap('gray'))

plt.show()



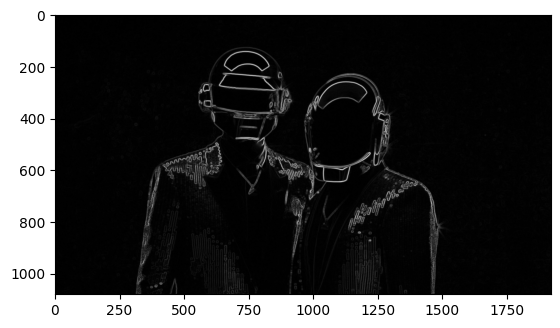
# Calculate the magnitude of the gradients obtained using the inbuilt function, again done to verify the correctness of the above value

mag = np.hypot(dx,dy)

mag = Normalize(mag)

plt.imshow(mag, cmap = plt.get\_cmap('gray'))

plt.show()



# Calculate direction of the gradients

Gradient = np.degrees(np.arctan2(gy,gx))

# Calculate the direction of the gradients obtained using the inbuilt sobel function

gradient = np.degrees(np.arctan2(dy,dx))

# NON-MAX SUPPRESSION

#Non Maximum Suppression

# This is done to get thin edges

def NonMaxSupWithInterpol(Gmag, Grad, Gx, Gy):

NMS = np.zeros(Gmag.shape)

for i in range(1, int(Gmag.shape[0]) - 1):

for j in range(1, int(Gmag.shape[1]) - 1):

if((Grad[i,j] >= 0 and Grad[i,j] <= 45) or (Grad[i,j] < -135 and Grad[i,j] >= -180)):

yBot = np.array([Gmag[i,j+1], Gmag[i+1,j+1]])

yTop = np.array([Gmag[i,j-1], Gmag[i-1,j-1]])

x\_est = np.absolute(Gy[i,j]/Gmag[i,j])

if (Gmag[i,j] >= ((yBot[1]-yBot[0])\*x\_est+yBot[0]) and Gmag[i,j] >= ((yTop[1]-yTop[0])\*x\_est+yTop[0])):

NMS[i,j] = Gmag[i,j]

else:

NMS[i,j] = 0

if((Grad[i,j] > 45 and Grad[i,j] <= 90) or (Grad[i,j] < -90 and Grad[i,j] >= -135)):

yBot = np.array([Gmag[i+1,j] ,Gmag[i+1,j+1]])

yTop = np.array([Gmag[i-1,j] ,Gmag[i-1,j-1]])

x\_est = np.absolute(Gx[i,j]/Gmag[i,j])

if (Gmag[i,j] >= ((yBot[1]-yBot[0])\*x\_est+yBot[0]) and Gmag[i,j] >= ((yTop[1]-yTop[0])\*x\_est+yTop[0])):

NMS[i,j] = Gmag[i,j]

else:

NMS[i,j] = 0

if((Grad[i,j] > 90 and Grad[i,j] <= 135) or (Grad[i,j] < -45 and Grad[i,j] >= -90)):

yBot = np.array([Gmag[i+1,j] ,Gmag[i+1,j-1]])

yTop = np.array([Gmag[i-1,j] ,Gmag[i-1,j+1]])

x\_est = np.absolute(Gx[i,j]/Gmag[i,j])

if (Gmag[i,j] >= ((yBot[1]-yBot[0])\*x\_est+yBot[0]) and Gmag[i,j] >= ((yTop[1]-yTop[0])\*x\_est+yTop[0])):

NMS[i,j] = Gmag[i,j]

else:

NMS[i,j] = 0

if((Grad[i,j] > 135 and Grad[i,j] <= 180) or (Grad[i,j] < 0 and Grad[i,j] >= -45)):

yBot = np.array([Gmag[i,j-1] ,Gmag[i+1,j-1]])

yTop = np.array([Gmag[i,j+1] ,Gmag[i-1,j+1]])

x\_est = np.absolute(Gy[i,j]/Gmag[i,j])

if (Gmag[i,j] >= ((yBot[1]-yBot[0])\*x\_est+yBot[0]) and Gmag[i,j] >= ((yTop[1]-yTop[0])\*x\_est+yTop[0])):

NMS[i,j] = Gmag[i,j]

else:

NMS[i,j] = 0

return NMS

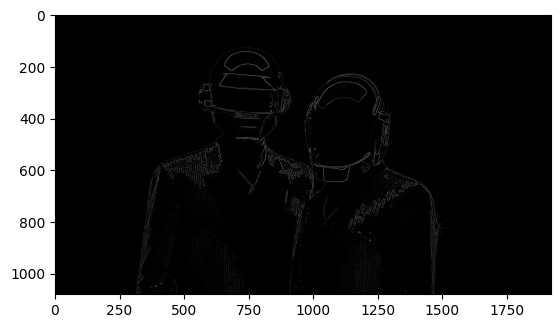
# Get the Non-Max Suppressed output

NMS = NonMaxSupWithInterpol(Mag, Gradient, gx, gy)

NMS = Normalize(NMS)

plt.imshow(NMS, cmap = plt.get\_cmap('gray'))

plt.show()



# Hysteresis thresholding

# Threshold Hysterisis

def DoThreshHyst(img):

highThresholdRatio = 0.2

lowThresholdRatio = 0.15

GSup = np.copy(img)

h = int(GSup.shape[0])

w = int(GSup.shape[1])

highThreshold = np.max(GSup) \* highThresholdRatio

lowThreshold = highThreshold \* lowThresholdRatio

x = 0.1

oldx=0

while(oldx != x):

oldx = x

for i in range(1,h-1):

for j in range(1,w-1):

if(GSup[i,j] > highThreshold):

GSup[i,j] = 1

elif(GSup[i,j] < lowThreshold):

GSup[i,j] = 0

else:

if((GSup[i-1,j-1] > highThreshold) or

(GSup[i-1,j] > highThreshold) or

(GSup[i-1,j+1] > highThreshold) or

(GSup[i,j-1] > highThreshold) or

(GSup[i,j+1] > highThreshold) or

(GSup[i+1,j-1] > highThreshold) or

(GSup[i+1,j] > highThreshold) or

(GSup[i+1,j+1] > highThreshold)):

GSup[i,j] = 1

x = np.sum(GSup == 1)

GSup = (GSup == 1) \* GSup

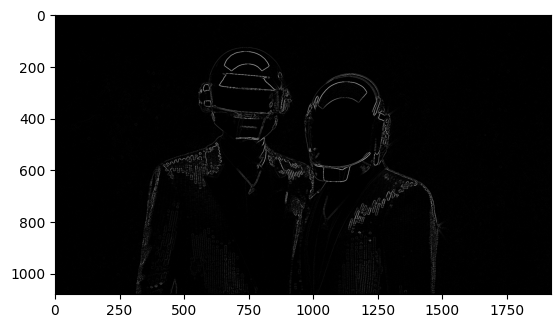
return GSup

# The output of canny edge detection

Final\_Image = DoThreshHyst(NMS)

plt.imshow(Final\_Image, cmap = plt.get\_cmap('gray'))

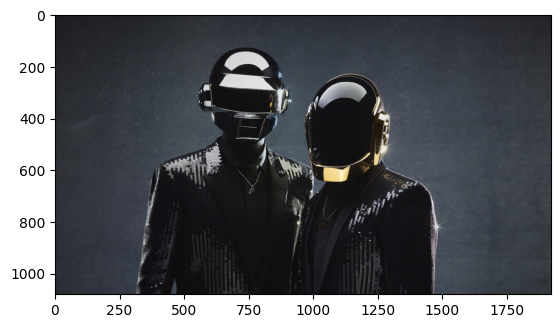
plt.show()



1. **Choose at least three images for showing your result.**

INPUT IMAGE CANNY EDGE

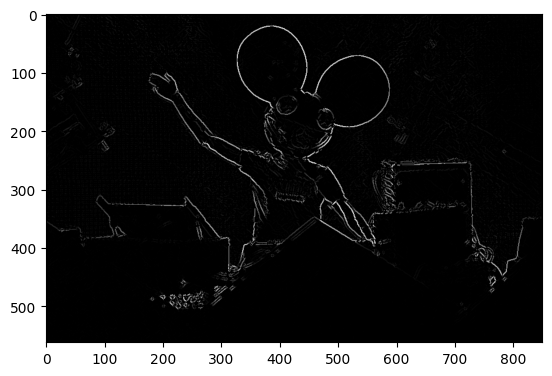
1. Daft Punk

 A picture containing text, screenshot, black

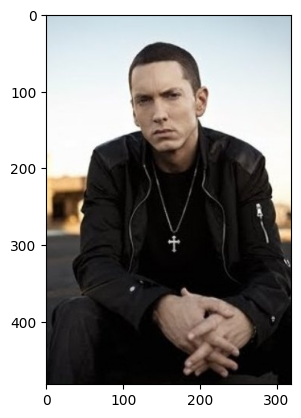
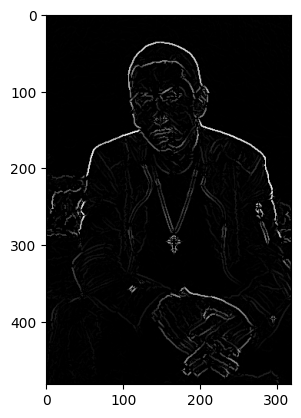
Description automatically generated

1. Deadmau5

A picture containing text, screenshot, concert, music

Description automatically generated 

1. Eminem

1. **Apply different sigma and different thresholding (high-low) and discuss the results.**

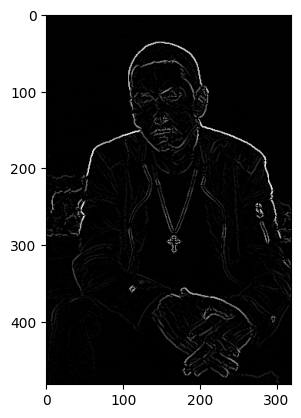
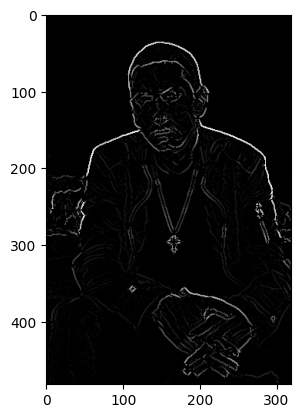
Input Image

A picture containing text, person, clothing, human face

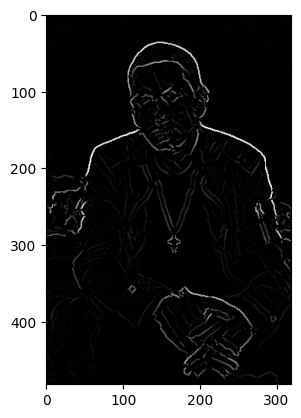
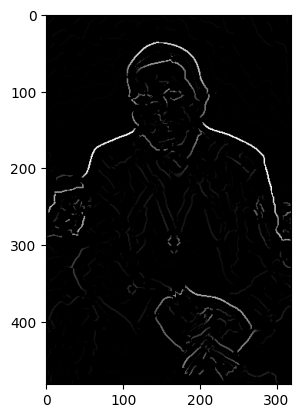
Description automatically generated

Varying Sigma

Sigma = 1 Sigma 1.5

Sigma = 2.5 Sigma = 5

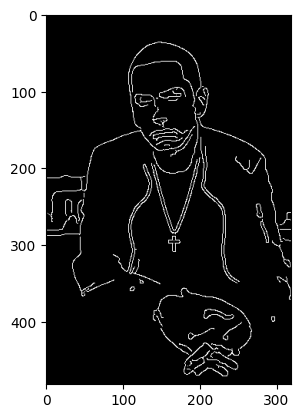
 

We get the best results when sigma is 1.5. As we increase the value of sigma, the images gets more blurry and hence less edges are detected. But if reduce the value of sigma, the image is blurred less and hence more noise is present which gives poor result.

**Different Thresholding**

**High Low**

A silhouette of a person

Description automatically generated with low confidence 

**4.** **If you are using libraries that have canny edge detector compare your implementation of canny with the method and discuss the change.**

import numpy as np

import cv2 as cv

from matplotlib import pyplot as plt

img = cv.imread('Andrew.jpeg',0)

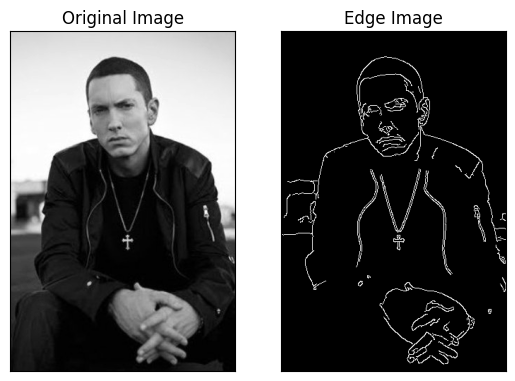
edges = cv.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')

plt.title('Original Image'), plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(edges,cmap = 'gray')

plt.title('Edge Image'), plt.xticks([]), plt.yticks([]



**Reason:** The edge detection using OpenCV is able to find the minute details based on change in intensity of image. The output edges are more accurate and more precise.