Edge and Circle Detection Using Hough Transform

Introduction

We were given the following two tasks which were to be done using the Hough Transform implemented from scratch:

a. Lane Detection: To identify lanes on a road in a given image (Straight-line detection)
b. Coin Detection: To identify the boundary of coins in an image (Circle detection)

Hough Transform

Hough Transform is an algorithm used to identify edges in an image of any general shape, though this assignment only assumes that the shapes to be detected are a circle and line. Given an image containing only the points on edges of various objects in the image (obtained by canny edge detection), the hough transform iterates through every edge point in the image and determines the boundary line of an object by calculating parameters of the locus in which this point would lie on. Then, the algorithm decides, by voting, the locus corresponding to parameters that are obtained by the maximum number of points, which is the boundary of certain objects in an image.

Lane Detection by Hough Transform

Assumptions

I assumed the following while working on the assignment:

1. The image provided contains a uniform road with no potholes or other deformities.
2. The user of the hough_line() function provides a threshold value, which determines the lines being considered.

Implementation Details

Preprocessing

I did the following preprocessing on the image:

1. Cropping the image
   This is done to remove the part which is not a road. On including such parts I found that different levels of blurring were required for the road and the part which is not a road.
   The latter required more blurring since we do not want any line to be detected here and hence, this part shouldn't have an appearance in the canny edge detection image output.

According to my assumption, the former did not require any blurring as the road only contained lane markings.
Hence, the program is only focused on parts having lanes, in order to perform effective lane detection.

2. Grayscaling the image
   I also explored color spaces like HSV or Lab. However, since the road is generally dark while the lanes are marked with white, there was enough contrast in the colors of the features I wished to focus on. Hence, such color spaces did not provide much advantage in my implementation.
3. Thresholding the image
4. Canny Edge Detection
   This step would give us the boundary points of the lanes on the road, which would be further used by the Hough Transform Algorithm to detect the boundary lines of the lanes. Hence, detecting the lanes

For all the above steps, I used built-in functions from OpenCV and collected all the steps in a function: preprocess() which takes the cropped input image and returns the canny detection output of it.

Hough Transform algorithm for lines

This algorithm was used to identify the borderlines of the lanes in the image. Since the usual representation of the line given by:

$$y = mx + c$$

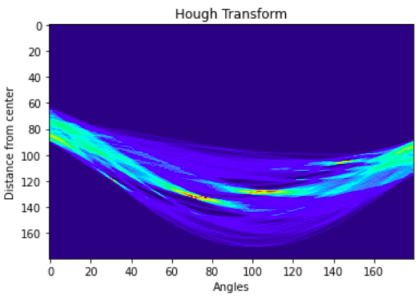
is hard to scale since m and c both can vary from -∞ to ∞, I used this representation of a line:

$$\rho = x\cos(\theta) + y\sin(\theta)$$

since, $\rho \in (-\ image\ diagonal,\ image\ diagonal)$, which is the distance of the origin from the line and $\Theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$, where image diagonal is given by:

$$image\ diagonal\ = \sqrt{(image\ width)^2\ +\ (image\ height)^2}$$

Hence, we maintain arrays of the two parameters and an accumulator with same number of rows as the values of ρ stored and same number of columns as values of θ stored.
We then iterate through the non zero pixels of the canny edge detected image, and calculate value of rho for every value of theta, since each value of theta denotes a separate line and hence would form another point in the hough space.
Then we find the index of a value of rho closest to the one calculated within the list of all rhos initialized. We then increment the accumulator at the rho index and the theta index.
Upon visualizing the contents of the accumulator, we get sinusoids which are the same as the ones formed in hough space as a result of using each point as a parameter in this space. We get a sinusoid as it's a curve between ρ and θ.



The slightly brighter regions indicate intersections of the sinusoids formed by various edge points in the image. Each intersection corresponds to a tuple: (rho, theta), which are the parameters of a line on which all the points, whose sinusoids intersect, lie.

Postprocessing

After applying the houghs transform algorithm, I converted the accumulator into a list of tuples with the value greater than the threshold and it's location i.e the row number and the column number.
The row number and the column number determines the values of rho and theta from the list initialized. These values are in turn used to calculate the x and y-intercepts of the line being identified in the image.
We then use the cv.lines() to draw the lines by using their x-intercept and y-intercept.

Results

The following are the results obtained on a few images I tried the algorithm on:

Coin Detection by Hough Transform

Assumptions

I made the following assumptions while working on the assignment:

1. The coordinates of the centre of the circle are discrete as they are pixel locations.
2. Radius value is assumed to be constant

Implementation details

Preprocessing

I did the following preprocessing on the image:

1. Convert to Grayscale
2. Gaussian blur on the image with kernel size of (19, 19)
   While detecting the boundary of the coin, the inscription of the coin just contributes to noise and doesn't add to the clarity. Hence, in order to completely remove the coin inscriptions, such large magnitude blurring is required.
3. Thresholding the image (Binarizing it)
4. Performing canny edge detection on it.

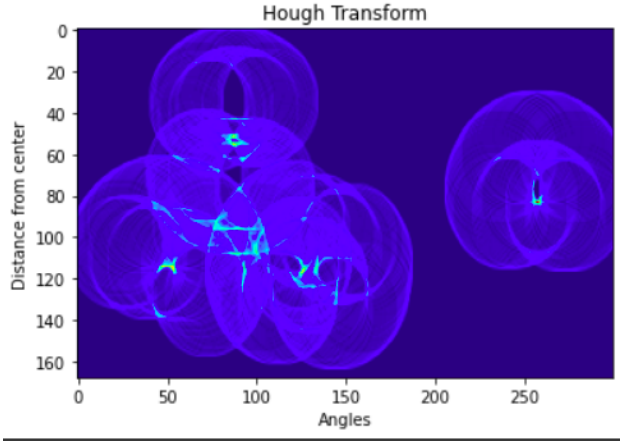
For all the above steps, I used built-in functions from OpenCV and collected all the steps in a function: preprocess() which takes the input image path and returns the canny detection output of it.

Hough Transform algorithm for circles

Since we have assumed the radius of the coin (i.e circle to be detected) to be constant, there are only two parameters in the circle equation that we need to find: the x and y coordinates of the circle (a and b):

$$(x - a)^2 + (y - b)^2 = r^2$$

We know that a point in the image space results in a circle in the hough space i.e the point becomes the center of the circle which contains all the possible points which are 2-tuples storing the coordinates of the centers of all possible circles in which the point would lie on. An intersection of these circles gives the center of the required circle in the image space. Also, while visualizing the accumulator I observed that, the circles intersecting form a hollow circle in the hough space if the radius provided isn't right. As we go closer to the optimum value of the radius, this hollow circle converges to an intersection point.
In the diagram below, the bright spots indicate intersections of the circles:

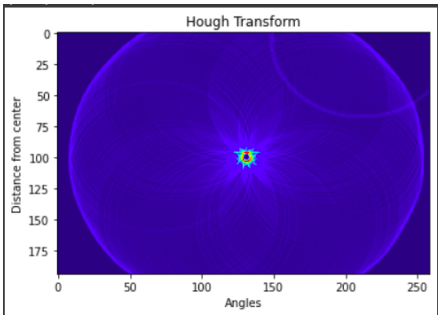


Postprocessing

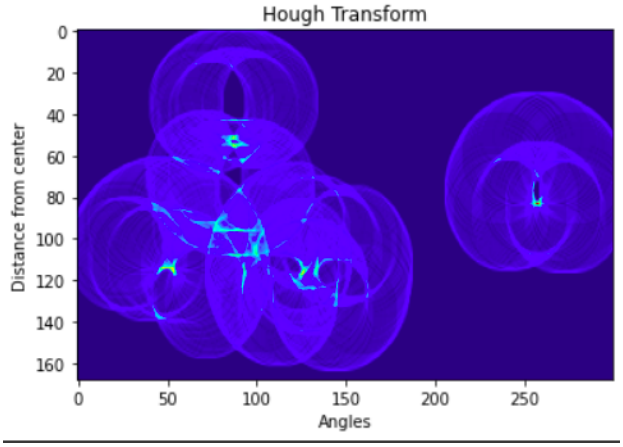
The indices at which the accumulator stores the values are nothing but the coordinates of the center of the circle. We, hence, simply pass the coordinates to the cv.circle() to draw the circle on the image, to detect a coin.

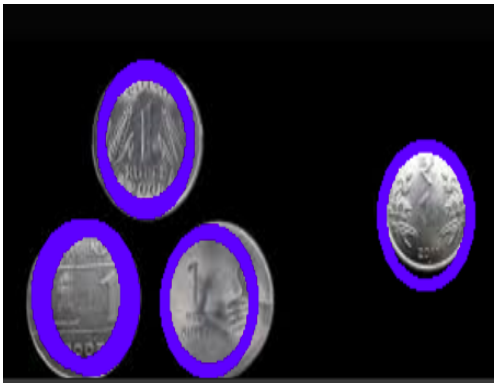
Results

1. Radius of 63 units



2. Radius of 17 units





3. Radius of 17 units