CIP ASSIGNMENT-1

1. Search the Web for Cumani colour edge detector and write a report on it. Analyse how it may be better than doing Sobel edge detection three times on an RGB image

CUMANI EDGE DETECTOR :

Cumani edge detector is a colour edge detection technique, this technique deals with vector operations instead of scalar as the image consists of R,G,B componenets with in it. The color information are treated as color vectors and the difference between the norms of the color vectors are calculated. The cumani's technique considers the colour field as a 2d vector field with 3 components i.e., r,g,b colour components

I(x, y) = {I_R(x,y), I_G(x,y), I_B(x,y)}

suggested new norm of directional derivative which is quadratic in nature and known as squared local contrast. in this techniques, By calculating the extreme Eigen values , Eigen vectors and zero crossing points, the edges can be detected in an image.

comparison between sobel and cumani edge detectors:

The Sobel detector uses differentiation operation for computing gradient in horizontal and vertical direction and then convolving the original image with the masks, edges in an image can be detected.these operators are sensitive to noise may not produce accuracy as the magnitude suffers when noise is more. sobel uses a 3*3 mask for edge detection.

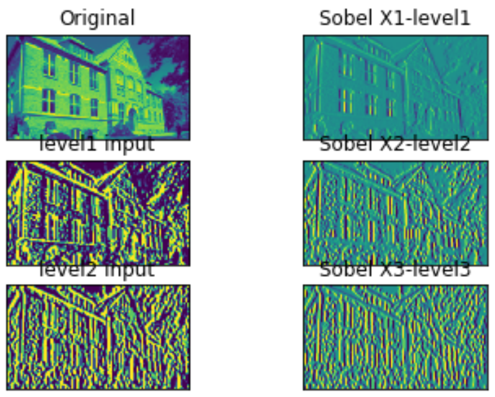
Sobel–Feldman operator is based on convolving the image mask,therefore relatively inexpensive in terms of computations. Another problem is the gradient approximation that it produces is relatively crude, in particular for high-frequency variations in the image,the detector doesnt work much efficiently,Wheras in cumani's it uses combining the component gradients for edge detection as a result it produce smooth edged images.

Let us consider a sobel edge detection on a image 3 times as below it uses function horizantal edge here for now Sobel x,it performs convolution of image with a kernel matrix here the kernel size we considered is 5

cv2.Sobel(frame,cv2.CV_64F,1,0,ksize=5)

cv2.Sobel(original_image,ddepth,xorder,yorder,kernelsize)

In [3]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

#level 1 edge detection
img = cv2.imread('images/edge.jpeg',) # loading image
new_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # converting to gray scale
img = cv2.GaussianBlur(new_img,(3,3),0)# remove noise
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)  # x # performing convolution
cv2.imwrite("images/sobelx1.jpeg",sobelx)

#level 2 edge detection
img1 = cv2.imread('images/sobelx1.jpeg',)
new_image1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img1 = cv2.GaussianBlur(new_image1,(3,3),0) # remove noise
sobelx1 = cv2.Sobel(img1,cv2.CV_64F,1,0,ksize=5)  # x # performing convoluti
on
cv2.imwrite("images/sobelx2.jpeg",sobelx1)

#level 3 edge detection
img2 = cv2.imread('images/sobelx2.jpeg',)
new_image2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
img2 = cv2.GaussianBlur(new_image2,(3,3),0) # remove noise
sobelx2 = cv2.Sobel(img2,cv2.CV_64F,1,0,ksize=5)  # x # performing convoluti
on


plt.subplot(3,2,1),plt.imshow(img)
plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(3,2,2),plt.imshow(sobelx)
plt.title('Sobel X1-level1'), plt.xticks([]), plt.yticks([])

plt.subplot(3,2,3),plt.imshow(img1)
plt.title('level1 input'), plt.xticks([]), plt.yticks([])

plt.subplot(3,2,4),plt.imshow(sobelx1)
plt.title('Sobel X2-level2'), plt.xticks([]), plt.yticks([])

plt.subplot(3,2,5),plt.imshow(img2)
plt.title('level2 input'), plt.xticks([]), plt.yticks([])

plt.subplot(3,2,6),plt.imshow(sobelx2)
plt.title('Sobel X3-level3'), plt.xticks([]), plt.yticks([])

plt.show()
```



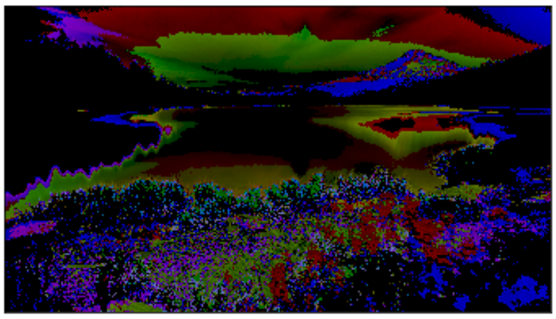
2. Implement colour ranging operation in RGB space on colour images. The inputs for your operation are a colour image and a colour range specification as r_c, r_bw, g_c, g_bw, b_c, b_bw where r_c stands for red colour value, r_bw is the width of the range, i.e., colours between r_c - r_bw and r_c + r_bw must be retained in the image and all other 'r' values should be set to 0. The other parameters are for green and blue colours. You should handle errors when values go out of range.

In [4]:
```
# importing required libraries for performing colouring ranging operations
import numpy as np
import cv2
from matplotlib import pyplot as plt

def lower_bound(colour, bandwidth):
    if (colour-bandwidth < 0):
        return 0
    elif (colour-bandwidth > 255):
        return 255
    else:
        return colour-bandwidth

def upper_bound(colour, bandwidth):
    if (colour+bandwidth > 255):
        return 255
    elif (colour+bandwidth < 0):
        return 0
    else:
        return colour+bandwidth

def perform_ranging( clr_value, low_bound, up_bound):
    if clr_value > low_bound and clr_value < up_bound:
        return clr_value
    else :
        return 0

def color_ranging(r_c, r_bw, g_c, g_bw, b_c, b_bw, rgb_image):
   # GENERATE A NEW ARRAY OF RGB IMAGE SHAPE TO STORE THE RANGED IMAGE VALUE
S INTO IT
    size=rgb_image.shape
    print ("rgb_img shape=",size)

    new_rgb_img=np.zeros(rgb_image.shape).reshape(rgb_image.shape)
    print ("new_rgb_img shape=",new_rgb_img.shape)

    #GET THE UPPER AND LOWER BOUNDARIES FOR ALL THE COLOURS

    r_l_bound=lower_bound(r_c, r_bw)
    g_l_bound=lower_bound(g_c, g_bw)
    b_l_bound=lower_bound(b_c, b_bw)
    r_u_bound=upper_bound(r_c, r_bw)
    g_u_bound=upper_bound(g_c, g_bw)
    b_u_bound=upper_bound(b_c, b_bw)

    vec_perform_ranging=np.vectorize(perform_ranging)

    #PERFORMING COLOUR RANGING ON THE NEW COPIED IMAGE (IMG IS COPIED JUS TO
GET SHAPE OF THE ACTUAL IMAGE)

    new_rgb_img[:, :, 0]=vec_perform_ranging(rgb_image[: , :, 0], r_l_bound,
r_u_bound)
    new_rgb_img[:, :, 1]=vec_perform_ranging(rgb_image[: , :, 1], g_l_bound,
g_u_bound)
    new_rgb_img[:, :, 2]=vec_perform_ranging(rgb_image[: , :, 2], b_l_bound,
b_u_bound)

    #PLOTTING THE IMAGE USING MATPLOTLIB LIBRARY

    plt.imshow(rgb_image),plt.xticks([]),plt.yticks([]),plt.title('ACTUAL')
    plt.show()
    plt.imshow(new_rgb_img),plt.xticks([]),plt.yticks([]),plt.title('RANGED'
)
    plt.show()

def main():
    image=cv2.imread("images/nature.jpg")
    rgb_image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
```

```
rgb_img shape= (1080, 1920, 3)
new_rgb_img shape= (1080, 1920, 3)
```

ACTUAL

RANGED

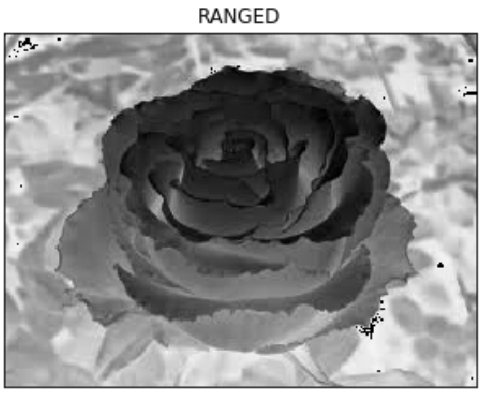


3. Implement colour ranging operation in HSV space on colour images. The inputs for your operation are a colour image and a colour range specification as h_c, h_bw, s_c, v_c where h_c stands for hue value, h_bw is the width of the range, i.e., colours between h_c - h_bw and h_c + h_bw must be retained in the image and all other hue values should be set to 0. The parameter s_c is a saturation threshold. Only those pixels with saturation value above the threshold should be retained. The parameter v_c is a value threshold and only pixels with value greater than v_c should be retained. You should handle errors when values go out of range.

Threshold values for hsv consired are as follows: hue = 0 to 180 saturation = 0 to 255 value = 0 to 255

In [5]:
```
# importing required libraries for performing colouring ranging operations
import numpy as np
import cv2
from matplotlib import pyplot as plt
from matplotlib.colors import rgb_to_hsv,hsv_to_rgb

def lower_bound(colour, bandwidth):
    if (colour-bandwidth < 0):
        return 0
    elif (colour-bandwidth > 180):
        return 180
    else:
        return (colour-bandwidth)%180

def upper_bound(colour, bandwidth):
    if (colour+bandwidth > 180):
        return 180
    elif (colour+bandwidth < 0):
        return 0
    else:
        return (colour+bandwidth) %180

def hue_ranging( hue_value,low_bound,up_bound ):
    if  low_bound<up_bound:
        if hue_value > low_bound and hue_value<up_bound:
            return hue_value
        else :
            return 0
    else:
        if hue_value < low_bound and hue_value>up_bound:
            return hue_value
        else :
            return 0

def hsv_ranging(value,thr):
    if thr>255:
        threshold=255
    elif thr <0:
        threshold=0
    else:
        threshold=thr
    if value > threshold:
        return value
    else:
        return 0

def color_ranging(h_c, h_bw, s_c, v_c,hsv_image,rgb_image):
   # GENERATE A NEW ARRAY OF RGB IMAGE SHAPE TO STORE THE RANGED IMAGE VALUE
S INTO IT
    size=hsv_image.shape
    print ("rgb_img shape=",size)
    new_hsv_img=np.zeros(hsv_image.shape).reshape(hsv_image.shape)
    print ("new_rgb_img shape=",new_hsv_img.shape)

    #GET THE UPPER AND LOWER BOUNDARIES FOR ALL THE COLOURS

    hue_low_bound=lower_bound(h_c, h_bw)
    hue_up_bound=upper_bound(h_c, h_bw)

    vec_hue_ranging=np.vectorize(hue_ranging)
    vec_hsv_ranging=np.vectorize(hsv_ranging)

    #PERFORMING COLOUR RANGING ON THE NEW COPIED IMAGE (IMG IS COPIED JUS TO
GET SHAPE OF THE ACTUAL IMAGE)

    new_hsv_img[:, :, 0]=vec_hue_ranging(hsv_image[: , :, 0],hue_low_bound,h
ue_up_bound)
    new_hsv_img[:, :, 1]=vec_hsv_ranging(hsv_image[: , :, 1],s_c)
```

```
rgb_img shape= (194, 259, 3)
new_rgb_img shape= (194, 259, 3)
```



4. Implement vector median filter and any one of the basic vector edge detectors. Show example images to demonstrate that these are better than the grayscale versions for colour images.

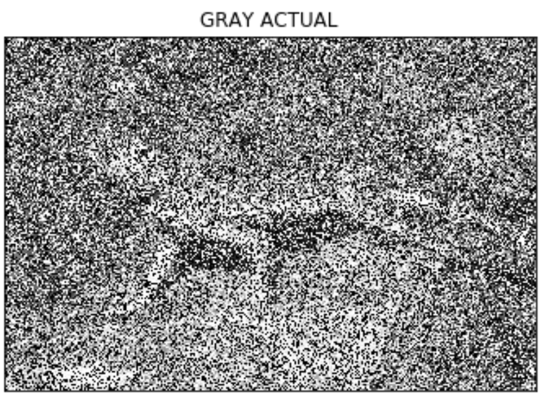
4(A) VECTOR MEDIAN FILTER IMPLEMENTATION

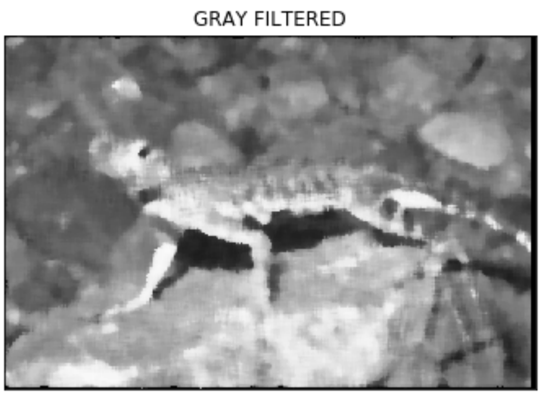
A vector median filter is used to remove the noise in a image here we have taken two images a gray scale and a coloured image to remove the noise in it the colour image noise is removed by converting it into a gray scale image. here size of the filter used is 10 to obtain the denoised image of the lizard (as it is having more noise)

NOTE:

THIS CODE IS OBTAINED FROM A ONLINE WEBSITE AND USED FOR A DETAILED IMPLEMENTATION , DIRECT EXECUTION THROUGH INBUILT LIBRARIES IS ALSO IMPLEMENTED

In [6]:
```
import numpy
from PIL import Image,ImageFilter
import cv2
from matplotlib import pyplot as plt

def median_filter(data, filter_size):
    temp = []
    indexer = filter_size // 2
    denoised_img = []
    denoised_img = numpy.zeros((len(data),len(data[0])))
    for i in range(len(data)):
        for j in range(len(data[0])):
            for z in range(filter_size):
                if i + z - indexer < 0 or i + z - indexer > len(data) - 1:
                    for c in range(filter_size):
                        temp.append(0)
                else:
                    if j + z - indexer < 0 or j + indexer > len(data[0]) - 1
:
                        temp.append(0)
                    else:
                        for k in range(filter_size):
                            temp.append(data[i + z - indexer][j + k - indexe
r])

            temp.sort()
            denoised_img[i][j] = temp[len(temp) // 2]
            temp = []
    return denoised_img


def gray_main(image):
    noisy=cv2.imread(image)
    img = Image.open(image).convert("L")# opening the image into greyscale m
ode using "L" as mode
    arr = numpy.array(img)
    denoised_img = median_filter(arr, 10)
    result = Image.fromarray(denoised_img)
    plt.imshow(result),plt.xticks([]),plt.yticks([]),plt.title('GRAY FILTERE
D')
    plt.show()
img="images/lizard_noisy.png"
noisy=cv2.imread(img)
plt.imshow(noisy),plt.xticks([]),plt.yticks([]),plt.title('GRAY ACTUAL')
plt.show()
gray_main(img)
```

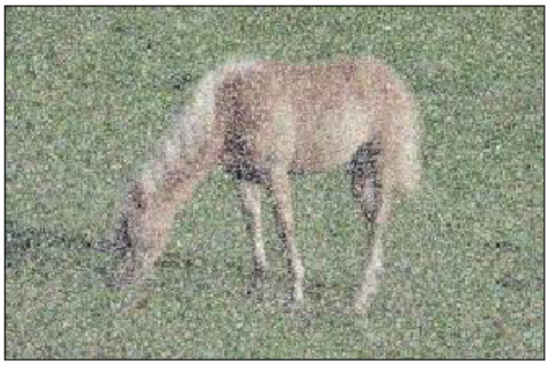




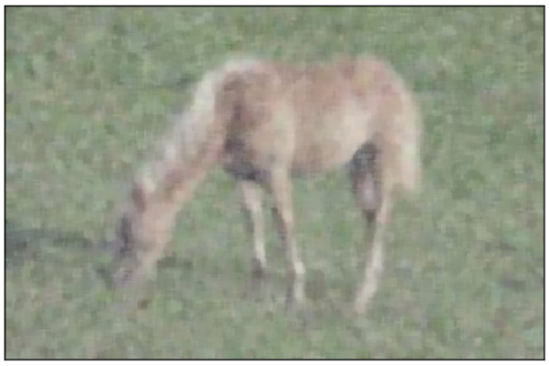
```
In [7]: def coloured_filter(image):
            img = Image.open(image)
            new_img = img.filter(ImageFilter.MedianFilter(5))
            plt.imshow(img),plt.xticks([]),plt.yticks([]),plt.title('coloured ACTUAL
        ')
            plt.show()
            plt.imshow(new_img),plt.xticks([]),plt.yticks([]),plt.title('colour FILT
        ERED')
            plt.show()
```

In [8]:
```
image1="images/noisy2.jpeg"
coloured_filter(image1)
gray_main(image1)
```

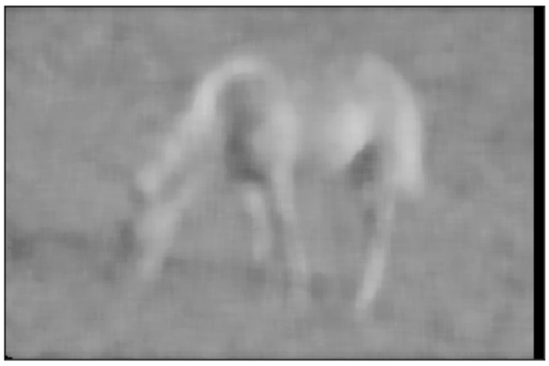
coloured ACTUAL



colour FILTERED

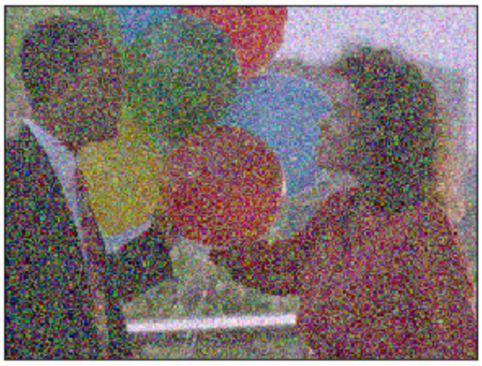


GRAY FILTERED

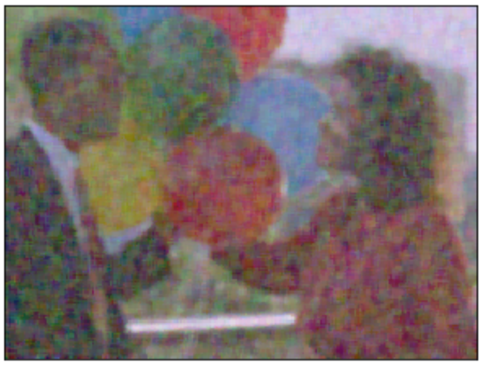


In [9]:
```
image2="images/noise.jpeg"
coloured_filter(image2)
gray_main(image2)
```

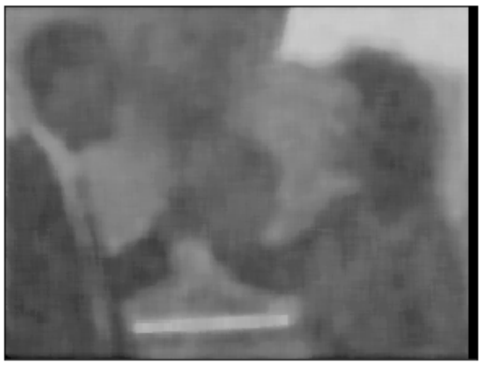
coloured ACTUAL



colour FILTERED



GRAY FILTERED



4(B) LAPLACIAN AND SOBEL EDGE DETECTORS

A sobel edge detection on a image performs convolution of image with a kernel matrix here the kernel size taken is 5 cv2.Sobel(frame,cv2.CV_64F,1,0,ksize=5) cv2.Sobel(frame,cv2.CV_64F,0,1,ksize=5)

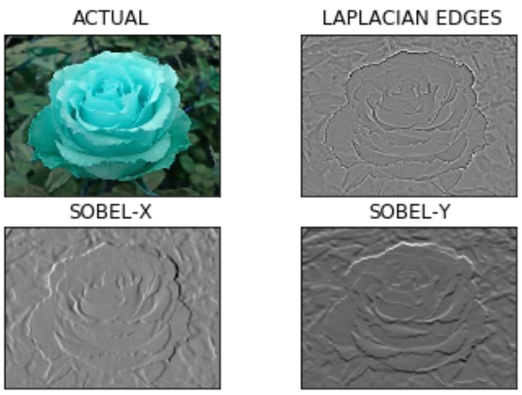
cv2.Sobel(original_image,ddepth,xorder,yorder,kernelsize)

A laplacian edge detector is little different from sobel,in sobel we can detect horizantal and vertical individual and use a kernal with variable size,but in laplacian only a single kernel is used and second order derivative is calculated in single pass.

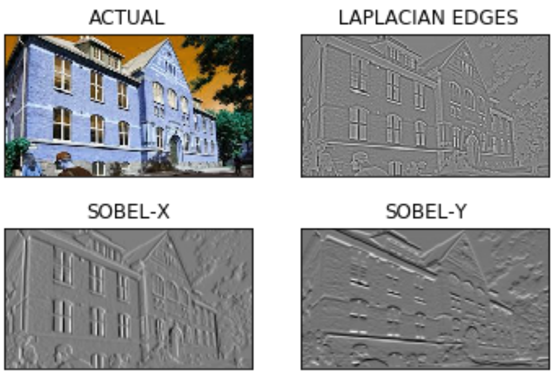
cv2.Laplacian(src, ddepth, other_options...)

In [10]:
```
import cv2
import numpy as np
def edge_detectors (image):
    img = cv2.imread(image,) # loading image
    plt.subplot(2,2,1),plt.imshow(img),plt.xticks([]),plt.yticks([]),plt.tit
le('ACTUAL')
    new_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.GaussianBlur(new_img,(3,3),0)# remove noise
    sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=3) #HORIZANTAL EDGE DETECTIO
N
    sobelY = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=3) #VERTICAL EDGE DETECTION
    laplacian = cv2.Laplacian(img,cv2.CV_64F)  # LAPLACIAN EDGE DETECTION
    plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray'),plt.xticks([]),pl
t.yticks([]),plt.title(' LAPLACIAN EDGES')
    plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray'),plt.title('SOBEL-X')
, plt.xticks([]), plt.yticks([])
    plt.subplot(2,2,4),plt.imshow(sobelY,cmap = 'gray'),plt.title('SOBEL-Y')
, plt.xticks([]), plt.yticks([])
```

In [11]:
```
edge_detectors("images/vec2_rose.jpeg")
```



In [12]:
```
edge_detectors("images/edge.jpeg")
```



In [13]:
```
edge_detectors("images/chair.jpeg")
```