CSE 565 - Project 1

Amlan Gupta (#50288686)

October 2018

1 Question

Write programs to detect edges in Fig. 1 (along both x and y directions) using Sobel operator. In your report, please include two resulting images, one showing edges along x direction and the other showing edges along y direction

The program for edge detection along x direction and y direction is given below. After applying the soble operator the output has been normalized to make the edges more distinguishable.

```
import cv2
import numpy as np

def print_image(img, image_name):
        cv2.namedWindow(image_name, cv2.WINDOW_NORMAL)
        cv2.imshow(image_name, img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

def convolve_img(img, kernel,kernel_radius):


        height, width = img.shape
        output_image = [[0 for col in range(width)] for row in range(height)]


        for i in range(kernel_radius, height-kernel_radius):
                for j in range(kernel_radius, width-kernel_radius):

                        loop_end = (kernel_radius*2)+1

                        sum = 0
                        for x in range(0,loop_end):
                                for y in range(0,loop_end):
                                        sum += kernel[x][y] * img[i-kernel_radius+x][j-kernel_radius+y]

                        output_image[i][j] = sum


        return np.asarray(output_image)


def edge_detection_x(img):

        x_kernel = [[-1, 0, 1], [-2, 0, 2], [-1, 0 , 1]]

        edge_x_img = convolve_img(img,x_kernel,1)

        h,w = edge_x_img.shape

        max_val = 0
        for i in range(0,h):
                for j in range(1,w):
```

```
                        edge_x_img[i][j] = abs(edge_x_img[i][j])
                        max_val = max(max_val,edge_x_img[i][j])

        pos_edge_x = [[0.0 for col in range(w)] for row in range(h)]

        for i in range(0,h):
                for j in range(1,w):
                        pos_edge_x[i][j] = edge_x_img[i][j]/max_val


        print_image(np.asarray(pos_edge_x),'x_edge_detection_normalized')


def edge_detection_y(img):

        y_kernel = [[-1, -2, -1], [0, 0, 0], [1, 2 , 1]]

        edge_y_img = convolve_img(img,y_kernel,1)

        h,w = edge_y_img.shape

        max_val = 0
        for i in range(0,h):
                for j in range(1,w):
                        edge_y_img[i][j] = abs(edge_y_img[i][j])
                        max_val = max(max_val,edge_y_img[i][j])

        pos_edge_y = [[0.0 for col in range(w)] for row in range(h)]

        for i in range(0,h):
                for j in range(1,w):
                        pos_edge_y[i][j] = edge_y_img[i][j]/max_val


        print_image(np.asarray(pos_edge_y),'y_edge_detectioin_normalized')


def main():

        task_1_img = cv2.imread("task1.png", 0)
        edge_detection_x(task_1_img)
        edge_detection_y(task_1_img)


main()
```

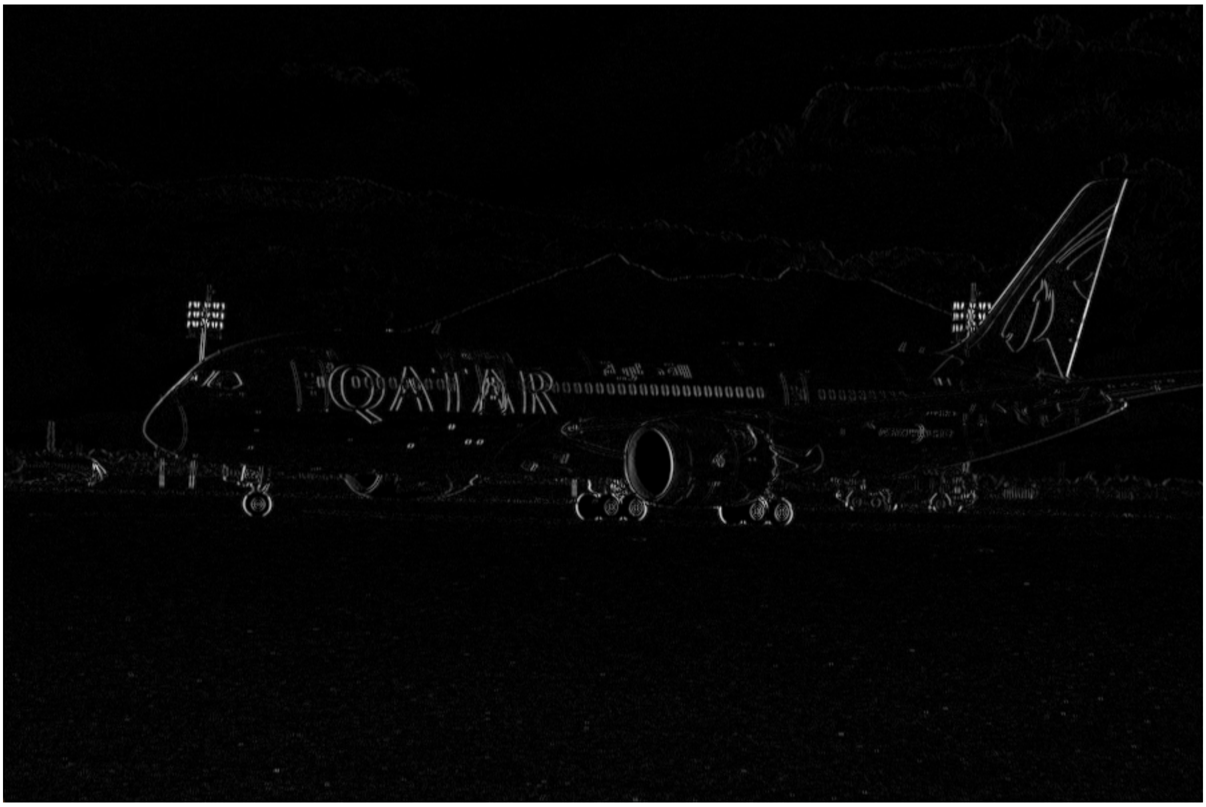


Figure 1: Detected edge along x direction

Figure 2: Detected edge along y direction

2 Question

Write programs to detect keypoints in an image according to the following steps, which are also the first three steps of Scale-Invariant Feature Transform (SIFT).

```python
import cv2
import numpy as np
from math import sqrt
from math import exp



def print_image(img, image_name):
        cv2.namedWindow(image_name, cv2.WINDOW_NORMAL)
        cv2.imshow(image_name, img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()


def write_image(img, image_name):
        cv2.imwrite(image_name + '.png',img)


def apply_blur(img, kernel,kernel_radius):

        #applying 7x7 gausian blur
        height, width = img.shape
        output_image = [[0 for col in range(width)] for row in range(height)]

        for i in range(kernel_radius, height-kernel_radius):
                for j in range(kernel_radius, width-kernel_radius):

                        loop_end = (kernel_radius*2)+1

                        sum = 0
                        for x in range(0,loop_end):
                                for y in range(0,loop_end):
                                        sum += kernel[x][y] * img[i-kernel_radius+x][j-kernel_radius+y]

                        output_image[i][j] = sum


        return np.asarray(output_image)


def gaussian(x, mu, sigma):
  return exp( -(((x-mu)/(sigma))**2)/2.0 )


def get_gaussian_kernel(sigma):

        kernel_radius = 3

        hkernel = [gaussian(x, kernel_radius, sigma) for x in range(2*kernel_radius+1)]
        vkernel = [x for x in hkernel]
        kernel2d = [[xh*xv for xh in hkernel] for xv in vkernel]

        kernelsum = sum([sum(row) for row in kernel2d])
        kernel2d = [[x/kernelsum for x in row] for row in kernel2d]

        return kernel2d


def resize_image_to_half(img):

        height, width = img.shape

        output_image = [[0 for col in range(int(width/2))] for row in range(int(height/2))]
```

```python

	i_op = 0
	for i in range(0,height):
		j_op = 0

		if i%2 == 0:
				continue

		for j in range(0, width):
			if j%2 == 0:
				continue

			output_image[i_op][j_op] = img[i][j]
			j_op+=1

		i_op+=1


	return np.asarray(output_image)



def generate_gaussian_blur_for_an_image(img, octav_id, sigma_row):

	for i in range(len(sigma_row)):

		gussian_blurred_img = apply_blur(img, get_gaussian_kernel(sigma_row[i]), 3)

		write_image(gussian_blurred_img,'gb_img_'+ octav_id +'_'+str(i))


def generate_octavs(image_1,sigma_table):

	# octav 1: original image
	write_image(image_1,'octav_1_original')
	generate_gaussian_blur_for_an_image(image_1,'octav_1', sigma_table[0])

	# octav 2: original image/2
	image_2 = resize_image_to_half(image_1)
	write_image(image_2,'octav_2_original')
	generate_gaussian_blur_for_an_image(image_2,'octav_2', sigma_table[1])

	# octav 3: original image/4
	image_3 = resize_image_to_half(image_2)
	write_image(image_3,'octav_3_original')
	generate_gaussian_blur_for_an_image(image_3,'octav_3', sigma_table[2])

	# octav 4: original image/8
	image_4 = resize_image_to_half(image_3)
	write_image(image_4,'octav_4_original')
	generate_gaussian_blur_for_an_image(image_4,'octav_4', sigma_table[3])


def compute_DoG(list):

	for j in range(1,5):
		for i in range(0,4):


			img_lower_blur = cv2.imread("gb_img_octav_" + str(j) + "_" + str(i) + ".png", 0)
			img_higher_blur = cv2.imread("gb_img_octav_" + str(j) + "_" + str(i+1) + ".png", 0)

			height, width = img_lower_blur.shape
```

```
                        difference = [[0 for col in range(width)] for row in range(height)]


                        for h in range(0,height):
                                for w in range(0, width):
                                        difference[h][w] = int(img_higher_blur[h][w]) - int(img_lower_blur[h][w])

                        difference = np.asarray(difference)
                        write_image(difference,'dog_octav_'+ str(j)+'_'+ str(i))

                        list.append(difference)

        return list



def find_marked_maxima_minima(dog_top, dog_middle, dog_bottom, scale_multiplier, original_img):

        height, width = dog_middle.shape

        # traversing image
        for h in range(1,height-1):
                for w in range(1, width-1):

                        #threshold
                        if dog_middle[h][w]<2:
                                continue


                        # traversing and comparing 26 neighbours
                        is_maxima = True

                        for i in range(h-1,h+2):
                                for j in range(w-1,w+2):
                                        if (dog_middle[h][w] < dog_middle[i][j]) or
                                        (dog_middle[h][w] < dog_top[i][j]) or
                                        (dog_middle[h][w] < dog_bottom[i][j]):
                                                is_maxima = False
                                                break

                                if not is_maxima:
                                        break

                        if is_maxima:
                                original_img[h*scale_multiplier][w*scale_multiplier] = 255
                        else:

                                is_minima = False

                                for i in range(h-1,h+2):
                                        for j in range(w-1,w+2):
                                                if (dog_middle[h][w] > dog_middle[i][j]) or
                                                (dog_middle[h][w] > dog_top[i][j]) or
                                                (dog_middle[h][w] > dog_bottom[i][j]):
                                                        is_minima = False
                                                        break

                                        if not is_minima:
                                                break
                                if is_minima:
                                        original_img[h*scale_multiplier][w*scale_multiplier] = 255

```

```
        return original_img

def find_keypoints(original_img, list_of_dog):

        find_marked_maxima_minima(list_of_dog[0],list_of_dog[1],list_of_dog[2], 1, original_img)
        find_marked_maxima_minima(list_of_dog[1],list_of_dog[2],list_of_dog[3], 1, original_img)

        find_marked_maxima_minima(list_of_dog[4],list_of_dog[5],list_of_dog[6], 2, original_img)
        find_marked_maxima_minima(list_of_dog[5],list_of_dog[6],list_of_dog[7], 2, original_img)

        find_marked_maxima_minima(list_of_dog[8],list_of_dog[9],list_of_dog[10], 4, original_img)
        find_marked_maxima_minima(list_of_dog[9],list_of_dog[10],list_of_dog[11], 4, original_img)

        find_marked_maxima_minima(list_of_dog[12],list_of_dog[13],list_of_dog[14], 8, original_img)
        find_marked_maxima_minima(list_of_dog[13],list_of_dog[14],list_of_dog[15], 8, original_img)

        write_image(original_img,'keypoints')


def main():

        task_2_img = cv2.imread("task2.jpg", 0)

        sigma_table = [[1/sqrt(2), 1, sqrt(2), 2, 2*sqrt(2)],
                                        [sqrt(2), 2,  2*sqrt(2), 4, 4*sqrt(2)],
                                        [2*sqrt(2), 4, 4*sqrt(2), 8, 8*sqrt(2)],
                                        [4*sqrt(2), 8, 8*sqrt(2), 16, 16*sqrt(2)]]

        generate_octavs(task_2_img, sigma_table);

        list = []
        compute_DoG(list)
        find_keypoints(task_2_img, list)



main()
```

(1) include images of the second and third octave and specify their resolution (width x height, unit pixel

5 images from second octav are given below. Each of them have the resolution of 375 x 229 pixels.

(a) sigma = $\sqrt{2}$

(b) sigma = 2

(c) sigma = $2\sqrt{2}$

(d) sigma = 4

(e) sigma = $4\sqrt{2}$

Figure 3: Generated images for Second Octav (375 x 229 px each)

5 images from third octav are given below. Each of them have the resolution of 187 x 114 pixels.

(a) sigma = $2\sqrt{2}$

(b) sigma = 4

(c) sigma = $4\sqrt{2}$

(d) sigma = 8

(e) sigma = $8\sqrt{2}$

Figure 4: Generated images for Third Octav (187 x 114 px each)

(2) include DoG images obtained using the second and third octave

(a) DoG of 1st and 2nd blur

(b) DoG of 2nd and 3rd blur

(c) DoG of 3rd and 4th blur

(d) DoG of 4th and 5th blur

Figure 5: Difference of Gaussian images for second octav

As the above images are not distinguishable easily, a normalized version has been provided below.

(a) Normalized DoG of 1st and 2nd blur

(b) Normalized DoG of 2nd and 3rd blur

(c) Normalized DoG of 3rd and 4th blur

(d) Normalized DoG of 4th and 5th blur

Figure 6: Normalized version of Figure 5

(a) DoG of 1st and 2nd blur

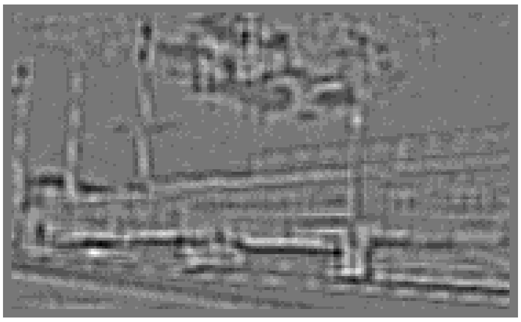
(b) DoG of 2nd and 3rd blur

(c) DoG of 3rd and 4th blur

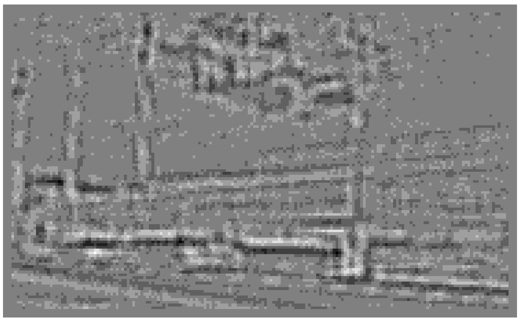
(d) DoG of 4th and 5th blur

Figure 7: Difference of Gaussian images for third octav

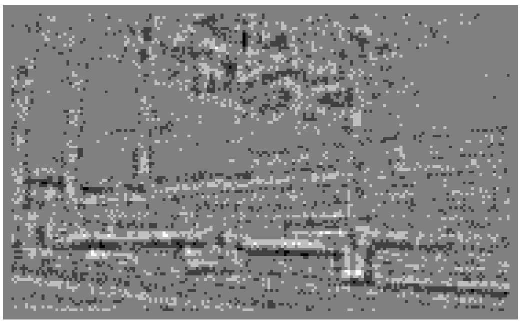
As the above images are not distinguishable easily, a normalized version has been provided below.



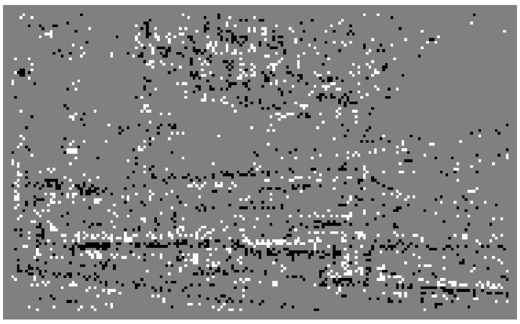
(a) Normalized DoG of 1st and 2nd blur



(b) Normalized DoG of 2nd and 3rd blur



(c) Normalized DoG of 3rd and 4th blur



(d) Normalized DoG of 4th and 5th blur

Figure 8: Normalized version of Figure 7

(3) clearly show all the detected keypoints using white dots on the original image
The keypoints has been marked in white on the original image.

Figure 9: Original image with all the keypoints

To eliminate extra keypoints a theshhold has been added. The pixels with intensity lower then 2 are being discarded.

Figure 10: Detected keypoints with a threshold

(4) provide coordinates of the five left-most detected keypoints (the origin is set to be the top-left corner)
Five left-most detected keypoints are:

1. (3, 114)
2. (3, 115)
3. (3, 228)
4. (3, 277)
5. (3, 278)

The keypoints with a black background are given below:

Figure 11: Keypoints contrasting agaist background

3 Question

For the task of cursor detection, which aims to locate the cursor in an image, two sets of images and cursor templates, named as "Set A" and "Set B", will be provided to you. Set A is composed of a total number of 25 images and 1 cursor template. Set A is for task 1., i.e., the basic cursor detection which contributes to 5 points. Set B is composed of a total number of 30 images and 3 different cursor template. Set B is for task 2., i.e., which contributes to 3 bonus points.
Cursor Detection code for Set A and Set B is given below

```
def match_template(original_image, laplacian_img, template, output_folder):

        w, h = template.shape[::-1]


        methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',
                    'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORMED']

        for meth in methods:

                oi = original_image.copy()
```

```python
                img = laplacian_img.copy()
                method = eval(meth)

                # Apply template Matching
                res = cv2.matchTemplate(img,template,method)

                min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

                if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
                    top_left = min_loc
                else:
                    top_left = max_loc
                bottom_right = (top_left[0] + w, top_left[1] + h)

                cv2.rectangle(oi,top_left, bottom_right, 255, 2)

                write_image(oi, output_folder + meth)


def match_driver(range_l, range_u, source_prefix, op_prefix, template):

        template = cv2.Laplacian(template,cv2.CV_8U)

        for img_no in range(range_l, range_u):

                original_image = cv2.imread(source_prefix + str(img_no) + '.jpg')

                img_source = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)

                laplacian_img = cv2.Laplacian(cv2.GaussianBlur(img_source, (3,3),0),cv2.CV_8U)

                output_folder = op_prefix + str(img_no)

                match_template(original_image, laplacian_img, template, output_folder)


def find_cursor():


        #Set 1 Images

        template = cv2.imread('task3/temp.jpg',0)

        #positive images

        match_driver(1,16, 'task3/pos_', 'task3_set1/pos_', template)

        #negative images

        match_driver(1,7, 'task3/neg_', 'task3_set1/neg_', template)
        match_driver(8,11, 'task3/neg_', 'task3_set1/neg_', template)


        #Set 2 Images

        #positive images
        template = cv2.imread('task3/task3_bonus/t1_x.jpg',0)

        match_driver(1,7, 'task3/task3_bonus/t1_', 'task3_set2/t1/pos_', template)

        template = cv2.imread('task3/task3_bonus/t2_x.jpg',0)

        match_driver(1,7, 'task3/task3_bonus/t2_', 'task3_set2/t2/pos_', template)
```

```python

        template = cv2.imread('task3/task3_bonus/t3_x.jpg',0)

        match_driver(1,7, 'task3/task3_bonus/t3_', 'task3_set2/t3/pos_', template)

        #negative images

        match_driver(1,7, 'task3/task3_bonus/neg_', 'task3_set2/neg/neg_', template)
        match_driver(8,13, 'task3/task3_bonus/neg_', 'task3_set2/neg/neg_', template)



def main():

        find_cursor()


main()
```

1. **Detect cursors in Set A.**
matchTemplate() function from OpenCv library has been used to find the cursors in the given image.

Figure 12: Template used to detect cursors

I have used 6 different modes to match templates:

1. cv2.TM_CCOEFF
2. cv2.TM_CCOEFF_NORMED
3. cv2.TM_CCORR
4. cv2.TM_CCORR_NORMED
5. cv2.TM_SQDIFF
6. cv2.TM_SQDIFF_NORMED

The performance for pos images are given in the below table:

Image	TM_CCOEFF_NORMED	TM_CCOEFF	TM_CCORR_NORMED	TM_CCORR	TM_SQDIFF_NORMED	TM_SQDIFF
pos_1	Yes	Yes	Yes			Yes
pos_2	Yes	Yes	Yes	Yes		Yes
pos_3	Yes	Yes	Yes	Yes		Yes
pos_4	Yes	Yes	Yes	Yes		Yes
pos_5	Yes	Yes	Yes	Yes		Yes
pos_6	Yes	Yes	Yes	Yes		Yes
pos_7	Yes	Yes	Yes	Yes		Yes
pos_8			Yes			
pos_9	Yes	Yes	Yes	Yes		Yes
pos_10	Yes		Yes			Yes
pos_11	Yes	Yes	Yes	Yes		Yes
pos_12	Yes	Yes	Yes	Yes		Yes
pos_13	Yes	Yes	Yes			Yes
pos_14	Yes	Yes	Yes			Yes
pos_15	Yes	Yes	Yes	Yes		Yes

From the table we can see, the best performing mode in this case is TM_CCORR_NORMED, which were able to detect cursor in all 15 images.

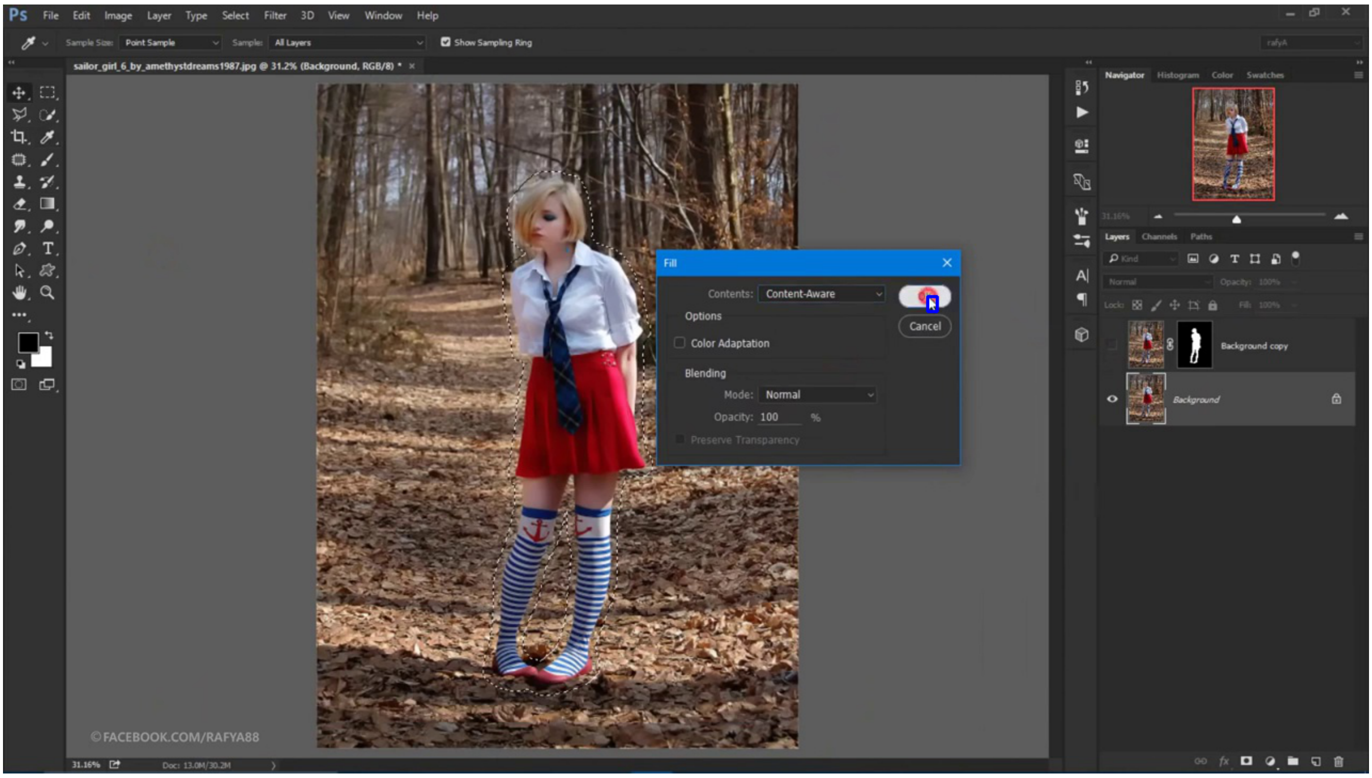


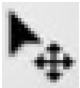
Figure 13: Example of cursor detection using cv2.TM_CCOEFF

Observation:

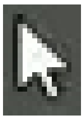
1. In some of the images, the cursor from photoshop toolset was detected instead of the intended cursor.
2. for a few neg images, the code is returning some false positives.

2. Detect cursors in Set B.

(a) template 1



(b) template 2



(c) template 3

Figure 14: Templates chosen for Set B

Performance for pos images in Set B.

Image	TM_CCOEFF_NORMED	TM_CCOEFF	TM_CCORR_NORMED	TM_CCORR	TM_SQDIFF_NORMED	TM_SQDIFF
pos_1						
pos_2	Yes	Yes	Yes	Yes		Yes
pos_3	Yes	Yes	Yes	Yes		Yes
pos_4	Yes	Yes	Yes	Yes		Yes
pos_5		Yes				Yes
pos_6	Yes	Yes	Yes	Yes		Yes

Observation:

1. for a few neg images, the code is returning some false positives.

References

[1] OpenCV Documentation. Template matching, 2014.

[2] Utkarsh Sinha. Sift: Theory and practice, 2016. aishack.in.