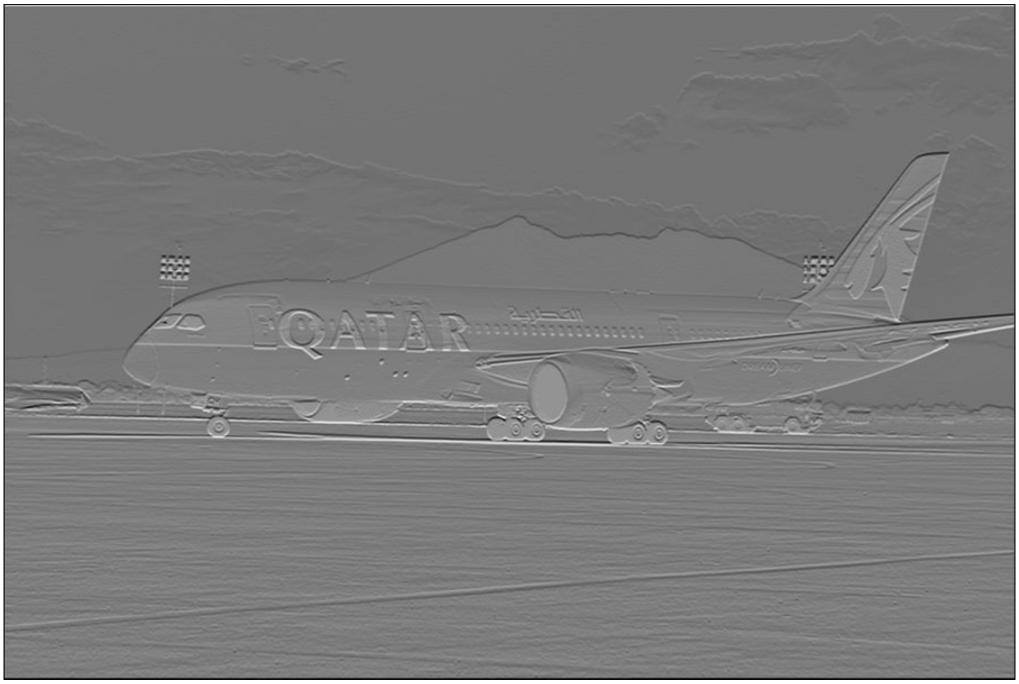
Project 1

CSE573: Computer Vision

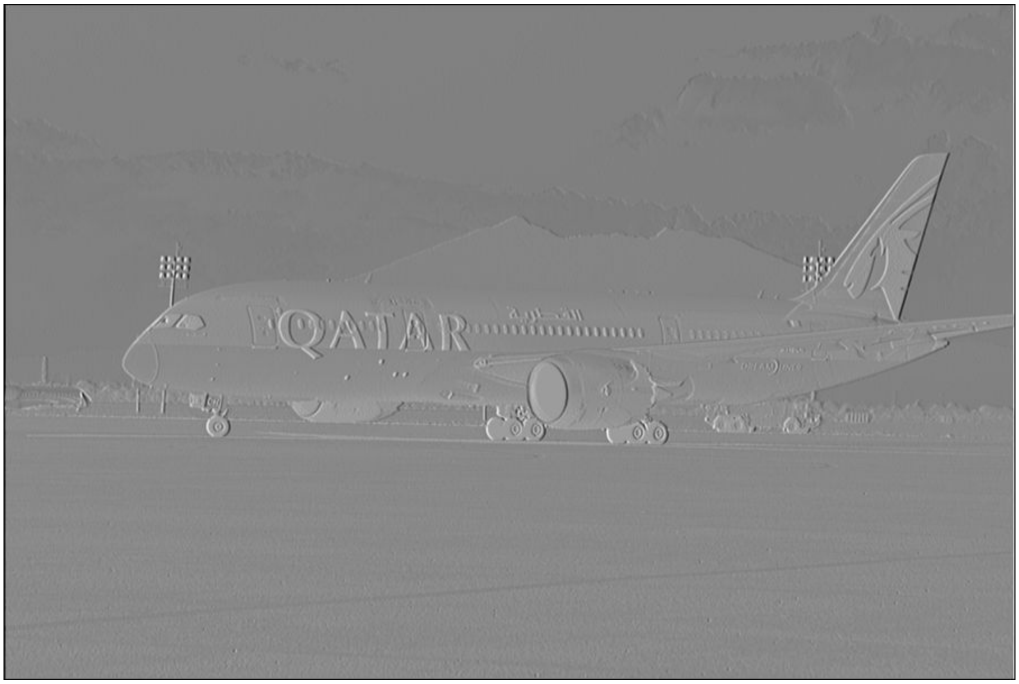
& Image Processing

NAME :- PRANAV VIJ
UB NUMBER :- 50290588

EDGE DETECTION



SOBEL X



SOBEL Y

CODE FOR EDGE DETECTION

1. Load Image and convert it from RGB to GrayScale Image. The image is then converted to an 2-D array.

```
img = cv2.imread("task1.png", 0)
a = np.asarray(img).tolist()
```

2. Add Padding to the image so that it could be multiplied with the sobel_y operator by iterating through each element of the Image.

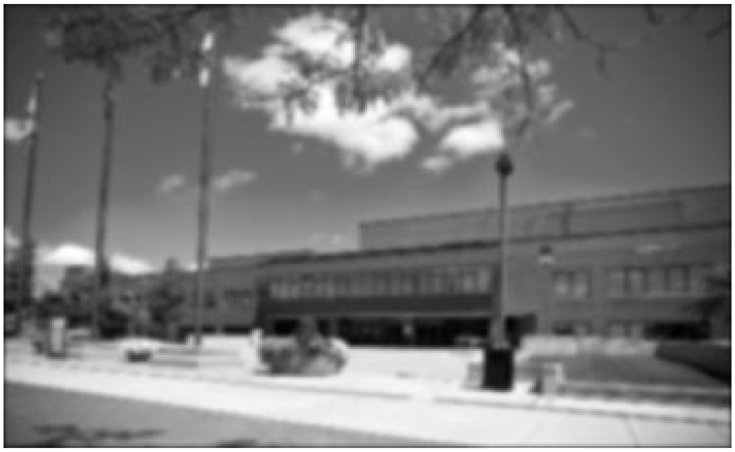
```
SOBEL_Y = [
    [-1,0, 1],
    [-2,0, 2],
    [-1,0, 1]
]

m = len(a) ## rows
n = len(a[0]) ## columns
X = [[0]*(n+2)]
for i in range(0, m):
    X.append([0] + a[i] + [0])
X.append([0]*(n+2))
Y = copy.deepcopy(X)
max_x = 0
min_x = X[1][1]

for i in range(1, m + 1):
    for j in range(1, n + 1):
        for k in range(-1, 2):
            for l in range(-1, 2):
                Y[i][j] += (SOBEL_Y[k + 1][l + 1] * X[i + k][j + l])
    max_x = max(max(Y[i]), max_x)
    min_x = min(min(Y[i]), min_x)
```

3. Normalizing the image so that could be converted to a 0 - 255 or 0 - 1 format scale. As multiplying with sobel_y operator would cause the values to reach out of the above range.

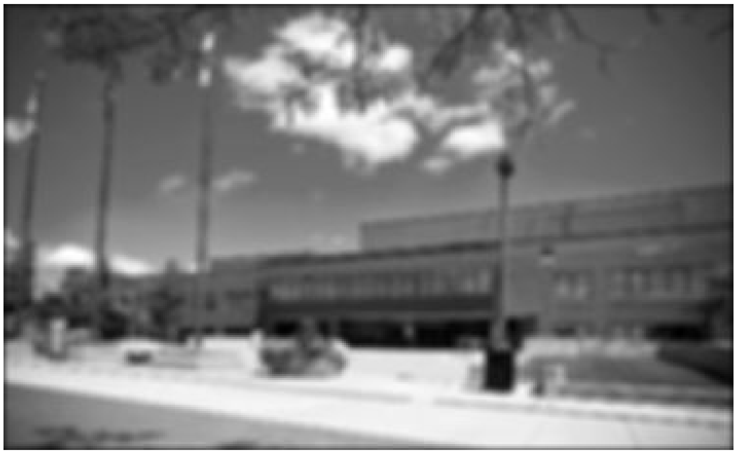
```
for i in range(1, m + 1):
    for j in range(1, n + 1):
        Y[i][j] = int((max_x - Y[i][j])/(max_x - min_x)*255)

img2 = np.array(Y)
cv2.imwrite('SOBEL_Y.png',img2)
cv2.imshow('image', img2)
cv2.waitKey(0)
```

KEYPOINT DETECTION

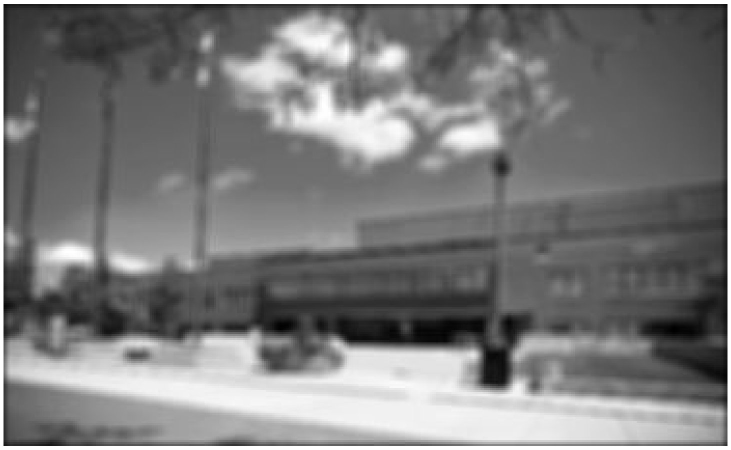
OCTAVE 2



σ = 1.414



σ = 2



σ = 2.828

σ = 4

σ = 5.656

The Image size for Octave 3 is 375 * 229 px which has been shown above in a tabular format for each sigma value corresponding to the 3rd Octave.

OCTAVE 3

σ = 2.828

σ = 4

σ = 5.65

σ = 8

σ = 11.31

The Image size for Octave 3 is 187 * 114 px which has been shown above in a tabular format for each sigma value corresponding to the 3rd Octave.

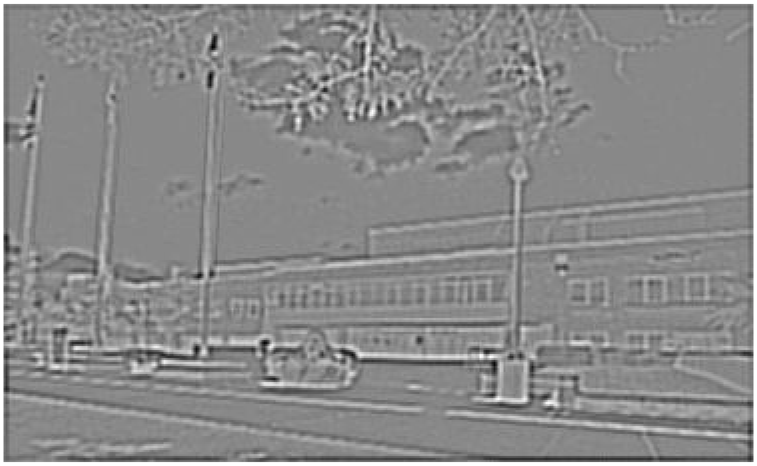
Convolving Gaussian kernel and Image Matrix we get the above result for different values of Sigma (σ) for octave 2 as shown in Tab 1. Using the below function, We have calculated the Octave matrix and applied the convolution.

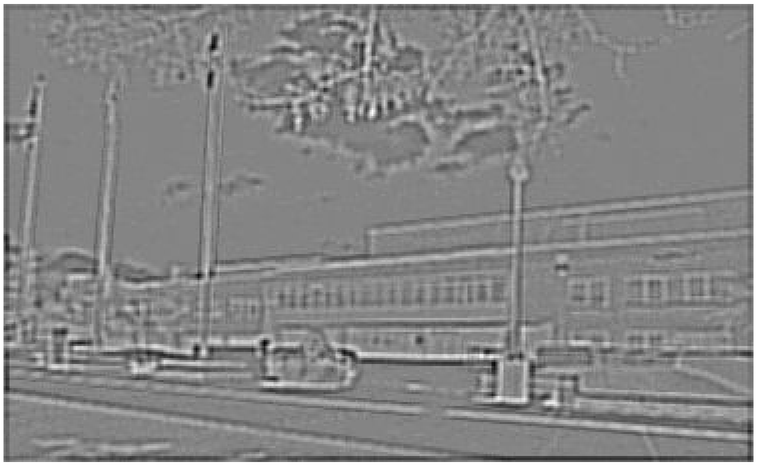
```
def create_octave_matrix():
      a = math.pow(2, -0.5)
      mult = math.pow(2, 0.5)
      OCTAVE_MATRIX = [[0]*5]
      OCTAVE_MATRIX[0][0] = a
      for i in range(1, 4):
            OCTAVE_MATRIX.append([0]*5)
            OCTAVE_MATRIX[i][0] = OCTAVE_MATRIX[i - 1][0] * 2
      for i in range(0, 4):
            for j in range(1, 5):
                  OCTAVE_MATRIX[i][j] = OCTAVE_MATRIX[i][j - 1] * mult
      return OCTAVE_MATRIX

def get_gausian_xy(x, y, sigma):
      sigma_2 = 2.0*sigma*sigma
      return (math.exp(-(x*x + y*y)/sigma_2))
```

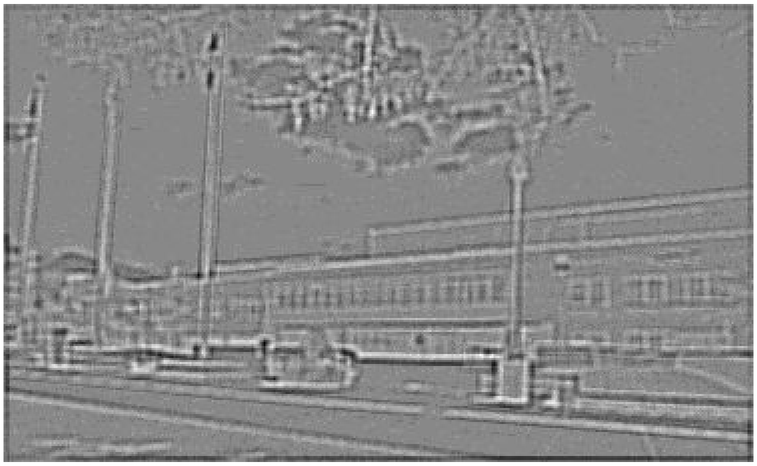
```
def gausian_matrix(n,sigma):
      A = [[0] * n]
      s = 0
      for i in range(0, n - 1):A.append([0] * n)
      for i in range(0, n):
            for j in range(0, n):
                  g = get_gausian_xy(i - (n - 1)/2, j - (n - 1)/2, sigma)
                  s += g
                  A[i][j] = g
      for i in range(0, n):
            for j in range(0, n):
                  A[i][j] /= s
      A = np.transpose(A)
      A = np.asarray(A).tolist()
      return A

def matrix_mult_rect_scan(A, B, n):# WHERE A IS BIGGER THAN B
      padding = int((n - 1)/2)
      row_a_size = len(A)
      col_a_size = len(A[0])
      MATRIX_SCANNED = []
      for i in range(0,row_a_size):
            MATRIX_SCANNED.append([0]*col_a_size)
      for i in range(padding, row_a_size - padding):
            for j in range(padding, col_a_size - padding):
                  for k in range(0, n):
                        for l in range(0, n):
                              MATRIX_SCANNED[i][j] += B[k][l]*A[i - padding +
k][j - padding + l]
      return MATRIX_SCANNED
```

Difference of Gaussians

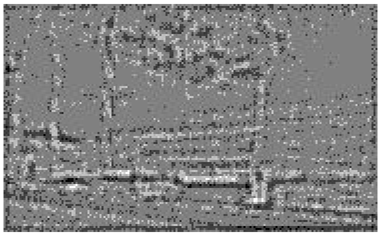
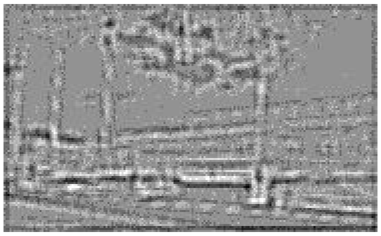
The difference of gaussian for octave 2



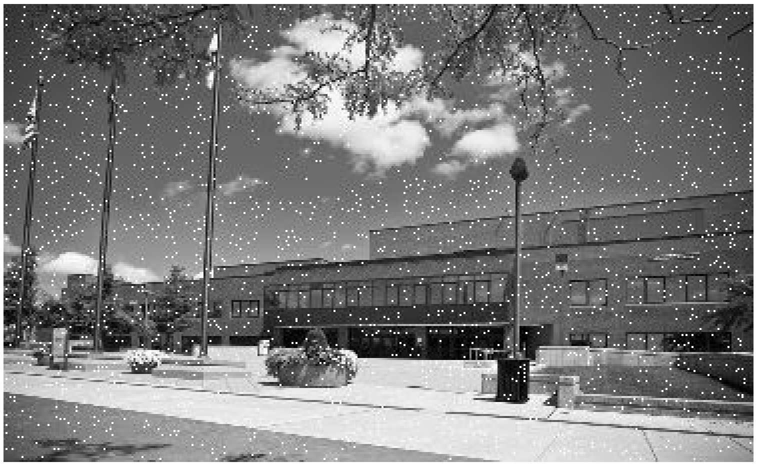




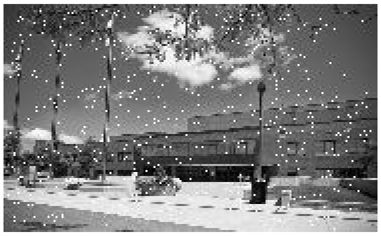
The difference of gaussian for octave 3

Key points of Octave 2



Key points of Octave 3



Below is the code for calculating the key points by using the difference of gaussian and calculating the local minima and local maxima for 26 points

Coordinates of the five left-most detected keypoints

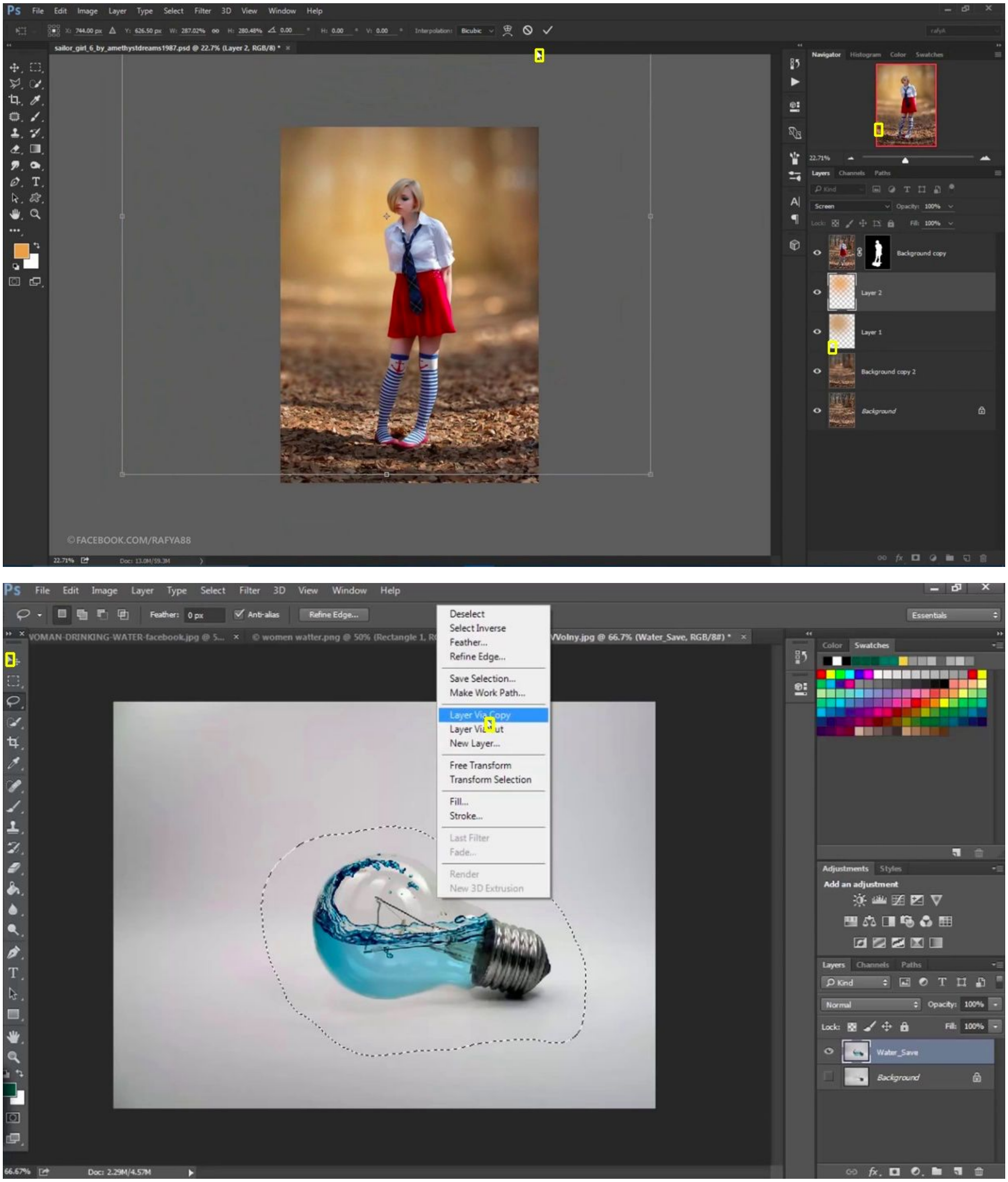
Keypoint 4 = (22,1), (26, 4), (45,5), (51,4), (29,5)
Keypoint 3 = (45, 2), (136, 3), (86, 5), (53, 4), (39, 4)
Keypoint 2 = (10, 1), (10, 1), (34, 1), (61, 1), (66, 1)
Keypoint 1 = (1, 4), (331, 1), (366, 1), (324, 1), (351, 1)

CODE FOR KEYPOINT DETECTION

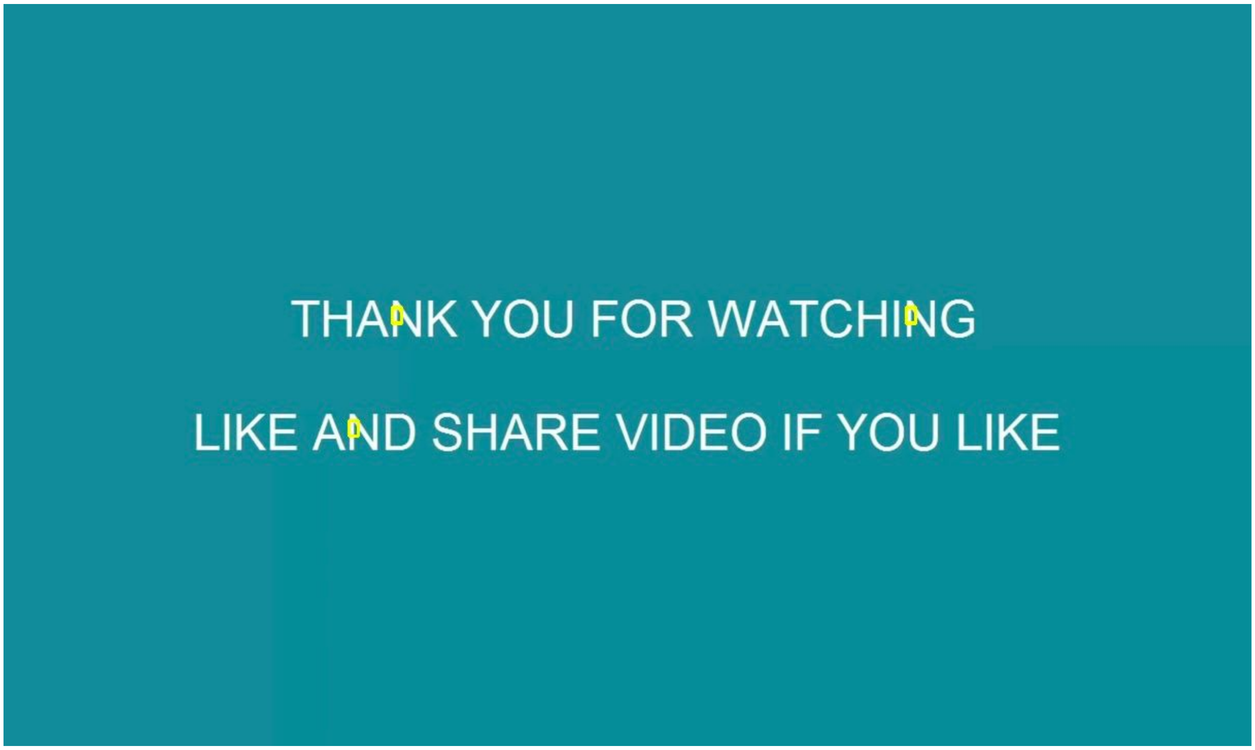
```
def subtract_matrix(A, B):
      row_a_size = len(A)
      col_a_size = len(A[0])
      row_b_size = len(B)
      col_b_size = len(B[0])
      if col_b_size != col_a_size or row_a_size != row_b_size:
            print("error found")
            return
      for i in range(0, row_b_size):
            for j in range(0, col_b_size):
                  A[i][j] = A[i][j] - B[i][j]
      return A

def show_image(matrix, i, j):
      img2 = np.array(matrix)
      cv2.imwrite('OCTAVE_MATRIX' + str(i) + '_' + str(j) + '.png', img2)
      #cv2.namedWindow('image', cv2.WINDOW_NORMAL)
      #cv2.imshow('image', img2)
      #cv2.waitKey(0)

def gausian_matrix_sigma(sigma, MATRIX, n):
      row_len = len(MATRIX)
      col_len = len(MATRIX[0])
      GAUSIAN_MATRIX = gausian_matrix(n, sigma)
      MATRIX  = add_padding(MATRIX, n, row_len, col_len)
      MATRIX_AFTER_MULT = matrix_mult_rect_scan(MATRIX, GAUSIAN_MATRIX, n)
      return MATRIX_AFTER_MULT

n = 7 # CONSIDERING n to b odd
row_len = len(a)
col_len = len(a[0])
OCTAVE_MATRIX = create_octave_matrix()
OCTAVES_VAL = []

for i in range(0, len(OCTAVE_MATRIX)):
      octave = []
      for j in range(0, len(OCTAVE_MATRIX[0]) - 1):
            matrix = gausian_matrix_sigma(OCTAVE_MATRIX[i][j], a, n)
            matrix_1 = gausian_matrix_sigma(OCTAVE_MATRIX[i][j+1], a, n)
```

```
        MATRIX_AFTER_MULT_1 = subtract_matrix(matrix_1, matrix)
        octave.append(MATRIX_AFTER_MULT_1)
    OCTAVES_VAL.append(octave)

index = 0
for x in range(1,3):
    for k in range(0, len(OCTAVES_VAL)):
        octave = k % 4
        if octave == 0:
            index += 1
        keypoint = 0
        img1 = OCTAVES_VAL[k][x - 1]
        img2 = OCTAVES_VAL[k][x]
        img3 = OCTAVES_VAL[k][x + 1]

        for i in range(1, len(img2) - 1):
            for j in range(1, len(img2[0]) - 1):
                max_val = img2[i - 1][j - 1]
                min_val = img2[i - 1][j - 1]
                for l in range(-1 , 2):
                    for m in range(-1, 2):
                        max_val = max(max_val, img1[i + l][j +
m], img3[i + l][j + m])
                        min_val = min(min_val, img1[i + l][j +
m], img3[i + l][j + m])
                        if l != 0 and m != 0:
                            min_val = min(min_val, img2[i +
l][j + m])
                            max_val = max(max_val, img2[i +
l][j + m])

                if min_val > img2[i][j] or max_val < img2[i][j]:
                    keypoint += 1

        print(keypoint)
        print(index, octave)
```

CURSOR DETECTION

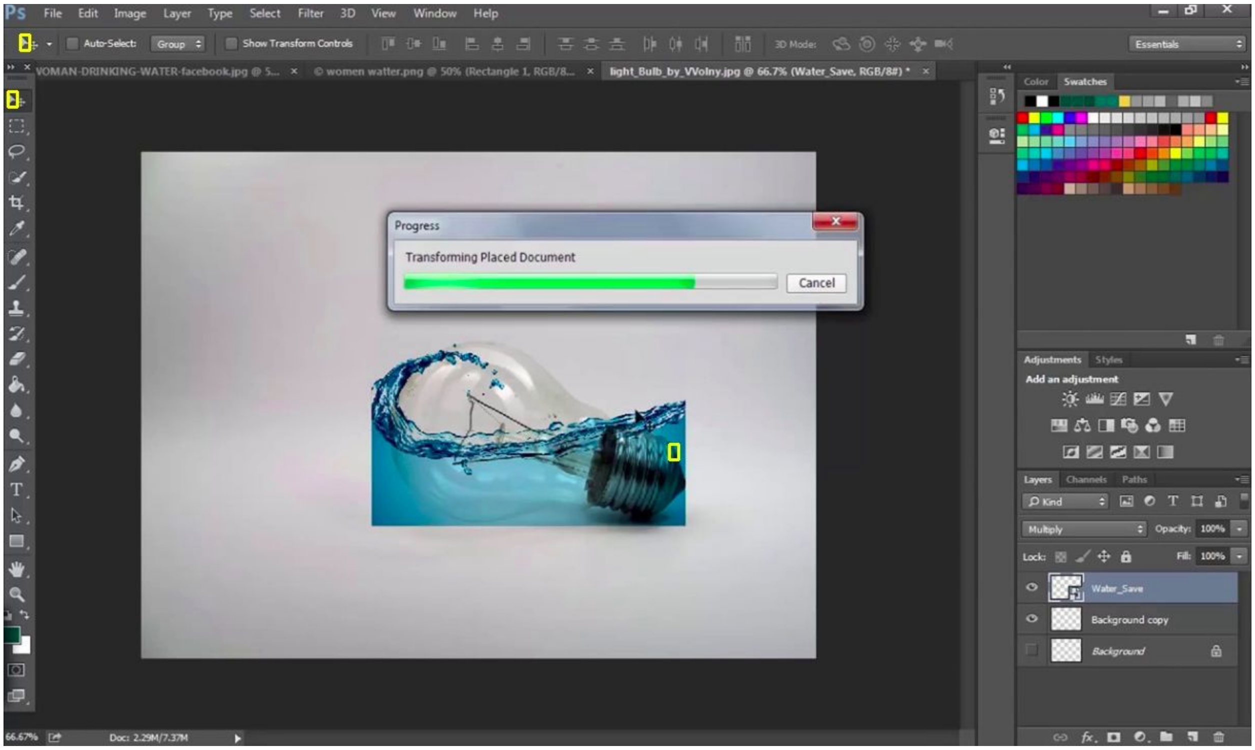


Positive Image



THANK YOU FOR WATCHING

LIKE AND SHARE VIDEO IF YOU LIKE



Progress

Transforming Placed Document

Cancel

Negative Image

CODE FOR CURSOR DETECTION

1. Resizing function for different template size and matching image.

```
def resize(img, scale_percent):
    width = int(img.shape[1] * scale_percent / 100)
    height = int(img.shape[0] * scale_percent / 100)
    dim = (width, height)
    resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
    return resized
```

2. Converting the image and template to LaPlacian Image after using gaussian blur (3* 3) on the image.

```
def get_laplacian(img):
    return cv2.Laplacian(img, cv2.CV_8U,ksize = 3)

def get_gausian(img):
    return cv2.GaussianBlur(img,(3,3),0)
```

3. In order to more refine the template matching added canny’s on the sub sampled images and applying template matching for the equally sized sub sampled template and Image

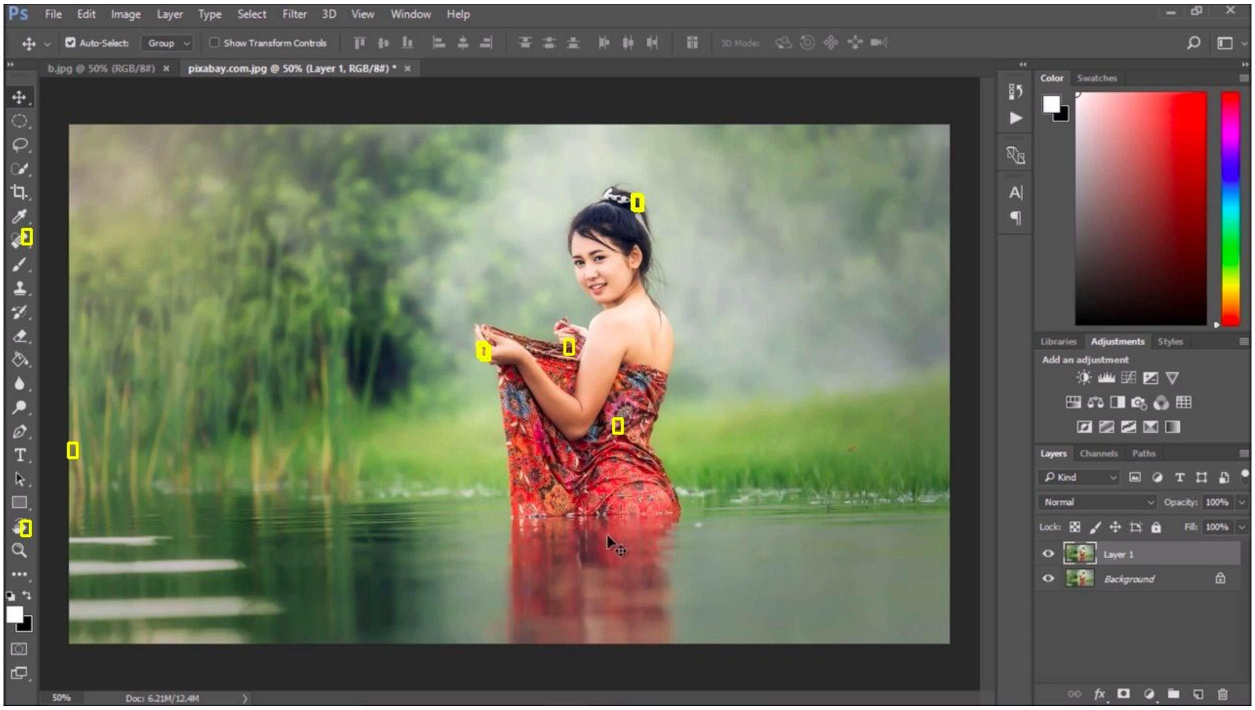
```
def get_canny(img,start,end):
    return cv2.Canny(img, start, end)

def show(img):
    cv2.imshow('image', img)
    cv2.waitKey(0)

def image_without_bg(img):
    height, width = img.shape[:2]
    mask = np.zeros(img.shape[:2],np.uint8)
    bgdModel = np.zeros((1,65),np.float64)
    fgdModel = np.zeros((1,65),np.float64)
    rect = (0,2,width,height)
    cv2.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_RECT)
    mask = np.where((mask==2)|(mask==0),0,1).astype('uint8')
    img1 = img*mask[:,:,np.newaxis]
    background = img - img1
    #Change all pixels in the background that are not black to white
```

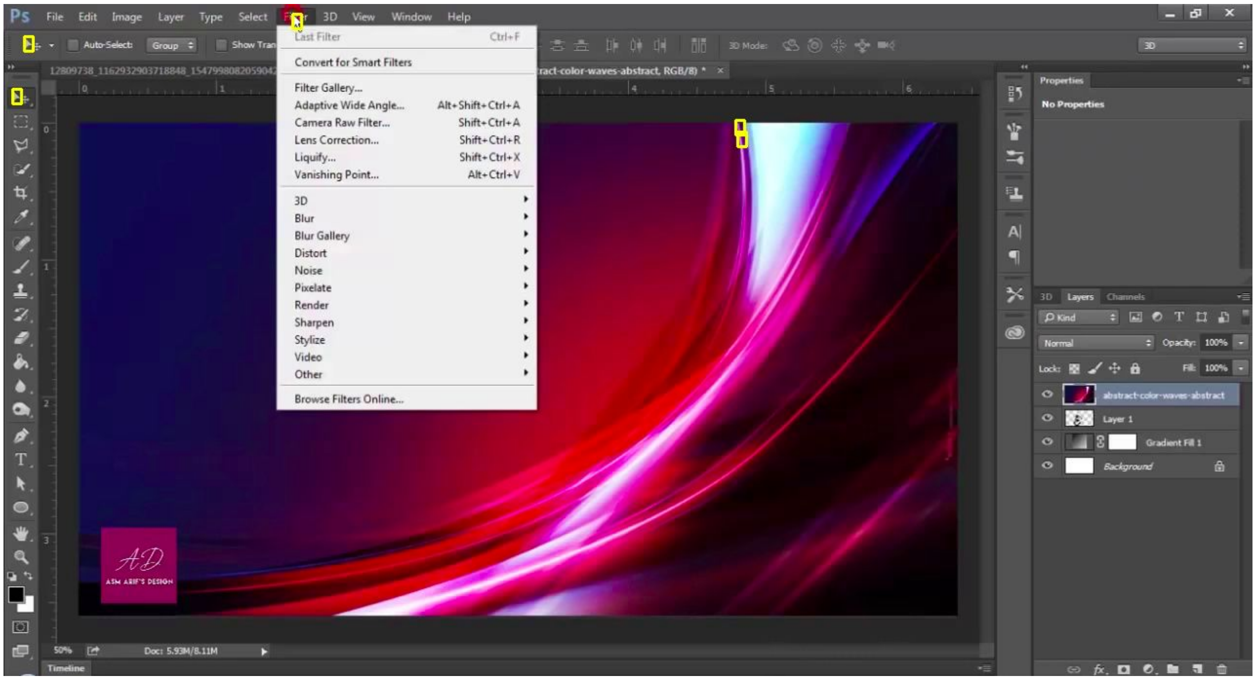
```
      background[np.where((background > [0,0,0]).all(axis = 2))] =
[255,255,255]
      #Add the background and the image
      final = background + img1
      return final

def is_canny_match(img, temp):
    mean_temp_int = np.mean(temp)
    temp_canny = cv2.Canny(temp, int(0.75*mean_temp_int),
int(1.75*mean_temp_int))
    mean_img_int = np.mean(img)
    img_canny = cv2.Canny(img, int(0.75*mean_temp_int),
int(1.75*mean_temp_int))

    try:
        res = cv2.matchTemplate(img_canny, temp_canny, cv2.TM_CCORR_NORMED)
        loc = np.where(res > 0.5)
        return len(loc[0]) > 0
    except:
        return False

def is_diff_optimized(img_patch, template_new):
      difference = cv2.subtract(get_gausian(template_new),
get_gausian(img_patch))
      diff = cv2.countNonZero(difference)
      if diff <= 100:
            return True
      return False

PATH_TEMPLATE = "small.png"
```

4. Traversing through all the images of pos and negative in order to find the template in the positive and negative Images. Two results have been shared above for negative and positive images.

```
for i in range(1, 14):
      PATH_IMG = "pos_" + str(i) +".jpg"
      img = cv2.imread(PATH_IMG)
      img_show = img.copy()
      img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY);
      img_gray = get_gausian(img_gray)
      img_gray = get_laplacian(img_gray)
      template = cv2.imread(PATH_TEMPLATE)
      template = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY);
      template = get_laplacian(template)
      scale_percent = 50

      for scale in range(scale_percent, scale_percent + 1):
            img_copy =  img_gray.copy()
            template_new = resize(template, scale)
            res = cv2.matchTemplate(img_copy, template_new, 
cv2.TM_CCORR_NORMED)
            w, h = template_new.shape[::-1]
            threshold = 0.633
            loc = np.where( res >= threshold)
            l_len = len(loc[0])
            if l_len > 0:
                  print(l_len)
            for pt in zip(*loc[::-1]):
                  img_patch = img_copy[pt[1]:pt[1]+(h), pt[0]:pt[0]+(w)]
                  if(is_canny_match(img_patch, template_new)):
                        cv2.rectangle(img_show, pt, (pt[0] + w, pt[1] + h), 
(0,255,255), 2)

            cv2.imshow('image', img_show)
            cv2.imwrite("pos_" + str(i) +".jpg", img_show)
            cv2.waitKey(0)
```

BONUS QUESTION


File Edit Image Layer Type Select Filter 3D View Window Help
b.jpg @ 50% (RGB/8#)
pixabay.com.jpg @ 50% (Layer 1, RGB/8#) *
Auto-Select: Group
Show Transform Controls
Color Swatches
Libraries Adjustments Styles
Add an adjustment
Layers Channels Paths
Normal
Opacity: 100%
Fill: 100%
Layer 1
Background
50%
Doc: 6.21M/12.4M


File Edit Image Layer Type Select Filter 3D View Window Help
Last Filter Ctrl+F
Convert for Smart Filters
Filter Gallery...
Adaptive Wide Angle... Alt+Shift+Ctrl+A
Camera Raw Filter... Shift+Ctrl+A
Lens Correction... Shift+Ctrl+R
Liquify... Shift+Ctrl+X
Vanishing Point... Alt+Ctrl+V
3D
Blur
Blur Gallery
Distort
Noise
Pixelate
Render
Sharpen
Stylize
Video
Other
Browse Filters Online...
Properties
No Properties
3D Layers Channels
Normal
Opacity: 100%
Fill: 100%
abstract-color-waves-abstract
Layer 1
Gradient Fill 1
Background
AD
ASM ARIF'S DESIGN
50%
Doc: 5.93M/8.11M
Timeline