ISTANBUL TECHNICAL UNIVERSITY

Faculty of Computer Science and Informatics

BLG453E

Computer Vision Homework-3 Report

Author

Cansu Yanık -150170704

NOTE: Outputs can be access via the Drive link below. I also added a video for part4

https://drive.google.com/open?id=1UpIehatrLbnbzBiGog392I5Zcn-MwDh4

1. Part 1: Sobel Filter

For this part, I wrote a function called *sobel_filter(image)* and it takes image as parameter. Figure-1 shows inside of the function.

```
def sobel_filter(image):

    row, col, ch = image.shape

    #Convert grayscale
    image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

    #Gaussian Smoothing
    image = cv2.GaussianBlur(image,(5,5),0)


    #vertical and horizontal masks
    Gx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    Gy = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    #Holds convolution results, matrices
    sobelx = np.zeros((row,col,1), np.uint8)
    sobely = np.zeros((row,col,1), np.uint8)

    #Padding for convolution
    image_temp = np.zeros((row + 2, col + 2))
    image_temp[1:-1, 1:-1] = image

    #Convolve image with kernels
    sobelx = convolve2d(image_temp, Gx, mode='same', boundary='fill', fillvalue=0)
    sobely = convolve2d(image_temp, Gy, mode='same', boundary='fill', fillvalue=0)


    #Remove padding part
    sobelx = sobelx[1:-1, 1:-1]
    sobely = sobely[1:-1, 1:-1]

    #Edge magnitude
    image = (sobelx*sobelx + sobely*sobely)**0.5

    '''
    #You can open comments to see and save the image
    cv2.imshow('Output', image)
    cv2.waitKey()
    cv2.imwrite("Part1.jpg", image)
    '''

    return sobelx, sobely
```

Figure-1: sobel_filter Funtion

I used *convolve2d* function from *scipy.signal* package, but in addition, I wrote my own convolution operation. Since it takes more times than the *convolve2d,* I used convolve2d. Function returns *sobelx* and *sobely* matrices for other parts. And, also I found edge magnitude and uploaded the image which shows edges named *Part2.jpg.* Edges can be a little thick due to Gaussian smoothing.

Note: Since I want to use only desired rectangle area of the screenshots, I did an operation to crop the image. I used this cropped version for all parts. Figure-2 shows the code.

```
#This part crops desired area from the whole screenshot
row, col, ch = image.shape
centerx = int(col/2)
centery = int(row/2)

upLeft = (int(col / 4), int(row / 4.5))
bottomRight = (int(col * 3 / 4), int(row * 3 / 4))
image = image[upLeft[1]:bottomRight[1], upLeft[0]:bottomRight[0]]
```

Figure-2: Cropping Operation

2. Part 2: Canny Edge Detector

In the second part, I wrote a function called *canny_edge_detector(image)* and it takes image as parameter. Figure-3 shows inside of the function.

```
def canny_edge_detector(image):

    row, col, ch = image.shape

    #Since Canny is sensitive to noise, first apply smoothing by using Gaussian kernel
    image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    image = cv2.GaussianBlur(image,(5,5),0)

    #I used threshold method to find optimum threshold and lower one is 3 times less than max threshold
    max_treshold,img = cv2.threshold(image,0,255,cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    low_treshold = max_treshold/3

    #Apply Canny filter
    image = cv2.Canny(image,low_treshold,max_treshold)

    #I applied contour and draw it
    contours, hierarchy = cv2.findContours(image.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    image = cv2.drawContours(image, contours, -1, (255,255,255), 1)

    return image
```

Figure-3: canny_edge_detector Function

Since canny function need an upper and a lower threshold to determine both strong enough edge points and it uses them to connect other edges to them, I used cv2.threshold() method to find optimum upper threshold. And since usually the higher threshold being 3 times the lower, I also found the lower threshold. The output image is shown below.

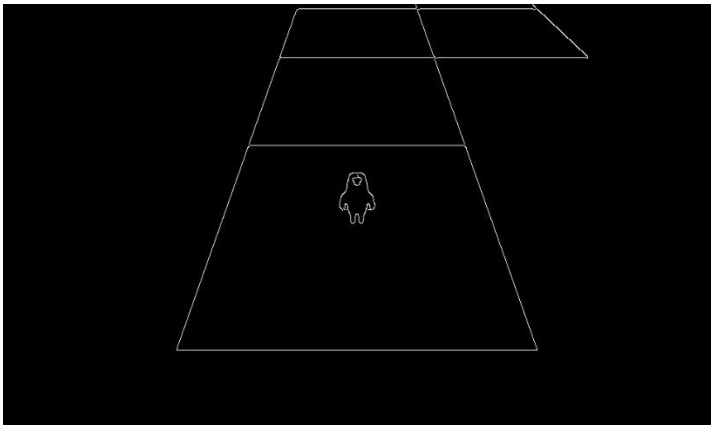

Figure-4: Output of Canny Edge Detection

3. Part 3: Minimum Eigenvalue Corner Detector

In the third part, I wrote a function called *min_eigenvalue_corner_detector(image)* and it takes image as parameter. Figure-5 shows inside of the function. In the code, you can see my comments that explains what I did. The output of this part is also shown below.

```python
def min_eigenvalue_corner_detector(image):

    row, col, ch = image.shape

    #Apply sobel filter to take derivatives
    sobelx, sobely = sobel_filter(image)

    #To be able to get image structure tensor(2x2), convolve each values with Gaussian kernel
    M_00 = cv2.GaussianBlur(sobelx*sobelx,(5,5),0)
    M_01 = cv2.GaussianBlur(sobelx*sobely,(5,5),0)
    M_11 = cv2.GaussianBlur(sobely*sobely,(5,5),0)

    #minimum eigen value calculation (I got a reference from our lecture slides)
    min_eigen_values = 0.5 * ((M_00 + M_11) - ((M_00 - M_11)**2 + 4*(M_01**2))**0.5)
    #min_eigen_values = cv2.dilate(min_eigen_values,None)



    image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    image = cv2.cvtColor(image,cv2.COLOR_GRAY2BGR)

    #I applied a threshold to take sharp corners and changed the colors
    image[min_eigen_values>0.15*min_eigen_values.max()]=[0,0,255]

    #Holds corners coordinates
    corner_x = []
    corner_y = []

    #To be able to see corners clearly, I traversed pixels, found red pixels, took coordinates and drew rectangles
    for i in range(row):
        for j in range(col):
            if (image[i][j][0] == 0 and image[i][j][1] == 0 and image[i][j][2] == 255):
                corner_x.append(j)
                corner_y.append(i)
                image = cv2.rectangle(image, (j, i), (j + 5, i + 5), (0, 255, 0), -1)

    '''
    #You can open comments to show and save output
    cv2.imshow('Output', image)
    #cv2.imwrite("Part3.jpg", image)
    cv2.waitKey()
    '''
    return corner_x, corner_y
```

Figure-5: min_eigenvalue_corner_detector(image) Function

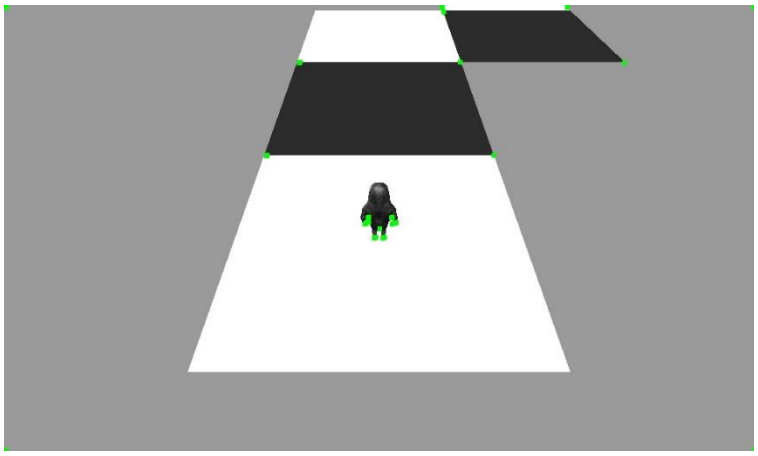

Figure-6: Corners of the Image

4. Part 4: "As I walk through the valley of the shadow of death..."

In this part, my goal is finding longest distance between center of image (monster itself) and corner points, taking this point and moving the monster to this point. Therefore, I wrote a function called *largets_point(image)* and it takes image as parameter. In the code, you can see my comments that explains what I did. Figure-7 shows the code. It returns farthest point's x and y values that is found. I just used the corner points.

Note: In my computer, I can reach the end but I am not sure for a different laptop. I recommend you run the game with big size screen.

```python
def largets_point(image):


    #Finds the center of image, this coordinate will be used for distance calculation
    row, col, ch = image.shape
    centerx = int(col/2)
    centery = int(row/2)

    #Finds image eigenvalues of image
    corner_x, corner_y = min_eigenvalue_corner_detector(image)

    for i in range(len(corner_x)):
        image = cv2.rectangle(image, (corner_x[i], corner_y[i]), (corner_x[i] + 5, corner_y[i] + 5), (0, 255, 0), -1)


    #Draw center of image
    image = cv2.rectangle(image, (centerx, centery), (centerx + 5, centery + 5), (0, 0, 255), -1)


    #Using corner points, finds the farthest point from the center coordinate
    #Since my algorithm finds also 4 corner of the image shape (0-0, 0-row, col-0, col-row), I checked them to ignore
    large_distance = 0

    for i in range(len(corner_x)):
        if(corner_x[i] != 0 and corner_y[i] !=0) and (corner_x[i] != 0 and corner_y[i] !=row) and (corner_x[i] != col and corner_y[i] !=0) and (corner_x[i] != col and corner_y[i] !=row):
            distance = ((corner_x[i] - centerx)**2 + (corner_y[i] - centery)**2)**0.5
            if distance > large_distance:
                if centery - corner_y[i] >= 0:
                    large_distance_x = corner_x[i]
                    large_distance_y = corner_y[i]
                    large_distance = distance

    #Draws the found point
    image = cv2.rectangle(image, (large_distance_x, large_distance_y), (large_distance_x + 8, large_distance_y + 8), (255, 255, 255), -1)

    '''
    #You can open comments to see output
    cv2.imshow('Output', image)
    cv2.waitKey()
    '''

    return large_distance_x, large_distance_y
```

Figure-7: largest_point Function

In Figure-8, walking operation for monster is given. In two steps, monster can reach the end of the path. I just use the farthest point to reach the end. Also, I applied a limit to stop the monster on proper locations.

```python
#Game can finish in two step
#I recommend to wait between two step. Because it sleeps to take screenshot
for i in range(2):

    time.sleep(5)
    #Takes the screenshot, saves and reads
    myScreenshot = pyautogui.screenshot()
    myScreenshot.save('test.png')

    image = cv2.imread("./test.png")

    #Finds center coordinates of image
    row, col, ch = image.shape
    centerx = int(col/2)
    centery = int(row/2)

    #This part crops desired area from the whole screenshot
    upLeft = (int(col / 4), int(row / 4.5))
    bottomRight = (int(col * 3 / 4), int(row * 3 / 4))
    image = image[upLeft[1]:bottomRight[1], upLeft[0]:bottomRight[0]]

    #Finds larger distance between the corner and center coordinates
    large_distance_x, large_distance_y = largets_point(image)

    #run the monster to a proporsion of the distance
    for i in range(int((centery - large_distance_y)*0.04)):
        pyautogui.keyDown('shift')
        pyautogui.keyDown('w')

    pyautogui.keyUp('shift')
    pyautogui.keyUp('w')

    #This if condition is for making stop the monter when it reaches the end.
    if (((centerx - large_distance_x)*0.04) < 11):
        for i in range(int((centerx - large_distance_x)*0.06)):
            pyautogui.keyDown('shift')
            pyautogui.keyDown('d')

    pyautogui.keyUp('shift')
    pyautogui.keyUp('d')
```

Figure-8: Walking part for monster to reach the end of path

5. Part 4: Otsu Thresholding

In this part, I tried to find optimum threshold to do segmentation with my own calculations and then I found the optimum threshold by using existing function from cv2 (*cv2.threshold*). After that, I compared these two values and showed that optimum threshold can be found by maximize the between class variance.

Firstly, I took screenshot, saved and loaded the image, cropped the image and found the histogram of the image.

```
#You can take screenshot
#Takes screenshot, saves it and loads
time.sleep(5)

myScreenshot = pyautogui.screenshot()
myScreenshot.save('test.png')


image = cv2.imread("./test.png")

#This part crops desired area from the whole screenshot
row, col, ch = image.shape
centerx = int(col/2)
centery = int(row/2)

upLeft = (int(col / 4), int(row / 4.5))
bottomRight = (int(col * 3 / 4), int(row * 3 / 4))
image = image[upLeft[1]:bottomRight[1], upLeft[0]:bottomRight[0]]

#Convert image to grayscale
image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

histogram = cv2.calcHist([image],[0],None,[256],[0,256])
```

Figure-9: Finding histogram of the image

Then, I wrote an function called *calc_variance(start, end)*. Function takes two parameters. These parameters are the range of the current threshold. I sent these regions: 0 to Threshold, and Threshold+1 to 256 for background and foreground sections.

```
#Function takes a range of the treshold and calculates mean weight and variance
def calc_variance(start, end):

    #Finds total intensity
    total_pixel = 0
    for i in range(len(histogram)):
        total_pixel = total_pixel + histogram[i]

    #canculates weight and mean
    W = 0
    M = 0
    Mean = 0
    for i in range(start, end):
        W = W + histogram[i]
        M = M + i*histogram[i]
    Weight = W / total_pixel
    if W!=0: Mean = M / W

    #Calculates variance
    V = 0
    Variance = 0
    for i in range(start, end):
        V = V + (((i - Mean)**2) * histogram[i])
    if W!=0: Variance = V / W
    return Weight, Mean, Variance
```

Figure-10: Weight, Mean and Variance Calculation

$$\sigma^2_{between}(T) = \sigma^2_{total}(T) - \sigma^2_{within}(T). \qquad (0.1)$$

Above equation can be simplified like this:

$$\begin{aligned}
\text{Within Class Variance} \quad & \sigma^2_W = W_b\,\sigma^2_b + W_f\,\sigma^2_f \quad \text{(as seen above)} \\
\text{Between Class Variance} \quad & \sigma^2_B = \sigma^2 - \sigma^2_W \\
& = W_b(\mu_b - \mu)^2 + W_f(\mu_f - \mu)^2 \quad (\text{where } \mu = W_b\,\mu_b + W_f\,\mu_f) \\
& = W_b\,W_f\,(\mu_b - \mu_f)^2
\end{aligned}$$

Therefore, I applied this simplified form in my code.

```
#For each threshold value calc_variance function is called and I try to find minimum within variance
threshold = 0
max_between_class =  0
between_class = 0;
for i in range(0,256):

    #Calculate weight, mean and variance for both background and foreground
    Wb, Mb, Vb = calc_variance (0,i+1)
    Wf, Mf, Vf = calc_variance (i+1,256)

    #Calculates between variance, and try to find maximum one to find optimum threshold
    between_class = Wb * Wf * (Mb - Mf)**2
    if (between_class > max_between_class):
        max_between_class = between_class
        threshold = i
```

Figure-11: Finding Variance

For each threshold value, I took weight, mean and variance values, found between class variance and compare it with the last maximum between variance. Threshold which gives maximum between class variance is the optimum threshold. Then I applied this threshold to the image and also printed both thresholds which are founded by myself and the existing function. They give same result.

```
#Apply the threshold to image
for y in range(row):
    for x in range(col):
        # threshold the pixel
        pixel = image[y, x]
        output1[y, x] = 0 if pixel < threshold else pixel

#Also by using existing threshold methold find optimum one
threshold2, output2 = cv2.threshold(image,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
#Print boths and compare them
print(threshold, threshold2)
print(max_between_class)
```

Figure-12: Comparing both thresholds

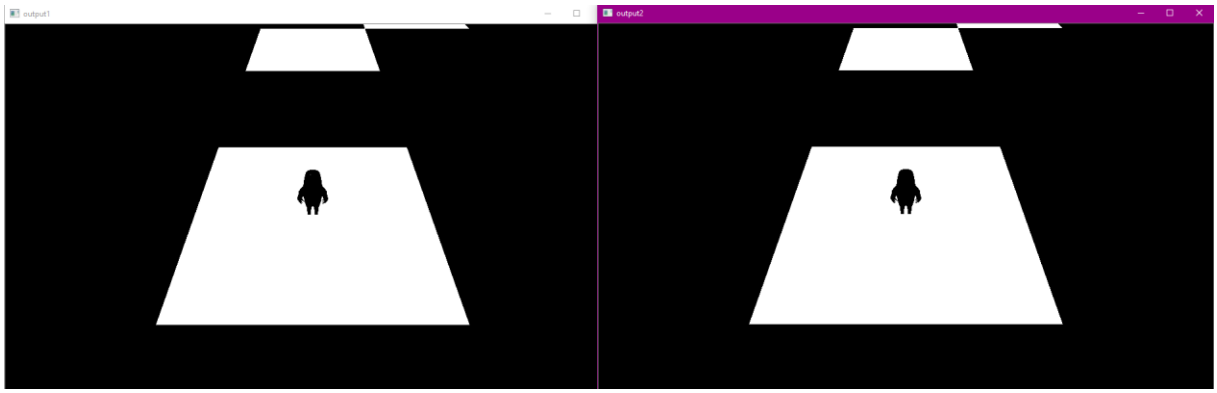

Figure-13: Outputs (my output and function's output)