Amirkabir University of Technology
(Tehran Polytechnic)
Department of Computer Engineering

Computer Vision

Assignment II

Soroush Mahdi
99131050

supervised by
Dr. Reza Safabakhsh

November 19, 2021

Contents

1 sudoku

To perform Hough Line transformation, edges of the image should be extracted first, therefor an edge detection is performed using the Canny algorithm. Then edges are given to both HoughLines and HoughLinesP functions, and the output lines are drawn on the original image.

```python
#reading image
img = cv2.imread('sudoku.jpg')
#making a copy of main image
img_p = img.copy()
#changing image to grayscale
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#performing edge detector on image
edges = cv2.Canny(gray,50,150,apertureSize=3)
#performing Probabilistic Hough Transform
lines = cv2.HoughLinesP(edges, 1, np.pi/180, 150, minLineLength=10, maxLineGap=250)
#drawing lines
for x1, y1, x2, y2 in lines[:,0]:
    cv2.line(img_p, (x1, y1), (x2, y2), (0, 0, 255), 3)

#performing Hough Transform
lines = cv2.HoughLines(edges,1,np.pi/180,200)
#drawing lines
for rho,theta in lines[:,0]:
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))

    cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)
#saving images
cv2.imwrite('houghlinesP.jpg',img_p)
cv2.imwrite('houghlines.jpg',img)
```

Listing 1: Code for Question 1

Results of the Hough Line Transformations are depicted below.

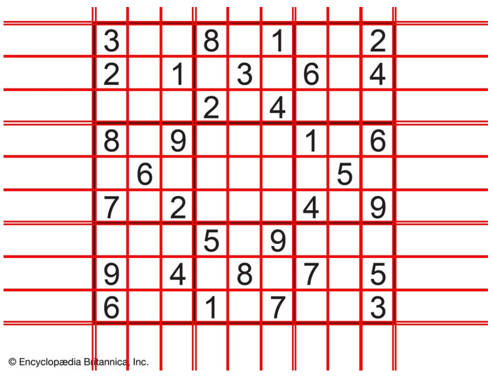
Figure 1: HoughLines function

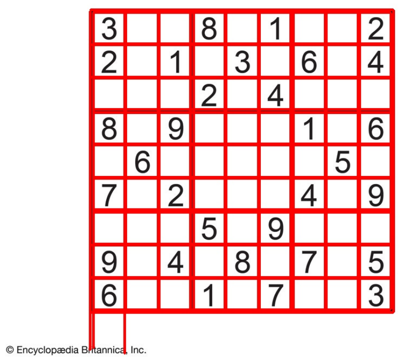
Figure 2: HoughLinesP function

the difference between Probabilistic Hough Transform and Hough Transform is that in the probabilistic version we don't consider all points instead we take only a random subset of points and that is sufficient for line detection. we should just decrease the threshold.

Explanation of parameters of HoughLineP function:

- rho: The resolution parameter rho in pixels
- theta: The resolution of the parameter theta in radians
- threshold: minimum number of intersecting points to detect a line

in this algorithm, we have to gather information for different values of theta and rho to decide if there exists a line with any specified theta and rho or not. theta values can be in the range of -180 to 180 degrees and rho can be in the range of zero to the length of the maximum side of the photo. but we cant consider these parameters continuous in implementation so we have to choose a resolution for them. for example, when we say theta resolution is pi/180 it means for theta we consider values to be integers between -180 and 180 degrees.

we save information for different values of theta and rho in a 2D array. every cell in this array corresponds to unique values of theta and rho. for a specified cell, we will increase its value if there exists a pixel that the edge detector has detected and the pixel is on the line corresponding to this cell. so cells with greater values will have better chances to tell us that there exists a line corresponding to them in the main photo. the threshold value will tell the algorithm what is the minimum value for a cell to be considered as a line in the main photo.

2 Detecting Coins!

```python
#creating an object of videoCapture for reading video
cap = cv2.VideoCapture('coins.mp4')

fourcc = cv2.VideoWriter_fourcc('M','J','P','G')
#creating an object of videoWriter for writing video
out = cv2.VideoWriter('output.avi',fourcc, 30, (int(cap.get(3)),int(cap.get(4))))
#reading video frame by frame and proccessing hough transformation
while(cap.isOpened()):
    #reading a frame from video
    ret, frame = cap.read()
    if ret: #if frame loaded seccessfully
        #turning readed fram to grayscale
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        #deniosing frame
        img_blur = cv2.medianBlur(gray, 13)
        # Apply hough transform on the image
        circles = cv2.HoughCircles(img_blur, cv2.HOUGH_GRADIENT, 1, img_blur.shape
    [0]/64, param1=275, param2=15, minRadius=30, maxRadius=35)
        # Draw detected circles
        if circles is not None:
            circles = np.uint16(np.around(circles)) #rounding circles values
            for i in circles[0, :]:
                cv2.circle(frame, (i[0], i[1]), i[2], (255, 0, 255), 3) #drawing
    circle on main frame

        #writing images to output video
        out.write(frame)
```

Listing 2: Code for Question 2

in this section first, I tried different settings for blurring frames so that the hough transformation can only detect coins and not the background text.

then I used HoughCircles function from opencv library to detect circles. this function uses the gradient information of edges (cv2.HOUGH-GRADIENT) and has an inner canny edge detector. the third function argument is The inverse ratio of resolution. The next parameter is Minimum distance between detected centers. param-1 is the Upper threshold for the internal Canny edge detector and param-2 is Threshold for center detection. The two next parameters are minimum and muximum radiuses for circles to detect.

now based on our problem we can use the fact that the radius of circles are fixed and equal for different coins so we can bound min and max radius parameters. I found this bound with trial and error. this bound can help the detector to be faster and more accurate.

now by blurring frames effectively the inner edge detector can easily detect coin edges without detecting background text edges. so the input edge image for hough transformation is ready. next by bounding the radius of circles, we can make hough transformation faster. in fact, we can say the speed of our code is acceptable for real-time applications.

bounding radius can also help hough transformation to perform more accurately by reducing the size of search space. we can see that our code can detect all coins in all frames very accurately.

3 OpenCv does not have Elliptical Hough Transformation!

in this section, I wrote this function to detect and draw an ellipse.

```python
def ellipse(image_rgb, edges, count=1 , acc = 1 , thr = 5):
    #performin hough transformation
    result = hough_ellipse(edges ,accuracy=acc , threshold=thr)#, min_size=5, max_size=35)
    #sorting based on accumulator
    result.sort(order='accumulator')
    # After finding all the possible ellipses in the the image using the Hough transform,
    # they are sorted based on multiple criterion for the most plausible ellipse to be found. These criterion are:
    # * Accumulation
    # * Deviance from the 90
    # * Deviance from ideal face height and width
    # * Deviance from the ideal area
    result = sorted(list(result),
                    key=lambda x: \
                    -x[0]/20 + abs(x[4]-1.5*x[3])/3 + abs(x[5]-np.pi/2)/.6 + \
                    (abs(image_rgb.shape[1]/5-x[4]) + abs(image_rgb.shape[0]/5-x[3]))/3 #+ \
                    #abs((image_rgb.shape[0]*image_rgb.shape[1])/(x[3]*x[4]) - 4)
                    )

    edges = color.gray2rgb(img_as_ubyte(edges))
    #drawing ellipse
    for res in result[:min(len(result), count)]:
        res = list(res)
        orientation = res[5]
        yc, xc, a, b = [int(round(x)) for x in res[1:5]]

        cy, cx = ellipse_perimeter(yc, xc, a, b, orientation, shape=image_rgb.shape)
        image_rgb[cy, cx] = (255, 0, 0)
        edges[cy, cx] = (250, 0, 0)
    return image_rgb, edges
```

Listing 3: ellipse detection function

in this function, i first used hough ellipse function from skimage. this function takes the edge map of the image as input. other parameters of this function are accuracy which is Bin size on the minor axis used in the accumulator and threshold which is the Accumulator threshold value for detecting ellipses.

he main part of this function is sorting the result of hough-ellipse function. After finding all the possible ellipses in the image using the Hough transform, they are sorted based on multiple criteria for the most plausible ellipse to be found. first, we sort it based on values of the accumulator, then other criteria are:

- Accumulation
- Deviance from the 90 degree
- Deviance from ideal face height and width
- Deviance from the ideal area

next i performed ellipse detection on webcam, the Code is on the next page.

```python
while(True):
    ret, frame = cap.read()

    if ret: #if frame has fecthed correctly
        #changing frame to rgb format for working with skimage
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        #performing gaussian pyramid
        image_rgb = tuple(pyramids.pyramid_gaussian(image, 3, multichannel=1))[-1]
        #turning image to gray scale and performin canny
        image_gray = color.rgb2gray(image_rgb)
        edges = canny(image_gray, sigma=1.5, low_threshold=0.2, high_threshold=0.8)
        #perforimng ellipse detection
        image_rgb, edges = ellipse(image_rgb, edges, acc = 20)
        #changing format to bgr
        final = image_rgb[:,:,::-1]
        final = cv2.resize(final, ( 640 , 480))
        cv2.imshow('ellipse',final )

    cv2.imshow('frame', frame)


    if cv2.waitKey(1) == 27: #waiting 30 miliseconds
        # ESC pressed
        print("Escape hit, closing...")
        break
```

Listing 4: ellipse detection on webcam

in this code first, I read the input frames on a loop, then change the format of the frame to rgb for working with skimage. the main part is downscaling image resolution using Gaussian pyramids. I did this because performing hough transformation on the main image was not practical and we know that hough transformation computation time is dependent on the number of parameters for detecting the target shape and resolution of the input image. so I reduced image resolution for improving computation time. I tried different levels of the pyramid and the level that has good accuracy and time is 3. then I change the image to grayscale and perform canny for edge detection. and finally, I give the edge map to the ellipse function.

here are some examples of running code on my webcam with different rotations and scales. with level 3 of pyramid we can work real time in our problem with an core i7 cpu.

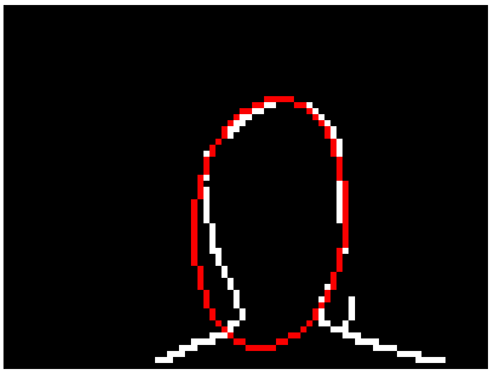
Figure 3: result for level 3 of Gaussian pyramids.

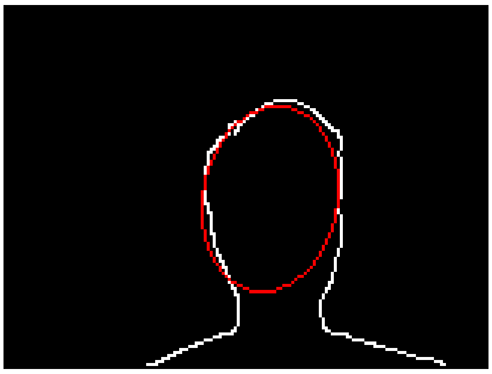
Figure 4: result for level 2 of Gaussian pyramids.

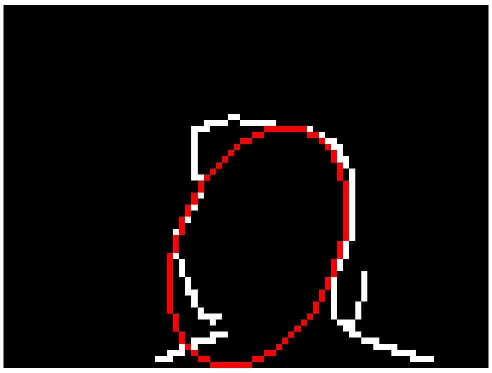
Figure 5: result for level 3 of Gaussian pyramids.

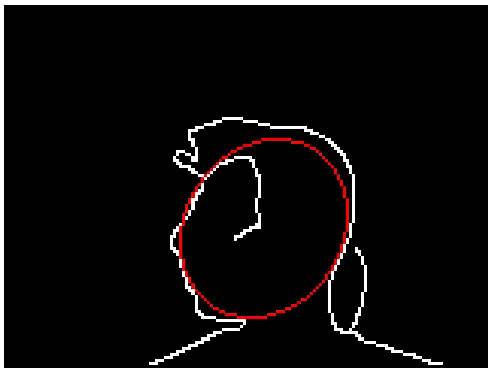
Figure 6: result for level 2 of Gaussian pyramids.

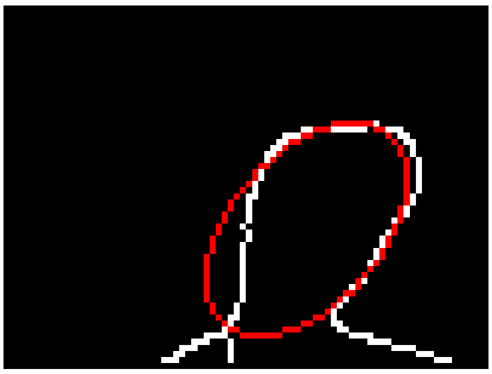
Figure 7: result for level 3 of Gaussian pyramids.

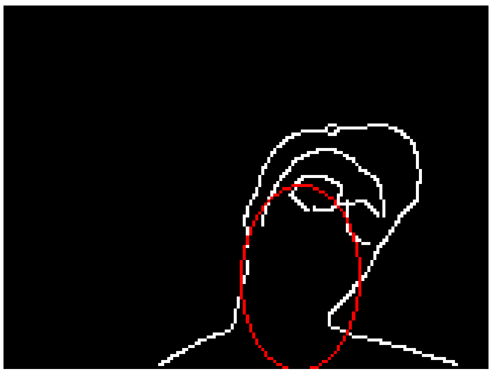
Figure 8: result for level 2 of Gaussian pyramids.

4 How to improve Elliptical Hough Transformation?

Since there is a myriad of ellipses in every frame, it's impossible to detect a face only using an ellipse detector algorithm. Therefore the algorithm needs to be biased towards the face-like ellipses. And the more the algorithm is biased, the more unreliable it is against rotation and scale. So a critical part of the task is to find the best point in this trade-off. Also another way of dealing with this problem is to use biases that aren't related to rotation and scale, such as location of nose and eyes. for examples we can detect eyes using hough transformation for circles. hough transformation for circles is fast and is robust again rotation.