AI 825 / Visual Recognition 13-02-22

Assignment 1

Instructor: Prof. Dinesh Babu

Shaikh Fahed IMT2019079

Report on code for Assignment 1

1 Image Processing

Made a function ***Processing(image)*** which makes the given image gray, blur and canny using the respective functions. This functions makes it easier to detect the lines and remove any disturbances if present. They enhance our accuracy.

```
def processing(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    canny = cv2.Canny(blur, 50,150)
    return canny
```

2 Masking

Now, I plotted the processed image into a matplotlib plot and **noted down the coordinates within which the road lies**, to remove all the unwanted disturbance out. We mask the image with black colour wherever there is no road. This removes the unwanted noise due to mountains and trees and other side objects. The function ***Masking(image)*** cuts down the area into two parts, road and non-road with the help of the coordinates the plot of the processed image gives. The coordinates give the polygon of the road, then with help of ***fillPoly()***, we fill the road part with white. It then uses the ***bitwise_and()*** function to return the Masked Image.

.

.

```
def Masking(image):
    height = image.shape[0]
    polygons = np.array([
        [(0, 60), (0, height), (594, height), (594, 70), (310, 0)]
        ])
    mask = np.zeros_like(image)
    cv2.fillPoly(mask, polygons, 255)
    maskedImage = cv2.bitwise_and(image, mask)
    return maskedImage
```

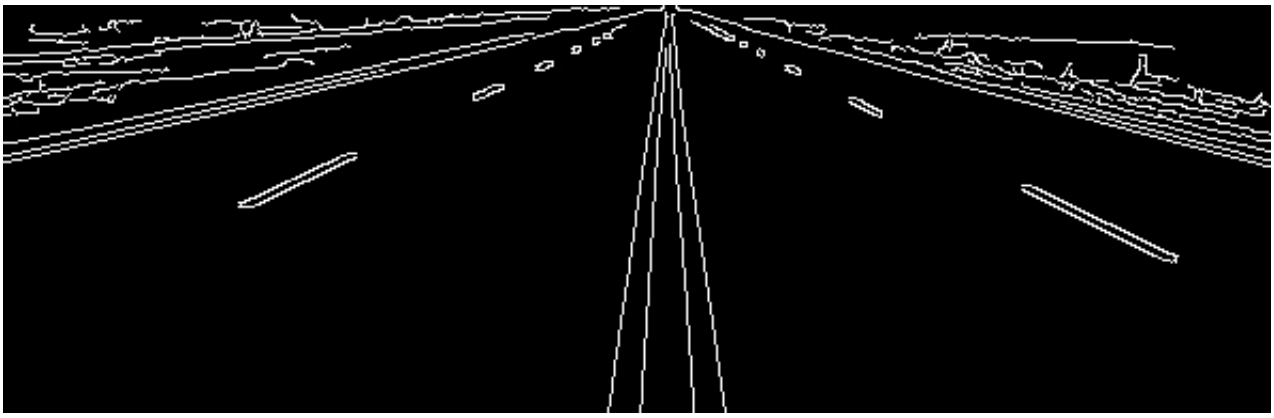
3 Main Code for Lane Detection

.

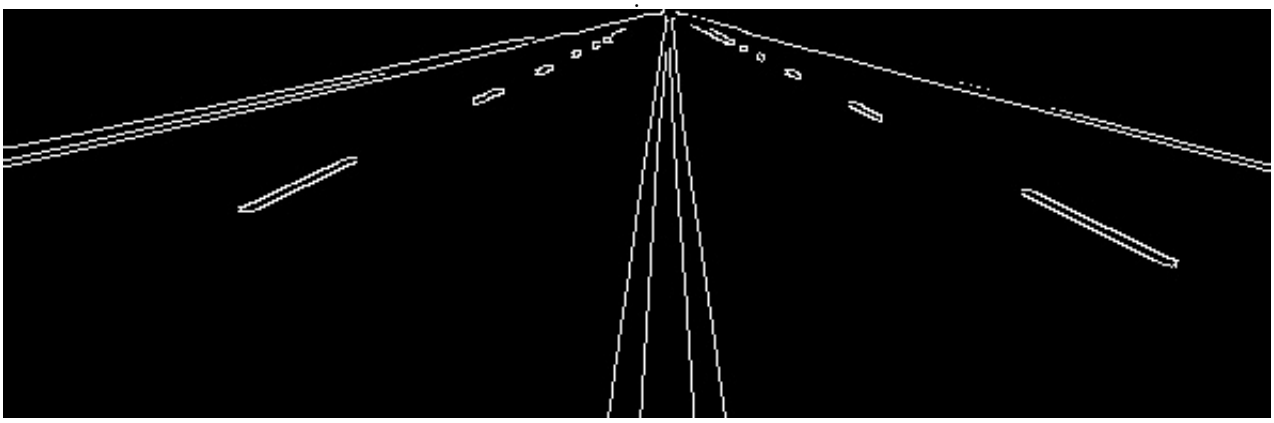
```
#PART1, LANE DETECTION
lanesimage = cv2.imread('lanes.jpg')
finalL = processing(lanesimage)
croppedImage = Masking(finalL)
lanes = HoughLinesp(croppedImage, lanesimage, 50)

cv2.imshow("Final Lanes", lanes)
# cv2.imwrite('Final Lanes.jpg', lanes)
```

.
.
We read the image and process it, as shown below, first is the initial image, we crop it and process it, that is, we gray is, blur it and make it a canny image as shown in the second picture. Then, with the help of a function we developed, we can clear of the disturbance due to the other particles and focus mainly on the road to detect the lanes. The Masking function helps us remove the lines formed due to the mountains, grass and fences etc present in the initial image. .
.

.
.



The upper one is just a canny image of original image and the below attached is the masked image, you can notice the difference between these two pictures. Which is notably due to the Masking function.



After all this processing ang masking, we called the function ***HoughLinesp(croppedImage, lanesimage, 50)*** which returns the original image with the lines detected drawn onto it.

4 Implementation of *HoughLinesp(croppedImage, lanesimage, 50)*

.

```
#hough line implementation algortihm.
def HoughLinesp(CannyImage,image,threshold):
    image1 = image.copy()
    width = CannyImage.shape[1]
    height = CannyImage.shape[0]
    Thetas = np.deg2rad(np.arange(0.0, 180.0))

    #Initializing the Accumulator.
    DiagonalLength = int(np.around(np.sqrt(width*width + height*height),0))
    Accumulator = np.zeros((2*DiagonalLength,len(Thetas)))
    Rhos = np.linspace(-DiagonalLength,DiagonalLength,2*DiagonalLength)

    for i in range(0,height):
        for j in range(0,width):
            for k in range(0,len(Thetas)):
                Rho = int(np.around(i*(np.cos(Thetas[k])) + j*(np.sin(Thetas[k])),0))
                Accumulator[Rho+DiagonalLength,k] += 1
```

.

We take the inputs of the processed image, the actual image, and a value for thresehold, to limit the accumulator. We declared a few useful variables and we also made ***Thetas*** which holds our values for θ. Initiated the Diagonal length, followed by the Accumulator. Made ***Rhos*** to hold the value of ρ. Then we run an iterative loop to fill the Accumulator and begin the voting.

.
.
.
.

```
linesList = []
#Thresholding the votes.
for i in range(0,Accumulator.shape[0]):
    for j in range(0,Accumulator.shape[1]):
        if Accumulator[i,j] > threshold:
            linesList.append([Rhos[i],Thetas[j]])
```

.

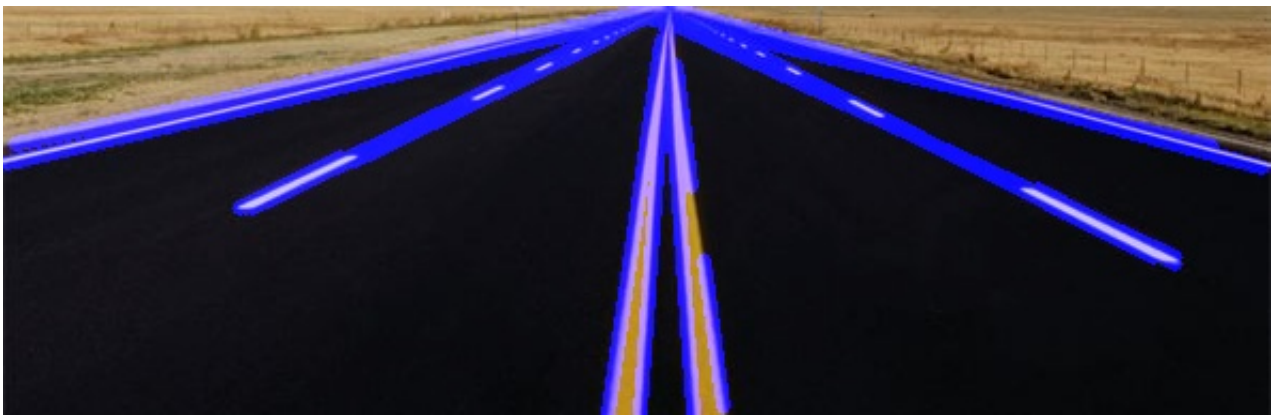
Here is an empty linesList which holds all the lines possible in the image, Now that we iterate throught the Accumulator to Threshold the voting, we append the lines in our list.

```
#Plotting the lines on given Image.
for i in range(0,len(linesList)):
    for i in range(0, len(linesList)):
        a = np.cos(np.deg2rad(linesList[i][1]))
        b = np.sin(np.deg2rad(linesList[i][1]))
        x0 = a*linesList[i][0]
        y0 = b*linesList[i][0]
        x1 = int(x0 + 1000*(-b))
        y1 = int(y0 + 1000*(a))
        x2 = int(x0 - 1000*(-b))
        y2 = int(y0 - 1000*(a))

        cv2.line(image1, (x1, y1), (x2, y2), (255, 0, 0), 5)

return image1
```

.

Now that we have our linesList filled, we plot the lines in the original image, using the ***cv2.line*** , function which takes the inputs of image, any two parametrical points and the color in which we need the detected lines to be, and the respective thickness, we have our image ready to be returned and shown in the main part of the code with the road lanes detected as below.

.
.
.
.



5 Main Code for Coin Detection

.
.

```
#PART1, COIN DETECTION
coinsimage = cv2.imread('coins.jpg')
finalC = processing(coinsimage)
coins = HoughCircle(finalC, coinsimage, 15)

cv2.imshow("Final Coins", coins)
# cv2.imwrite('Final Coins.jpg', coins)



cv2.waitKey(0)
cv2.destroyAllWindows
```

.
.

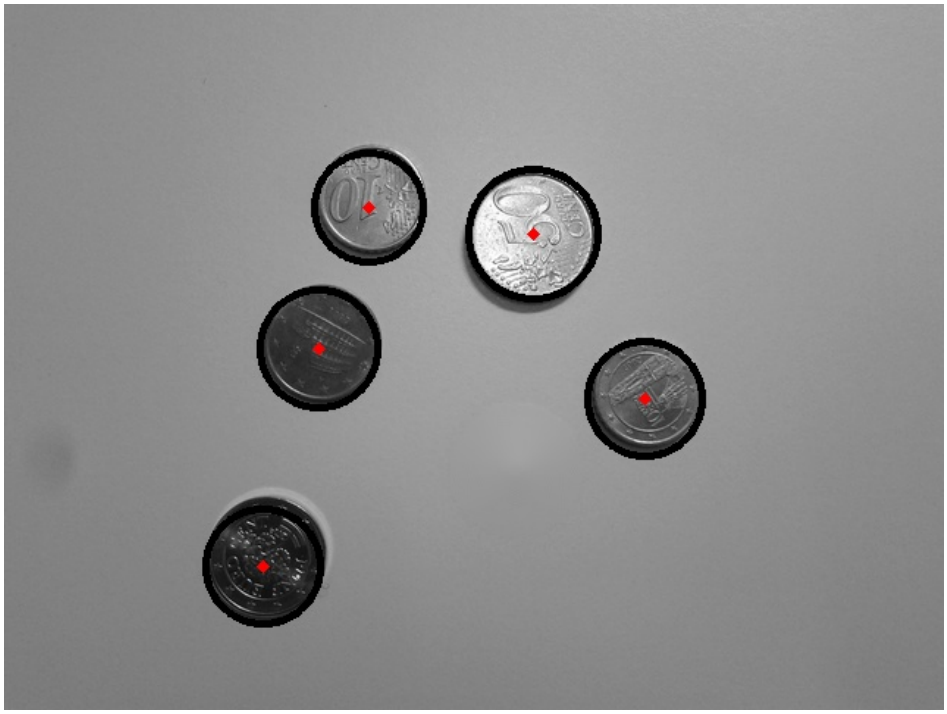
Just like the previous part, we read the image and process it, to be noted that there is no need of masking in this part. Attached below is the original image vs the processed image, we don't need to canny this image because the edges are clearly visible and there is no potential disturbance to the boundary of coins since the background is white. .

.
.

.
.
Blurring the image is important because we may face some distrubances due to the inner design or shadows of the circles.

Now I used a function ***HoughCircle(finalC, coinsimage, 15)*** which returns the image with the circles detected and highlighted as shown below, We will discuss how this function is implemented and all the necessary parameters ahead.

.
.



6 Implementation of *HoughCircle(finalC, coinsimage, 15)*

.

```
#Hough circle implementation.
def HoughCircle(CannyImage,image,radius):
    image1 = image.copy()
    width  = CannyImage.shape[1]
    height = CannyImage.shape[0]
    Thetas = np.deg2rad(np.arange(0.0, 180.0))

    #Initializing the Accumulator.
    list1 = []
    list2 = []
    accumulator = np.zeros((2*width,2*width))
    for i in range(0,height):
        for j in range(0,width):
            for k in range(0,len(Thetas)):
                a = int(np.around(i - radius*np.cos(Thetas[k]),0))
                b = int(np.around(j - radius*np.sin(Thetas[k]),0))
                list1.append(a)
                list2.append(b)
                accumulator[a+width,b+width] += 1
```

.

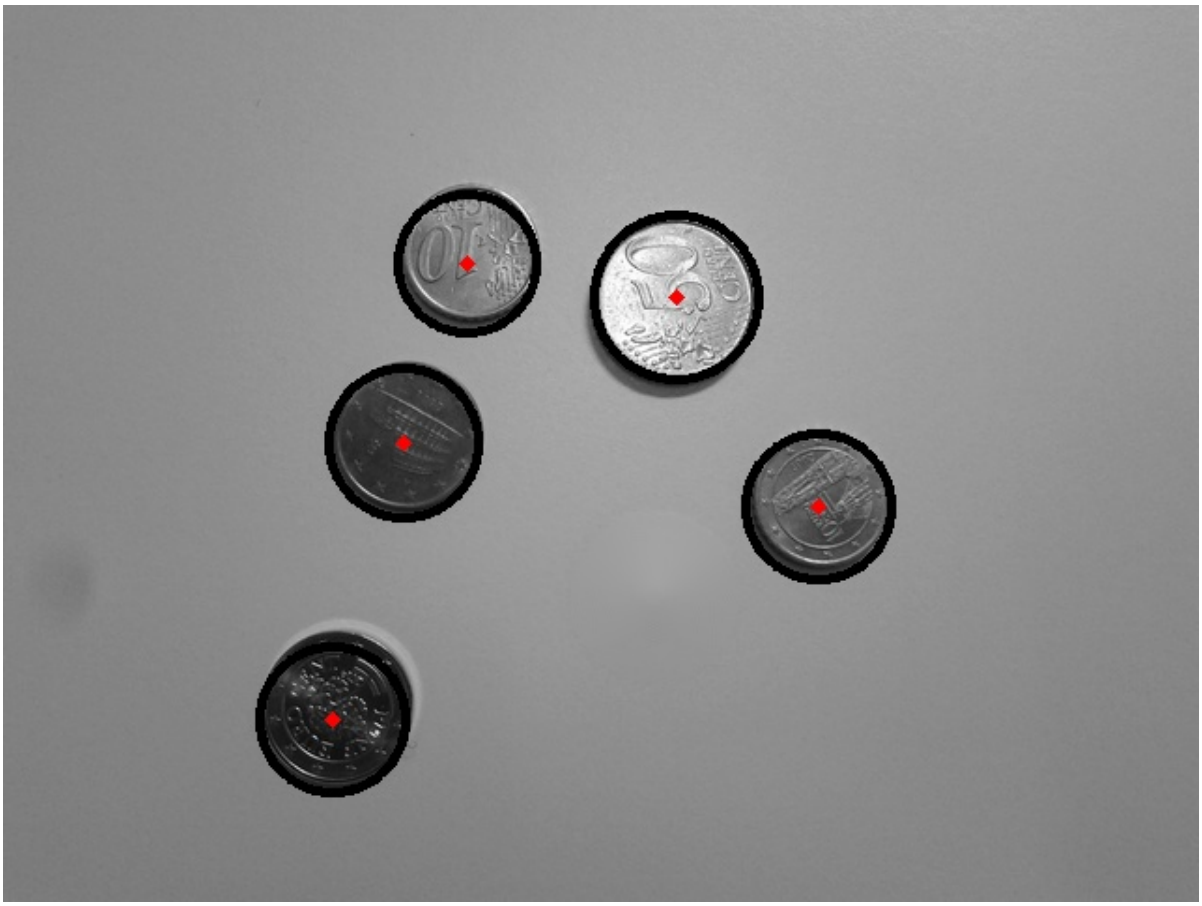
We declared a few important terms and initiated the accumulator, we made two lists to fill the accumulator. .

```
circle = []
#Thresholding the votes.
for i in range(0,accumulator.shape[0]):
    for j in range(0,accumulator.shape[1]):
        if accumulator[i,j] > 96:
            circle.append([i-width,j-width])

#Plotting the circles on given Image.
for center in circle:
    temp = center
    x = temp[0]
    y = temp[1]
    cv2.circle(image1, (x, y), radius, (0, 0, 0), 3)
    cv2.circle(image1, (x, y), 2, (0, 0, 255), 3)

return image1
```

.

We made an empty list circle to accommodate all the detected circles and then we threshold the accumulator iterating through it, and we append everytime we encounter a circle. We have given the threshold radius as a parameter to the function, and hence, with every center of the circle, we can plot a circle around it with the help of ***cv2.circle*** since we know the radius, and we put the color black as a boundary and place a red dot as the center.

.



.

Therefore, without using the library functions, We have successfully traced both lines in lanes and circles in coins images with self developed functions.