CSE 573 - Project 1

Atrayee Nag (#50288651)

October 2018

1 Edge Detection

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 below has been used to detect the edges of the image along x-axis and y-axis. I have implemented the Sobel operation for edge detection and have normalized the image to change the pixel intensity values to a familiarized range.

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


def apply_sobel_filter(sobel, img, edge, kernel_radius):
    height, width = img.shape
    for x in range(kernel_radius, height-kernel_radius):
        for y in range(kernel_radius, width-kernel_radius):
            loop_end = (kernel_radius*2)+1
            sum = 0
            for i in range(0,loop_end):
                for j in range(0,loop_end):
                    sum += sobel[i][j] * img[x-kernel_radius+i][y-kernel_radius+j]

            edge[x][y] = sum
    return edge


def edge_detection():
    img = cv2.imread("edge.png", 0)
    show_image(img)

    # Detect edge along x-direction
    sobel_x = [[-1,0,1],
               [-2,0,2],
               [-1,0,1]]

    height, width = img.shape
    edge_x = [[0.0 for col in range(width)] for row in range(height)]
    apply_sobel_filter(sobel_x, img, edge_x, 1)
    edge_x = np.asarray(edge_x)
    normalize_edge(edge_x)
    show_image(edge_x)

    # Detect edge along y-direction
    sobel_y = [[-1,-2,-1],
               [0,0,0],
               [1,2,1]]
    edge_y = [[0.0 for col in range(width)] for row in range(height)]
```

```python
    apply_sobel_filter(sobel_y, img, edge_y, 1)
    edge_y = np.asarray(edge_y)
    normalize_edge(edge_y)
    show_image(edge_y)


# Normalize image by dividing absolute value of pixels with the max value
def normalize_edge(edge):
    height, width = edge.shape
    max_val=0
    for i in range(0, height):
        for j in range(0, width):
            edge[i][j] = abs(edge[i][j])
            max_val = max(max_val,edge[i][j])
    for i in range(0, height):
        for j in range(0, width):
            edge[i][j] = edge[i][j]/max_val


def show_image(img):
    cv2.namedWindow('blur_dir', cv2.WINDOW_NORMAL)
    cv2.imshow('blur_dir', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


edge_detection()
```

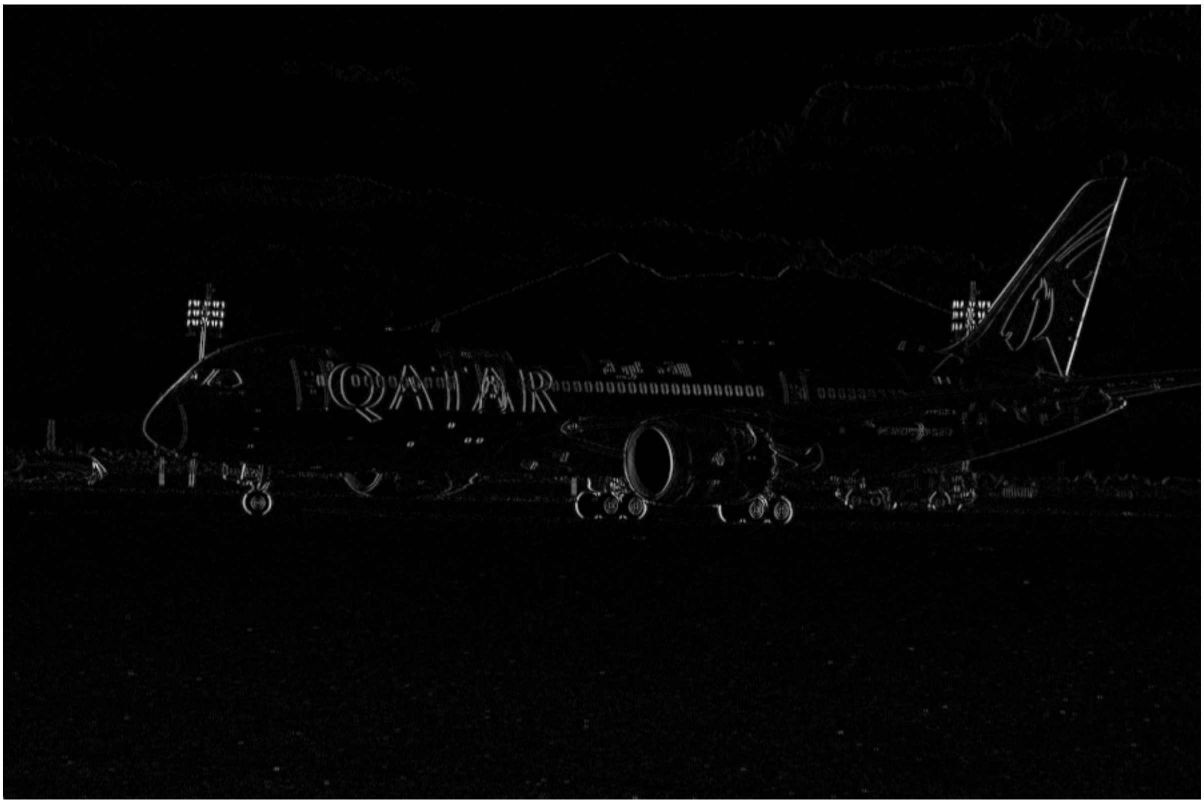


Figure 1: Edge along x direction

Figure 2: Edge along y direction

2 Keypoint Detection

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). 1. Generate four octaves. Each octave is composed of five images blurred using Gaussian kernels. For each octave, the bandwidth parameters (five different scales) of the Gaussian kernels are shown in Tab. 1. 2. Compute Difference of Gaussian (DoG) for all four octaves. 3. Detect keypoints which are located at the maxima or minima of the DoG images. You only need to provide pixel-level locations of the keypoints you do not need to provide sub-pixel-level locations.

The code for the above mentioned steps of the SIFT operation for keypoint detection is given below:

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

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


def apply_sobel_filter(sobel, img, edge, kernel_radius):
    height, width = img.shape
    for x in range(kernel_radius, height-kernel_radius):
        for y in range(kernel_radius, width-kernel_radius):
            loop_end = (kernel_radius*2)+1
            sum = 0
            for i in range(0,loop_end):
                for j in range(0,loop_end):
                    sum += sobel[i][j] * img[x-kernel_radius+i][y-kernel_radius+j]

            edge[x][y] = sum
    return edge
```

```python


def get_gaussian_kernel(sigma):
    kernel_radius = 3
    # compute the kernel elements
    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]
    # normalize the kernel elements
    kernelsum = sum([sum(row) for row in kernel2d])
    kernel2d = [[x/kernelsum for x in row] for row in kernel2d]
    return kernel2d


def apply_gaussian_blur(img, sigma):
    height, width = img.shape
    blur_image = [[0 for col in range(width)] for row in range(height)]
    blur_image = np.asarray(blur_image)
    blur_image = apply_sobel_filter(get_gaussian_kernel(sigma), img, blur_image, 3)
    return blur_image


def blurr_individual_images(img, octave_sigma, octave):
    for i in range(len(octave_sigma)):
        cv2.imwrite("blurr_"+octave+"_sigma"+str(i+1)+ ".jpg", apply_gaussian_blur(img, octave_sigma[i]))


def get_scaled_image(img, factor):
    x = 2
    height, width = img.shape
    octave_img = [[0 for col in range(width/2)] for row in range(height/2)]
    octave_x = 0
    for x in range(0, height):
        octave_y = 0
        if x%2 == 0:
            continue
        for y in range(0, width):
            if y%2 == 0:
                continue
            octave_img[octave_x][octave_y] = img[x][y]
            octave_y += 1
        octave_x += 1
    return np.asarray(octave_img)


def get_gauss_blurred_octaves(img, octave):
    octave1_sigma = np.array([1/sqrt(2), 1, sqrt(2), 2, 2*sqrt(2)])
    octave2_sigma = np.array([sqrt(2), 2, 2*sqrt(2), 4, 4*sqrt(2)])
    octave3_sigma = np.array([2*sqrt(2), 4, 4*sqrt(2), 8, 8*sqrt(2)])
    octave4_sigma = np.array([4*sqrt(2), 8, 8*sqrt(2), 16, 16*sqrt(2)])
    if octave == "octave1":
        blurr_individual_images(img, octave1_sigma, octave)
    if octave == "octave2":
        blurr_individual_images(img, octave2_sigma, octave)
    if octave == "octave3":
        blurr_individual_images(img, octave3_sigma, octave)
    if octave == "octave4":
        blurr_individual_images(img, octave4_sigma, octave)


def get_difference_of_gaussian(list_dog):
    for i in range(0, 4):
        for j in range(0, 4):
```

```
            img1 = cv2.imread("blurr_octave"+str(i+1)+"_sigma"+str(j+1)+".jpg", 0)
            img2 = cv2.imread("blurr_octave"+str(i+1)+"_sigma"+str(j+2)+".jpg", 0)
            height, width = img1.shape
            img3 = [[0 for col in range(width)] for row in range(height)]
            for x in range(0,height):
                for y in range(0, width):
                    img3[x][y] = int(img2[x][y]) - int(img1[x][y])
            img3 = np.asarray(img3)
            cv2.imwrite("octave_"+str(i+1)+"_dog"+str(j+1)+ ".jpg", img3)
            norm_img = cv2.normalize(img3, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
            cv2.imwrite("octave_"+str(i+1)+"_normalized_dog"+str(j+1)+ ".jpg", norm_img)
            list_dog.append(img3)
    return list_dog


def get_maxima_minima(dog_top, dog_middle, dog_bottom, scale, output_image):
    height, width = dog_middle.shape
    is_maxima = True
    is_minima = True
    for h in range(1, height-1):
        for w in range(1, width-1):
            if dog_middle[h][w]<2:
                continue
            is_maxima = True
            is_minima = True
            for x in range(h-1,h+2):
                for y in range(w-1,w+2):
                    if (dog_middle[h][w] < dog_middle[x][y])
                    or (dog_middle[h][w] < dog_top[x][y])
                    or (dog_middle[h][w] < dog_bottom[x][y]):
                        is_maxima = False
                        break

            for x in range(h-1,h+2):
                for y in range(w-1,w+2):
                    if (dog_middle[h][w] > dog_middle[x][y])
                    or (dog_middle[h][w] > dog_top[x][y])
                    or (dog_middle[h][w] > dog_bottom[x][y]):
                        is_minima = False
                        break
            if is_maxima or is_minima:
                output_image[h*scale][w*scale] = 255


def generate_keyPoints(list_dog):
    output_image = cv2.imread("keypoint.jpg", 0)
    get_maxima_minima(list_dog[0], list_dog[1], list_dog[2], 1, output_image)
    get_maxima_minima(list_dog[1], list_dog[2], list_dog[3], 1, output_image)
    get_maxima_minima(list_dog[4], list_dog[5], list_dog[6], 2, output_image)
    get_maxima_minima(list_dog[5], list_dog[6], list_dog[7], 2, output_image)
    get_maxima_minima(list_dog[8], list_dog[9], list_dog[10], 4, output_image)
    get_maxima_minima(list_dog[9], list_dog[10], list_dog[11], 4, output_image)
    get_maxima_minima(list_dog[12], list_dog[13], list_dog[14], 8, output_image)
    get_maxima_minima(list_dog[13], list_dog[14], list_dog[15], 8, output_image)
    cv2.imwrite("keypoint_detected.jpg", output_image)


def find_keypoints():
    img = cv2.imread("keypoint.jpg", 0)
    get_gauss_blurred_octaves(img, "octave1")
    octave2_image = get_scaled_image(img, 2)
    get_gauss_blurred_octaves(octave2_image, "octave2")
    octave3_image = get_scaled_image(octave2_image, 2)
```

```
    get_gauss_blurred_octaves(octave3_image, "octave3")
    octave4_image = get_scaled_image(octave3_image, 2)
    get_gauss_blurred_octaves(octave4_image, "octave4")
    list_dog = []
    list_dog = get_difference_of_gaussian(list_dog)
    generate_keyPoints(list_dog)


find_keypoints()
```

1. **Include images of the second and third octave and specify their resolution** $width * height, unit pixel$

   The Gaussian blurred images of the second octave which is scaled down by 2 are displayed below. The dimension of each image is 375 x 229

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

(b) sigma = 4

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

(d) sigma = 8

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

Figure 3: Images for Second Octave

The Gaussian blurred images of the third octave which is scaled down by 4 are displayed below. The dimension of each image is 187 x 114

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

(b) sigma = 4

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

(d) sigma = 8

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

Figure 4: Images for Third Octav

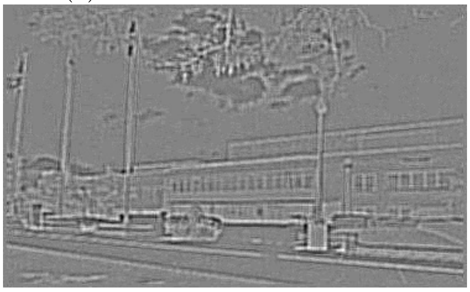
2. **Include DoG images obtained using the second and third octave.**

   Please find the four difference of Gaussian images in the second octave(original image scaled down to 1/2) along with their normalized forms below:

(a) Octave 2 $D_{OG}1$

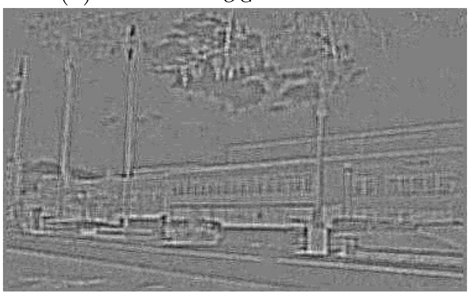
(b) Octave 2 $D_{OG}1Normalized$

(c) Octave 2 $D_{OG}1$

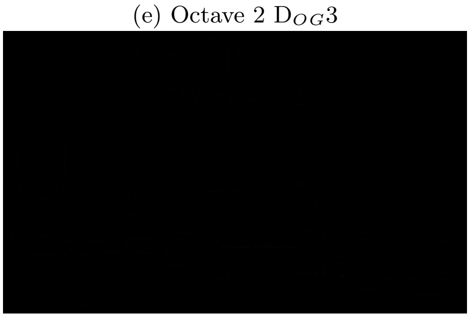


(d) Octave 2 $D_{OG}2Normalized$

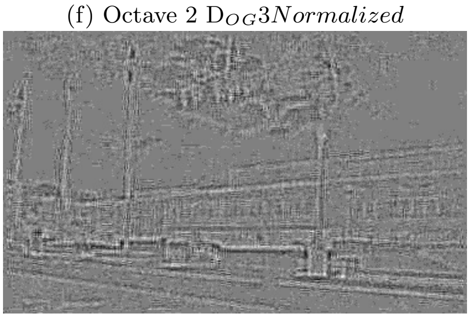
(e) Octave 2 $D_{OG}3$



(f) Octave 2 $D_{OG}3Normalized$



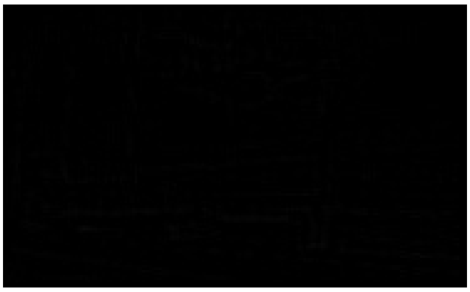
(g) Octave 2 $D_{OG}4$



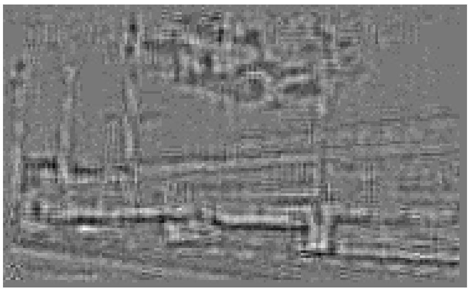
(h) Octave 2 $D_{OG}4Normalized$

Figure 5: DOG images for Second Octave

Please find the four difference of Gaussian images in the third octave(original image scaled down to 1/4) along with their normalized forms below:

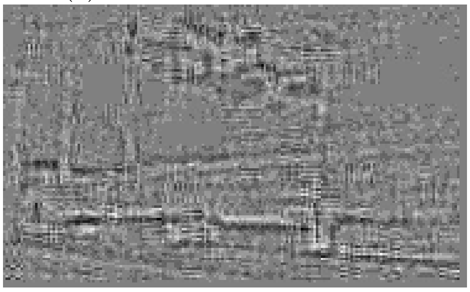


(a) Octave 3 $D_{OG}1$



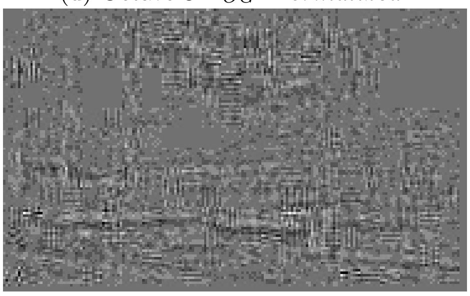
(b) Octave 3 $D_{OG}1Normalized$

(c) Octave 3 $D_{OG}2$

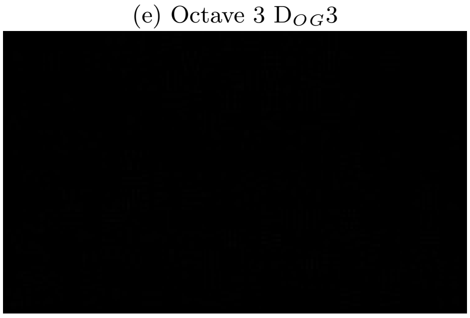


(d) Octave 3 $D_{OG}2Normalized$

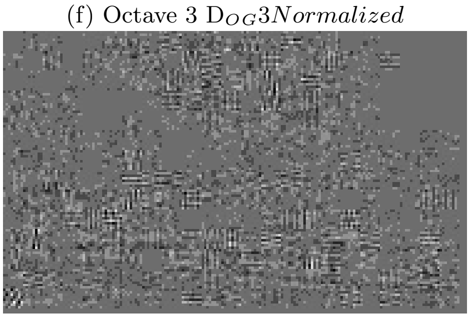
(e) Octave 3 $D_{OG}3$



(f) Octave 3 $D_{OG}3Normalized$



(g) Octave 3 $D_{OG}4$



(h) Octave 3 $D_{OG}4Normalized$

Figure 6: $D_{OG}ImagesforThirdOctave$

3. **Clearly show all the detected keypoints using white dots on the original image.**

   The below image shows all the keypoints detected on the original image. The keypoint pixels have been set to 255 to mark them with white.

Figure 7: Keypoints Detection on Original Image

Since the above image has detected a large number of keypoints, to make the image visibly clear, I have discarded some pixels from the difference of Gaussian. For this I have used the threshold technique, where the threshold has been set to 2 and anything below is eliminated. The original image after applying the threshold is displayed below:

Figure 8: Keypoint Detection on Original Image Normalized

4. **Provide coordinates of the five left-most detected keypoints (the origin is set to be the top-left corner).**

   The 5 coordinates of the leftmost keypoints are given below:

   Keypoint 1 : **(1, 85)**
   Keypoint 2 : **(1, 87)**
   Keypoint 3 : **(1, 120)**

Keypoint 4 : **(1, 122)**
Keypoint 5 : **(1, 444)**

3 Cursor Detection

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.

1. **Detect cursors in Set A.**

   Please find the code below which would detect the cursors in Set A and Set B by several template matching mechanisms.

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


def show_image(img):
    cv2.namedWindow('blur_dir', cv2.WINDOW_NORMAL)
    cv2.imshow('blur_dir', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


def match_template(img, template, img_gaus_laplace, template_laplace):
    img2 = img_gaus_laplace.copy()
    w, h = template.shape[::-1]
    # All the 6 methods for comparison in a list
    methods = ['cv2.TM_SQDIFF_NORMED']
    for m in methods:
        img = img2.copy()
        method = eval(m)
        # Apply template Matching on the blurred Laplacian image and the Laplacian template
        res = cv2.matchTemplate(img,template_laplace,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(img,top_left, bottom_right, 255, 2)
        show_image(img)

def find_cursor():
    template_1 = cv2.imread('template.png',0)
    template_laplace_1 = cv2.Laplacian(template_1,cv2.CV_8U)
    Check cursor for positive images for Set A
    for i in range(1,16):
        print("Positive images")
        img_1_pos = cv2.imread("pos_"+str(i)+".jpg",0)
        img_gaus_1_pos = cv2.GaussianBlur(img_1_pos, (3,3), 0)
        img_gaus_laplace_1_pos = cv2.Laplacian(img_gaus_1_pos,cv2.CV_8U)
        match_template(img_1_pos, template_1, img_gaus_laplace_1_pos, template_laplace_1);

    # Check cursor for negative images for Set B
    for i in range(1,10):
        print("Negative images")
```

```
        img_1_neg = cv2.imread("neg_"+str(i)+".jpg",0)
        img_gaus_1_neg = cv2.GaussianBlur(img_1_neg, (3,3), 0)
        img_gaus_laplace_1_neg = cv2.Laplacian(img_gaus_1_neg,cv2.CV_8U)
        match_template(img_1_neg, template_1, img_gaus_laplace_1_neg, template_laplace_1);

    #Check cursor for task 3 Set B
    for i in range(1, 4):
        template_2 = cv2.imread("t"+str(i)+".jpg",0)
        # Perform Laplacian on the template
        template_laplace_2 = cv2.Laplacian(template_2,cv2.CV_8U)
        for j in range(1,7):
            img_2 = cv2.imread("t"+str(i)+"_"+str(j)+".jpg",0)
            # Perform Gaussian and Laplacian on the image
            img_gaus_2 = cv2.GaussianBlur(img_2, (3,3), 0)
            img_gaus_laplace_2 = cv2.Laplacian(img_gaus_2,cv2.CV_8U)
            match_template(img_2, template_2, img_gaus_laplace_2, template_laplace_2);

    # Check cursor for negative images for Task 3 Set B
    for i in range(1, 4):
        template_2 = cv2.imread("t"+str(i)+".jpg",0)
        # Perform Laplacian on the template
        template_laplace_2 = cv2.Laplacian(template_2,cv2.CV_8U)
        for i in range(1,10):
            print("Negative images")
            img_1_neg = cv2.imread("neg_"+str(i)+".jpg",0)
            img_gaus_1_neg = cv2.GaussianBlur(img_1_neg, (3,3), 0)
            img_gaus_laplace_1_neg = cv2.Laplacian(img_gaus_1_neg,cv2.CV_8U)
            match_template(img_1_neg, template_1, img_gaus_laplace_1_neg, template_laplace_1);

find_cursor()
```

I have used the template shown below to detect the cursor in Set A positives and negative images.

Figure 9: Template for SetA

I have used 6 methods for template matching listed below:

1. CV$_T M_S QDIFF$ – **Squared Difference**
2.$CV_T M_S QDIFF_N ORMED$ – **Squared Difference Normalized**
3.$CV_T M_C CORR$ – **Cross Correlation**
4.$CV_T M_C CORR_N ORMED$ – **Cross Correlation Normalized**
5.$CV_T M_C COEFF$ – **Cross Coefficients**
6.$CV_T M_C COEFF_N ORMED$ – **Cross Coefficients Normalized**

Please find the table below to show the result of the template matching operations on the positive images of Set A:

Image	SQDIFF	SQDIFF_NORM	CCORR	CCORR_NORM	CCOEFF	CCOEFF_NORM
Pos 1	YES	NO	NO	YES	YES	YES
Pos 2	YES	NO	YES	YES	YES	YES
Pos 3	YES	NO	YES	YES	YES	YES
Pos 4	YES	NO	YES	YES	YES	YES
Pos 5	YES	NO	YES	YES	YES	YES
Pos 6	YES	NO	YES	YES	YES	YES
Pos 7	YES	NO	YES	YES	YES	YES
Pos 8	NO	NO	NO	NO	NO	NO
Pos 9	NO	NO	NO	NO	YES	NO
Pos 10	YES	NO	NO	YES	NO	YES
Pos 11	YES	NO	YES	YES	YES	YES
Pos 12	YES	NO	YES	YES	YES	YES
Pos 13	YES	NO	NO	YES	YES	YES
Pos 14	YES	NO	YES	YES	YES	YES
Pos 15	YES	NO	YES	YES	YES	YES

Assumptions and Observations:

1. From the above table, we can observe that Cross Correrational Normalization, Squared Difference, Cross Coefficient and Cross Coefficient Normalization has produced the best result.

2. The results would vary if we take a different template for matching. I have taken a template which produces best results.

3. For the positive images, false points i.e. the cursor of the photoshop toolbox are getting detected as well.

4. For the negative images, we are getting false cursor points for some images.

Please find a screen-shot of the cursor detected with Squared Difference template matching:

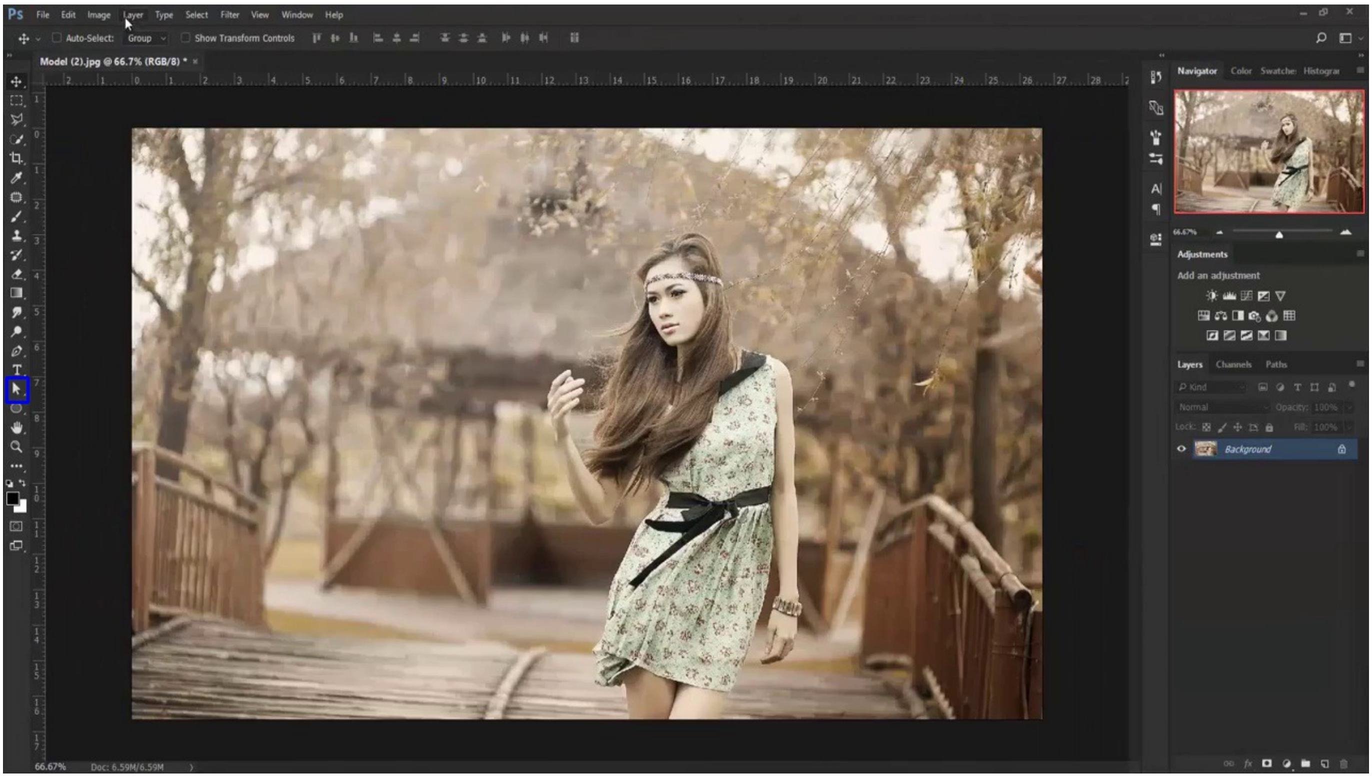


Figure 10: Template Matching in Set A

2. **Detect cursors in Set B.**

I have used the three templates below to detect the cursor in Set B positive and negative images:

Figure 11: Template 1 for SetB

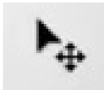


Figure 12: Template 2 for SetB

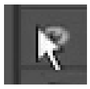


Figure 13: Template for SetB

Please find the table below to show the result of the template matching operations on the positive images of Set B:

Image	SQDIFF	SQDIFF_NORMED	CCORR	CCORR_NORMED	CCOEFF	CCOEFF_NORMED
T1_1	NO	NO	NO	NO	NO	NO
T1_2	YES	NO	YES	NO	YES	NO
T1_3	YES	NO	YES	YES	YES	YES
T1_4	YES	NO	YES	YES	YES	YES
T1_5	YES	NO	YES	YES	YES	YES
T1_6	YES	NO	YES	YES	YES	YES
T2_1	YES	YES	NO	YES	YES	YES
T2_2	NO	NO	NO	NO	NO	NO
T2_3	YES	NO	YES	YES	YES	YES
T2_4	YES	NO	YES	YES	YES	YES
T2_5	YES	NO	YES	YES	YES	YES
T2_6	YES	NO	YES	YES	YES	YES
T3_1	YES	NO	YES	YES	YES	YES
T3_2	YES	NO	NO	YES	YES	YES
T3_3	YES	NO	YES	YES	YES	YES
T3_4	YES	NO	YES	YES	YES	YES
T3_5	YES	NO	YES	YES	YES	YES
T3_6	YES	NO	NO	YES	YES	YES

Assumptions and Observations:

1. From the above table, we can observe that Cross Correrational Normalization, Squared Difference and Cross Coefficient has produced the best result.

2. The results would vary if we take a different template for matching. I have taken a template which produces best results.

4. For the negative images, we are getting false cursor points for some images.

Please find a screen-shot of the cursor detected with Squared Difference template matching:

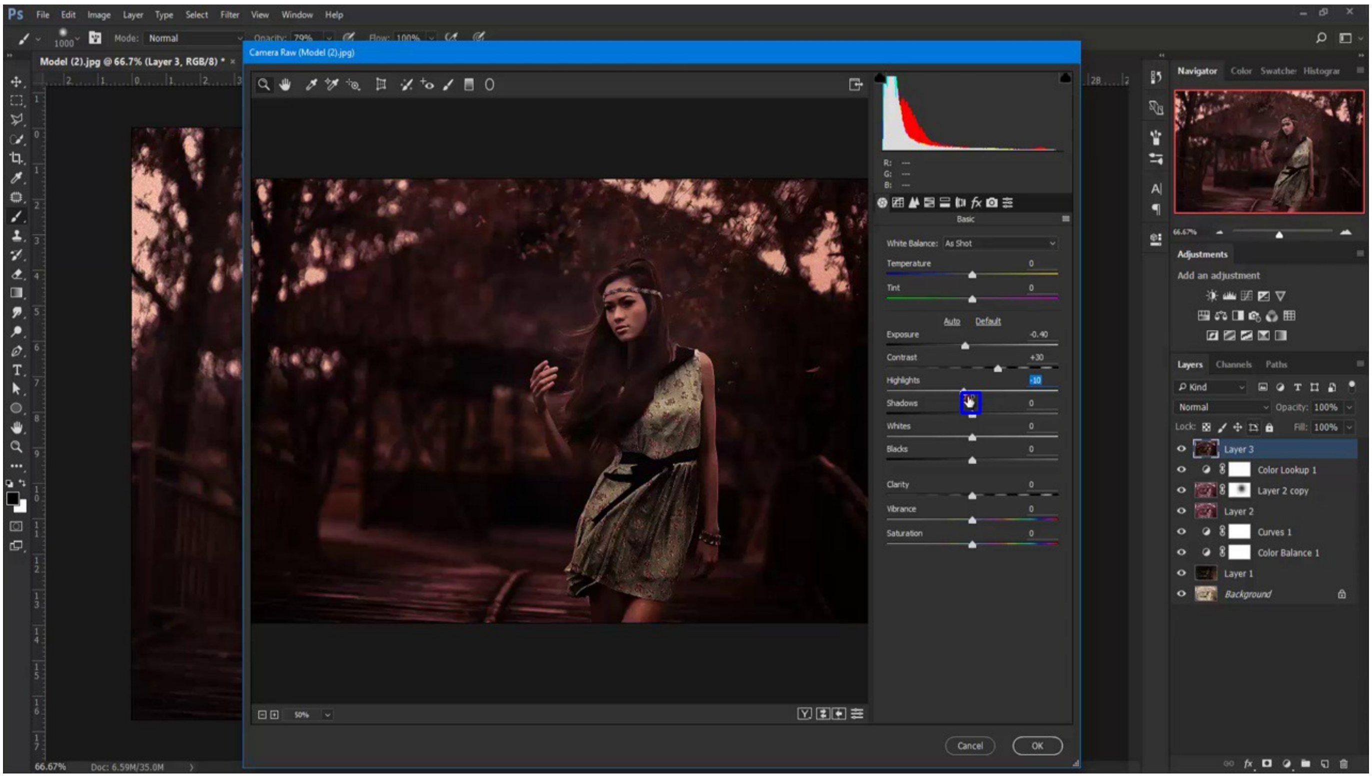


Figure 14: Template Matching in Set B

4 References

The resources that has helped me understand SIFT operation and template matching are as follows:

- SIFT: Theory and Practice : http://aishack.in/tutorials/sift-scale-invariant-feature-transform-log-approximation/
- Template Matching Open CV: https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template$_{m}$*atching/tem*