Traditional Lines and Circles detector

By

Jimut Bahan Pal
B1930050
Semester - I

Under the guidance
of

Tamal Maharaj

Submitted to the Department of Computer Science in
partial fulfilment of the requirements
for the degree of M.Sc.

Ramakrishna Mission Vivekananda Educational and Research Institute
Howrah - 711202
February, 2020

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, "Traditional Lines and Circles detector", is the outcome of the investigation and research carried out by me under the supervision of Tamal Maharaj.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Jimut Bahan Pal
B1930050

CERTIFICATION

This thesis titled, **"Traditional Lines and Circles detector"**, submitted by Jimut Bahan Pal as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree M.Sc. in Computer Science in February, 2020.

Group Members:

Jimut Bahan Pal
jimutbahanpal@yahoo.com

Supervisor:

Tamal Maharaj
Ph.D. from University at Buffalo NY USA
Department of Computer Science
Ramakrishna Mission Vivekananda Educational and Research Institute

ACKNOWLEDGEMENT

It is ritual that scholars express their gratitude to their supervisors. This acknowledgement is very special to me to express my deepest sense of gratitude and pay respect to my supervisor, Tamal Maharaj, Department of Computer Science, for his constant encouragement, guidance, supervision, and support throughout the completion of my project. His close scrutiny, constructive criticism, and intellectual insight have immensely helped me in every stage of my work. I also acknowledge the help received from Dr. Donald Knuth, Turing award winner, for giving me a digital permission to use his collected diamond street sign images. The four images are copyrighted by him.

I'm grateful to my father, Dr. Jadab Kumar Pal, Deputy Chief Executive, Indian Statistical Institute, Kolkata for constantly motivating and supporting me to develop this documentation along with the application. Finally, I acknowledge the help received from all my friends and well-wishers whose constant motivation has promoted the completion of this project.

Kolkata
February, 2020

Jimut Bahan Pal

Contents

List of Figures

ABSTRACT

The traditional method for extracting features from image is accomplished by Hough Transform. When an image is represented mathematically, it becomes easy to perform computation according to a certain algorithm and detect the required features from the image. The presence of such features is determined by a voting procedure in a parameter space, from which the selected features are obtained as local maxima by creating a certain threshold. This procedure for simple detection of lines was patented by Paul V.C. Hough in 1962. There has been various advancement in this field for detecting lines and circles, all of them are based on the parent Hough Transform way to vote for the corresponding features. The various ways of detecting features and the mathematics behind them is discussed in this report.

Chapter 1

Introduction

1.1 Why we need to detect features?

There is a need by various automation industries for performing certain decision by determining certain features [1] and complex patterns. Definition of feature depends on the problem type or the type [2] of application. The question is to determine whether a certain subset of image is present in another image, features are often in the form of continuous curves, straight line, corners or connected regions etc. This invention was originally patented by Paul V.C. Hough in 1962, for the study of subatomic particle tracks passing through a viewing field. This was initially used in the bubble chamber where they automated the study of viewing charged particles creating tracks along their path of travel of the size of 0.01 inch apart.

There are various algorithms in the field of computer vision which uses the feature as the starting point for performing additional computations. One of the important property of features is repeatability, i.e., whether the feature will be present in two orientations of the same scene from a different perspective. The most common operation done before feature detection is Gaussian smoothing [3] of the image. Features can be detected by either performing edge detection algorithms or by finding the derivative of the image. When the feature detection is to be performed in real time, then the algorithm may guide the feature detector to select certain parts of the image and work on them.

1.2 Basic types of image features

There are a few types of image features which are discussed below:

1.2.1 Corners

Corners are the point of image which has a very large gradient magnitude both in the x and y directions. Locally corners have 1 dimensional structure, i.e., they can be represented in terms of points. The neighborhood of an image helps to determine the features, since a single point could not determine the feature uniquely in an image.

1.2.2 Blobs

Blobs provide a complementary description of image features in contrast to corners, since it represents a certain region. Blobs are preferred for interest points, since they can detect areas in an image which are too smooth to be detected by a corner detector. When we are scaling the image up, then corners may become vague, since it will represent smooth curve, which in turn can be detected by blobs. Laplacian of Gaussian (LoG) and Difference of Gaussian (DoG) are the most commonly used blob detectors.

1.2.3 Ridges

The ridge detection in computer vision is similar to detecting geographical ridges. Ridges are a set of smooth function [4] of two variables. For a function with N variables, the ridges are a set of curves whose points are local maxima in N-1 dimensions. They are basically used to capture the interior of elongated objects in the image domain. They are used to capture blood vessels in medical images and road in aerial images.

Chapter 2

Hough Transform to detect lines

2.1 Theory

Hough Transform is a popular technique to detect shape [5], if we can represent that shape in mathematical form. It is robust to noise and occlusion. It even detects shape if it is broken or distorted a little bit. We are now going to see a very basic version of Hough transform to detect lines.

2.2 Mathematical form

A line can be represented as:

$$y = mx + c \tag{2.1}$$

In parametric form it can be represented as:

$$r = x\cos\theta + y\sin\theta \tag{2.2}$$

In Figure 2.1 we can see the parametric diagram of the line. Here r is the perpendicular distance from the origin to the line [6] and θ is the angle formed by this perpendicular line. Here, since the line is passing above the origin, the angle is taken to be greater than 180^o and r is taken to be negative. So, if the line is below origin, we will have a positive r and angle is less than 180^o.

It is therefore possible to associate each line of the image pair (r,θ). The matrix which collects this voting value is a member of Hough space for the set of straight lines in two dimensions. Given a single point in the plane all the lines passing through that point represents a sinusoidal curve in the (r,θ) plane which is unique to that point.

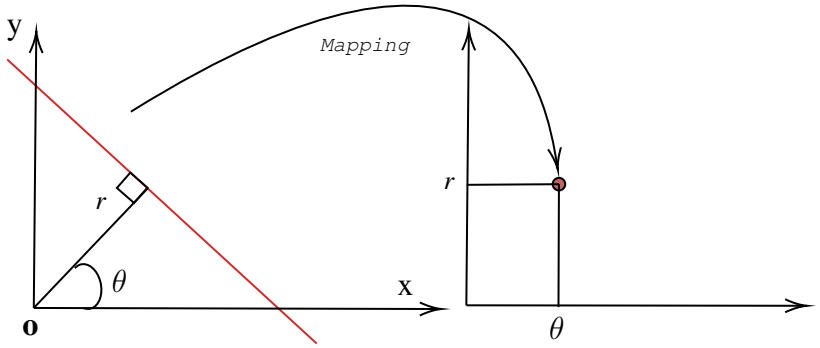


Figure 2.1: Hesse normal form of Hough transform mapped to Hough parameter space

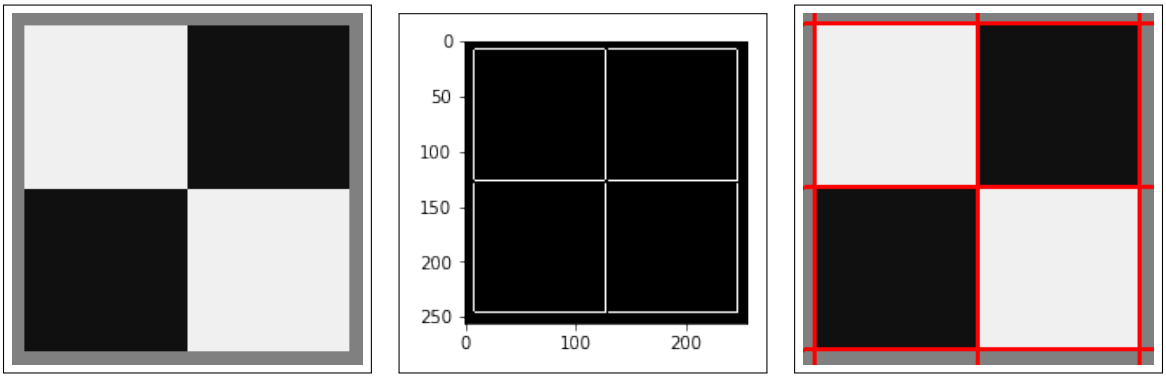


Figure 2.2: A figure containing the images by applying Hough Line transform. (Left: An original test image before applying Hough Line transform, Middle: After applying preprocessing to the first image, Right: Image after applying Hough transform results in detection of lines)

2.3 Our Implementation

When images are used in different areas of image analysis, it is important to reduce the amount of data in the image while preserving [7] the important, characteristic and structural information. When using edge detection, we are in turn reducing the size of the image considerably. The output of the edge detected version of the original image is still an image described by its pixels. If the lines, ellipses, circles, etc. are determined by their characteristic equations the amount of data would be reduced even more.

In theory, there will be several lines that would be formed in case if there is no thresholding value . The points where most of the edge points intersects, i.e., a peak in the accumulator cell, gives the true lines in the edge map. The resolution of the accumulator determines the precision with which the lines can be detected. Here the thickness of 1 is used for determining the thickness of the edge in the real picture.

We also note that several entrances on the true accumulator near the true edge point has a lot

(a) Accumulator matrix by applying Hough Line transform on the test image.

(b) Points selected by applying a threshold on the accumulator matrix on the test image.

Figure 2.3: A figure containing the images of accumulator matrix.

of similar values, so the size of the bin selection determines the precision of the line. We used the neighborhood parameter to tweak this value. The classical Hough transform gives the information for r and θ value which corresponds to an infinitely long line. We can tweak this value to determine the lines of finite length when drawing in the original image to reduce clutter.

The finite lines that are extracted by thresholding the value got from the accumulator is superimposed on the original image to produce bounding lines which determines the correctness of the algorithm in visual sense.

The algorithm for Hough Transform is,

1. Input image.
2. Detect the edge via edge detector algorithm
   (eg.: Canny Edge detector).
3. Select a threshold and convert the image to binary.
4. Initialize a 2D accumulator array.
5. For each pixel compute r and theta.
6. Store the value of r and theta in 2D accumulator array.
7. Pick a threshold to select the best points.

8. Plot the lines in the original image.

We have used Python3 as our tool for creating Hough transform, via NumPy library. We have converted the above steps into python3 code and implemented the Hough transform algorithm.

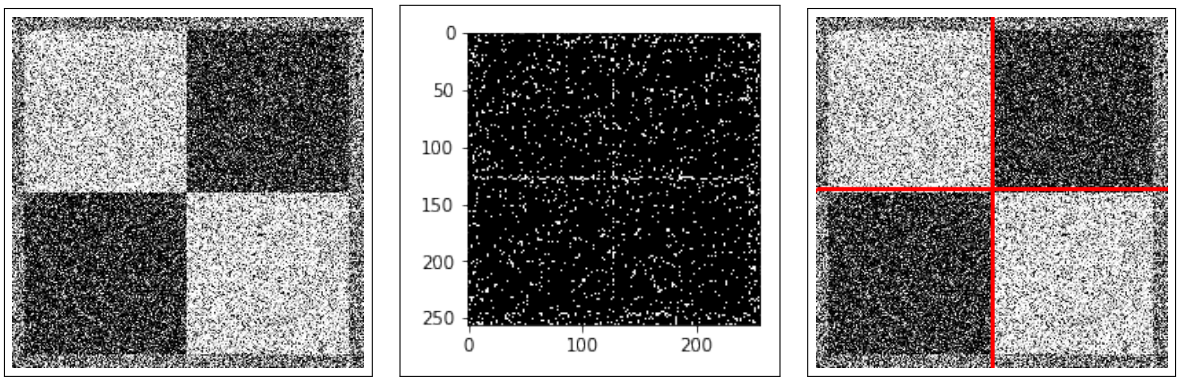


Figure 2.4: A figure containing the images by applying Hough Line transform on the noised version of the first image. (Left: An original noised version of the test image before applying Hough Line transform, Middle: Image after applying Hough transform results in detection of lines, Right: Image after applying Hough transform results in detection of lines)

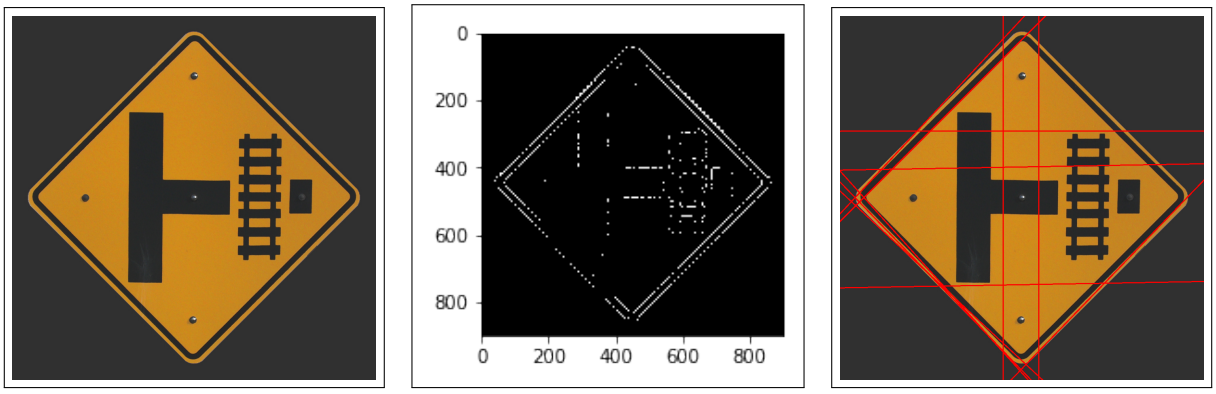


Figure 2.5: A figure containing the images by applying Hough Line transform. (Left: An original test image before applying Hough Line transform, Middle: After applying preprocessing to the first image, Right: Image after applying Hough Line transform results in detection of Lines). Picture of Diamond Sign is used with written permission from Donal Knuth.

We first take an input image as shown in Figure 2.2 (left). We then apply Hough transform according to the algorithm, the resulting image after the Hough transform Line detection algorithm finish executing is shown in Figure 2.2 (middle). The accumulator matrix corresponding to the given image is shown in Figure 2.3a. After applying the threshold on the accumulator matrix we get the selected points as line in Figure 2.3b. Here, we get six dots which corresponds to the 6 lines of the 6 corners.

Similarly, we do this for the noised version of the image without tweaking the parameters as given in the code. We first take an input image as shown in Figure 2.4 (left). The preprocessed image for detecting edge is achieved via image Figure 2.4 (middle). We then apply

(a) Accumulator matrix by applying Hough Line transform on the test image.

(b) Points selected by applying a threshold on the accumulator matrix on the test image.

Figure 2.6: A figure containing the images by applying Hough Line transform on the noised version of the first image.

Hough transform according to the algorithm, the resulting image after the Hough transform Line detection algorithm finish executing is shown in Figure 2.4 (right). The accumulator matrix corresponding to the given image is shown in Figure 2.6a. After applying the threshold on the accumulator matrix we get the selected points as line in Figure 2.6b.

We can clearly see that the algorithm detects 50% of the lines even if there is occlusion and tremendous noise, this is achieved because of the fact that the voting is done by each pixel on the image. We are now convinced that Hough transform to detect lines is a really good algorithm even if this is of simple nature.

We implement the same for some different image of diamond size which has different lines. The size of the image is big, so we need to do some change in the parameters of the preprocessing which will help us to detect lines. The original image is shown in fig. 2.7 (left) and fig. 2.5 (left) respectively. The preprocessed version of them are shown in fig. 2.7 (middle) and fig. 2.5 (middle) respectively. The final images which are got by line detection algorithm are shown in fig. 2.7 (right) and fig. 2.5 (right) respectively. We also plot the parameter space of the accumulator matrix to just give a sense how the thresholding is done from the matrix in fig. 2.8.

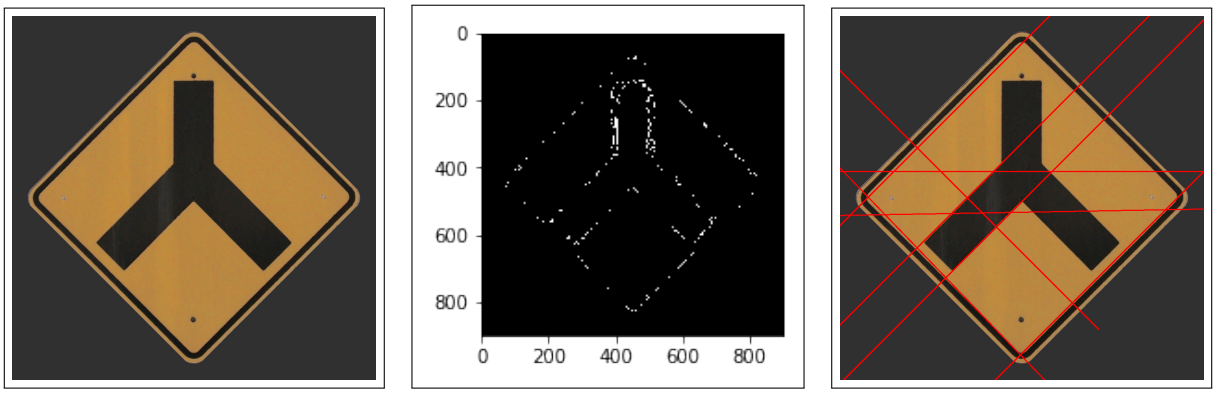


Figure 2.7: A figure containing the images by applying Hough Line transform. (Left: An original test image before applying Hough Line transform, Middle: After applying preprocessing to the first image, Right: Image after applying Hough Line transform results in detection of Line). Picture of Diamond Sign is used with written permission from Donal Knuth.

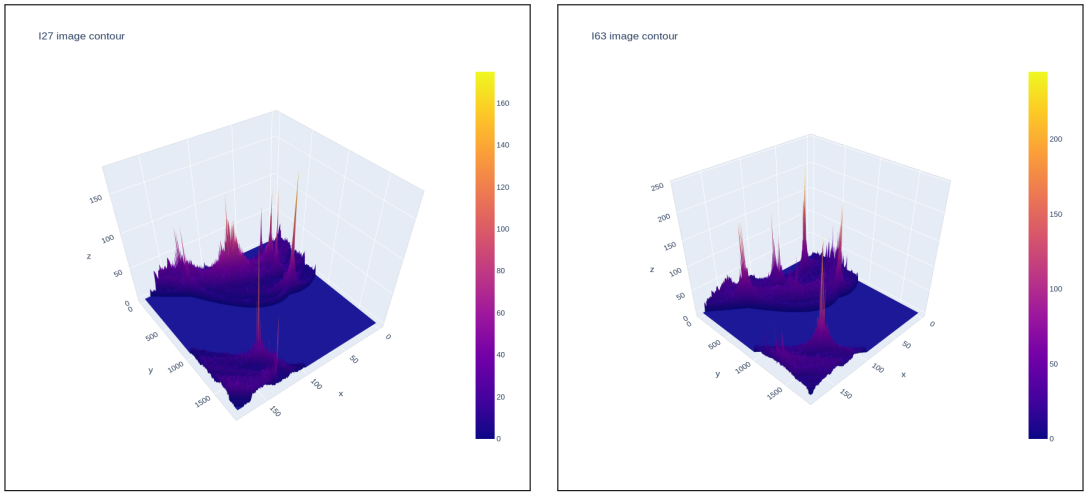


Figure 2.8: The 3D plot of the accumulator matrix (Left for mercedes symbol and Right for other collection of symbols).

2.4 Complexity

The Hough transform algorithm is computationally very expensive. The space complexity of the algorithm is k^n. Here since this is done for 2 dimensional Hough space, so the complexity is k^2. Time complexity for this algorithm is k (constant) in terms of voting elements, i.e., edge points. This cannot be processed in real time if the size of the image is very large, so certain other types of feature detectors are used in that case. This is the naive version of feature detection.

2.5 Improvements: Probabilistic Hough Transform

Here, we can see that even for a line with just two arguments, it takes a lot of time to compute the accumulator space. Probabilistic Hough Transform is an optimization to the naive Hough

Transform. It doesn't take all the points into consideration, rather it takes a random subset of points and that is sufficient for line detection.

Chapter 3

Hough Transform to detect circles

3.1 Theory

This is similar to Hough transform for detecting lines, except the fact that this is used for feature extraction for detection of circles [8] in imperfect images.

A circle is represented mathematically [9] as:

$$(x - x_{center})^2 + (y - y_{center})^2 = r^2 \tag{3.1}$$

Here, (x_{center}, y_{center}) is the center of the circle, and r is the radius of the circle. When the (x, y) is fixed, the parameters can be found according to eq. (3.1). The parameter space would be a 3D matrix (a, b, r). The circles would be identified by intersection of many conic surface of inverted right angle cones whose apex would be $(x, y, 0)$. In 3D space, the circle parameters can be identified by the intersection of many conic surfaces that are defined by points on the 2D circle.

There are basically two stages to do this, firstly fixing radius then finding the optimal center of the circles in 2D parameter space, the second stage is to find the optimal radius in a one dimensional parameter space.

3.2 Mathematical form

In parametric form circle can be represented as:

$$x = a + r\cos\theta \tag{3.2}$$

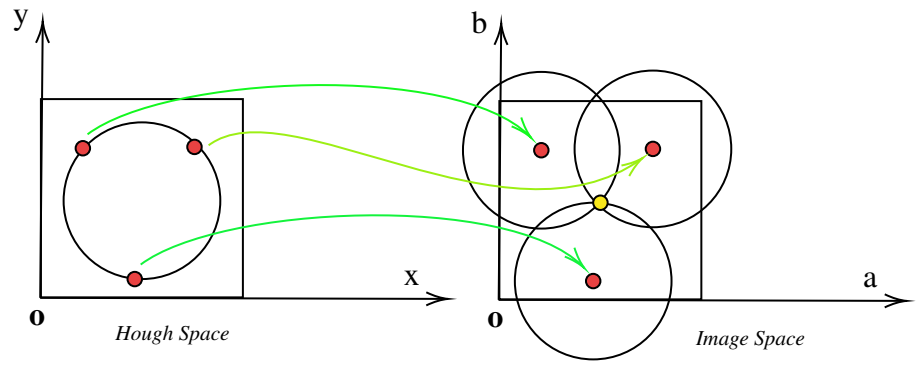


Figure 3.1: The circle in the Hough space and circle formed by intersection in the image space.

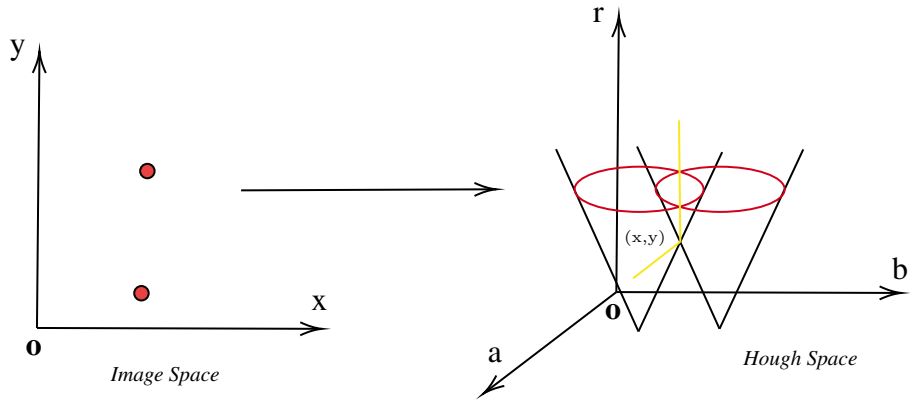


Figure 3.2: For an unknown radius r, the unknown gradient direction can be represented in terms of cones in 3D Hough space.

$$y = b + r \sin \theta \tag{3.3}$$

If the radius is fixed, then the parameter space would be reduced to 2D as shown in fig. 3.1. If we consider each point in the original circle, it can define a circle centered at (x, y) with radius r. The intersection of all such circles in the parameter space would correspond to the center point in the original circle. Consider fig. 3.1 where we detect 3 points in the original circle as shown in the left, corresponding to which we get the intersection of all the circles formed by taking them as the radius.

We use the same type of accumulator matrix here too which finds the intersection point in the parameter space as shown in fig. 3.2. The elements in the accumulator matrix denotes the number of "circles" in the parameter space that is passing through the corresponding grid cell in the parameter space formed by buckets. We use the same "voting" technique. The initial matrix is formed of all zeros. For each edge point in the original space we formulate a circle in the parameter space. We find the local maxima in the accumulator matrix via a certain threshold.

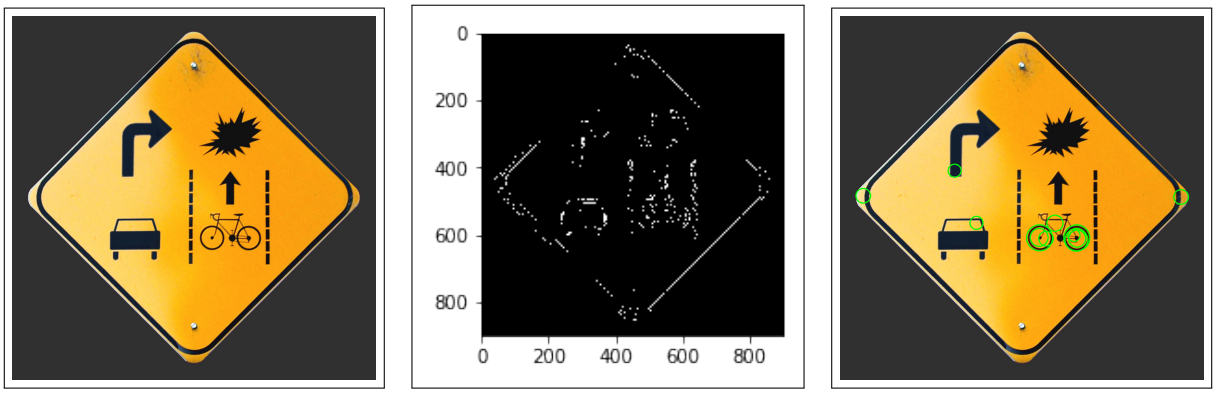


Figure 3.3: A figure containing the images by applying Hough Circle transform. (Left: An original test image before applying Hough Circle transform, Middle: After applying preprocessing to the first image, Right: Image after applying Hough Circle transform results in detection of Circles). Picture of Diamond Sign is used with written permission from Donal Knuth.

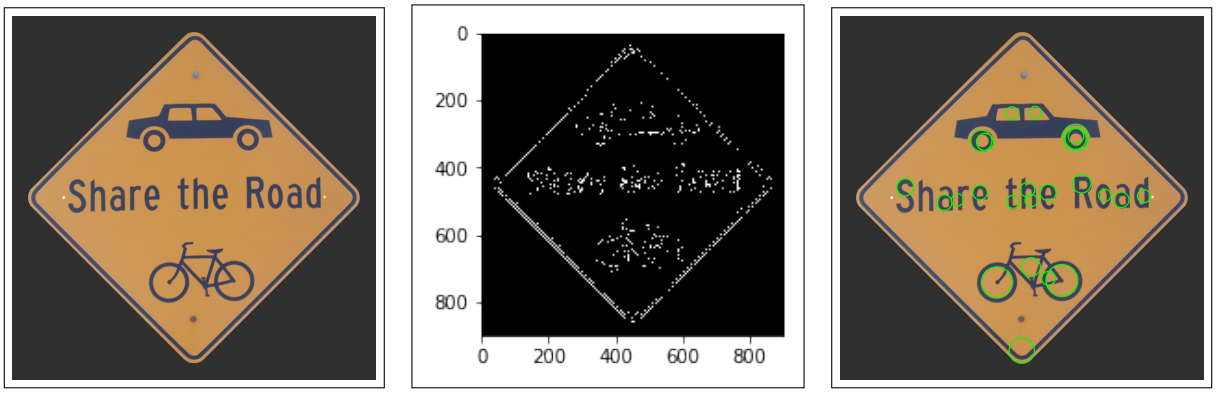
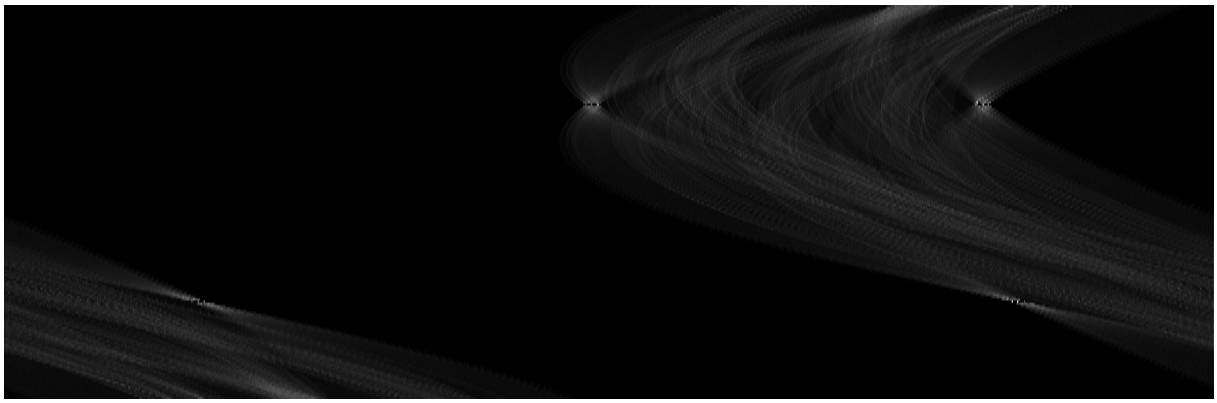


Figure 3.4: A figure containing the images by applying Hough Circle transform. (Left: An original test image before applying Hough Circle transform, Middle: After applying preprocessing to the first image, Right: Image after applying Hough Circle transform results in detection of Circles). Picture of Diamond Sign is used with written permission from Donal Knuth.

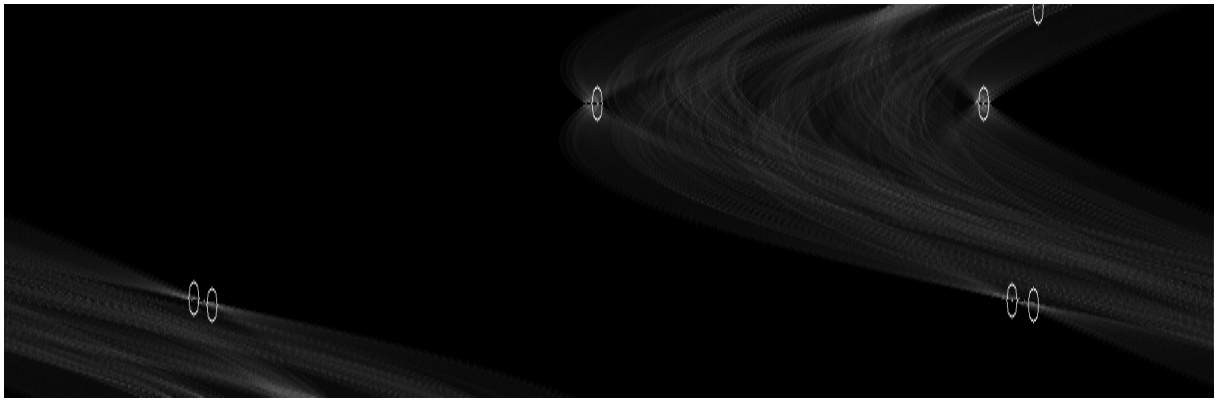
3.3 Our Implementation

The algorithm for Circle Hough Transform is,

1. Input image.
2. Detect the edge via edge detector algorithm
(e.g.: Canny Edge detector).
3. Select a threshold and convert the image to binary.
4. Initialize a 3D accumulator array.
5. For each pixel and each r in the image compute a and b.
6. Store the value of a, b, and r in 3D accumulator array.
7. Pick a threshold to select the best points.
8. Plot the circles in the original image.

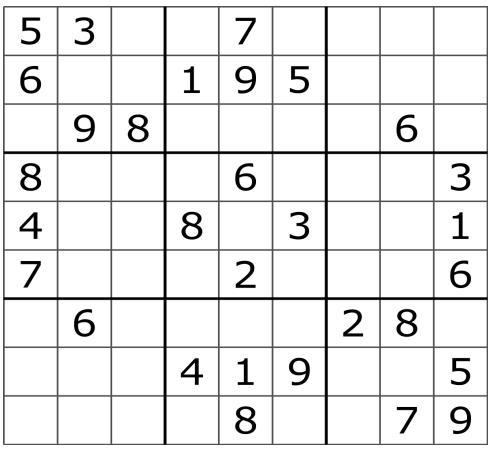


(a) Accumulator matrix by applying Hough Line transform on the test image.

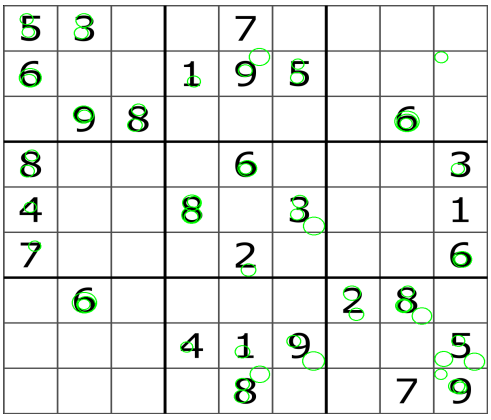


(b) Points selected by applying a threshold on the accumulator matrix on the test image.

Figure 3.5: A figure containing the images by applying Hough Line transform on the noised version of the first image.



(a) An image of sudoku



(b) After applying Hough Circle Transform on the image

Figure 3.6: A figure containing the images by applying Hough Circle transform on the image of sudoku. Picture Courtesy: wikimedia.

We can see that the same type of algorithm in different parametric form is used by the Hough Circle detector algorithm. Here, the algorithm can cope with occlusion and gaps caused by noise since each point is processed independently. It can also detect multiple instance [10] of model in a single pass. The complexity of time search increases exponentially with the number of model parameters. Non-target shapes can produce spurious peaks in parameter space. It is tricky to select good bins for faster computation for determining the circles.

We first take an input image as shown in Figure 3.4 (left). The preprocessed image for detecting edge is achieved via image Figure 3.4 (middle). We then apply Hough Circle transform according to the algorithm, the resulting image after the Hough transform Circle detection algorithm finish executing is shown in Figure 3.4 (right).

The accumulator matrix corresponding to the given image is shown in Figure 3.5a. After applying the threshold on the accumulator matrix we get the selected points as line in Figure 3.5b.

Similarly we take an input image as shown in Figure 3.3 (left). The preprocessed image for detecting edge is achieved via image Figure 3.3 (middle). We then apply Hough Circle transform according to the algorithm, the resulting image after the Hough transform Circle detection algorithm finish executing is shown in Figure 3.3 (right). The algorithm is implemented for image of sudoku as shown in fig. 3.6a. The result of the algorithm is given in fig. 3.6b. Here we see that the algorithm detects many other digits which looks like circle in occlusion as original circle.

3.4 Improvements: Adaptive Hough Transform

The only modification is the Adaptive Hough Transform uses a small accumulator array and the flexible ”coarse to fine” accumulation and search strategy for determining peaks in the Hough parameter space. This is basically superior to its previous version in terms of storage and computations.

Chapter 4

Conclusion

We have seen that the problem of line and circle detection becomes easy when we represent the edge image in parametric form. We apply a set of voting algorithm to determine the exact location of lines and circles [11] in the original image. The algorithm is robust to some occlusion in the original image and even works when there is a considerable amount of noise in the image. The main aim of the invention is to automate the process of detecting lines and circles in images which requires a lot of patience when done by humans. The algorithm is robust to sizes and works quite well as a traditional line and edge detection algorithm.

4.1 Applying to other examples

We apply the full set of Hough line and circle detection algorithm to the picture of Uchiha Clan as shown in fig. 4.1. The accumulator matrix for the circles obtained via running a robust loop from a begining to end radius is shown in fig. 4.2a. The detected circles are shown in fig. 4.2b. The edge version of the preprocessed image is got in fig. 4.3 (right). The 3D plot of the accumulator matrix is shown in fig. 4.3 (left). Similarly we use the image of sudoku as shown in fig. 4.4a and the resultant image is got in fig. 4.4b. We also try the algorithm to an image containing coins and line as shown in fig. 4.5a and the output obtained is shown in fig. 4.5b.

4.2 A further note on the two algorithms

It is clear that the circle detection algorithm is performing better than the line detection, since we are using a range of radius to test the algorithm. We also see that the preprocessing of the image beforehand can give a better result in both the detection of the lines and circles. The line

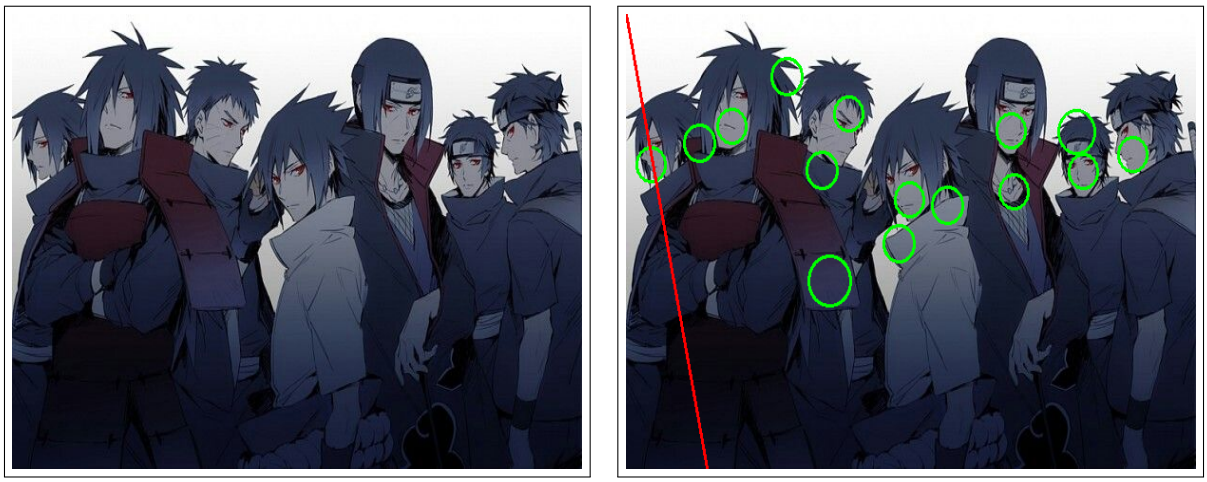
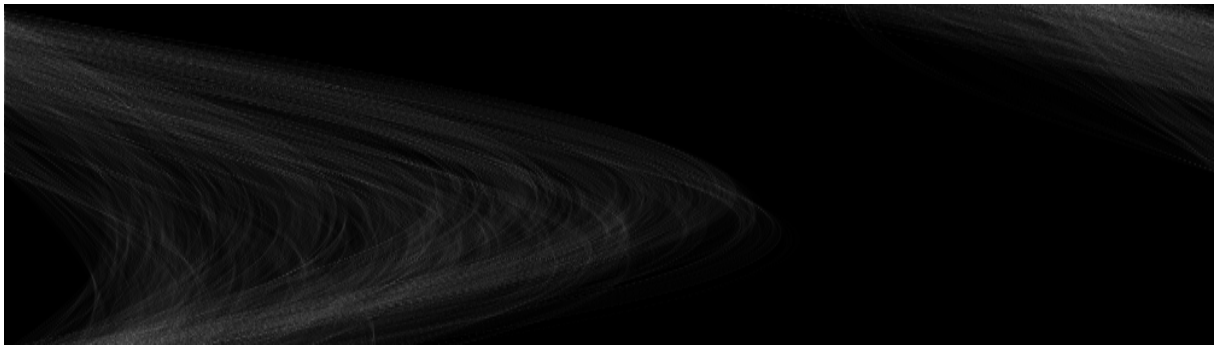
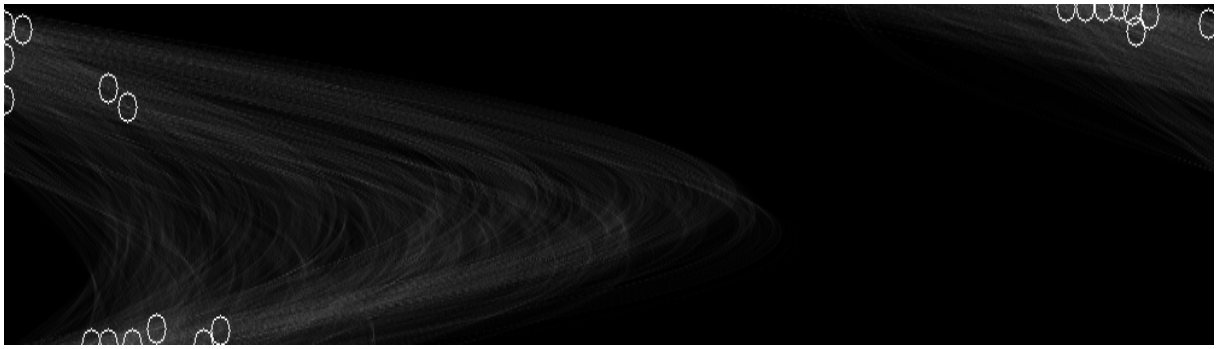


Figure 4.1: A figure containing the images by applying Hough Circle and Line transform. (Left: An original image of Uchiha Clan before applying Hough transforms, Right: Image after applying Hough Circle and Line transform results in detection of Circles and Lines). Source: steamuserimages.

detection algorithm was used in a constraint environment with a good amount of hand tuned pre-processing in the bubble chamber when it was patented. There might be some misclassification of circles like the circles may not be present but also gets detected, but this is a good algorithm for its time (1962's). The computation is a bit extensive and there has been various papers which deals with the optimization of this algorithm in terms of correctness and computation.



(a) Accumulator matrix by applying Hough Line transform on the test image.



(b) Points selected by applying a threshold on the accumulator matrix on the test image.

Figure 4.2: A figure containing the images by applying Hough Line transform on the noised version of the first image.

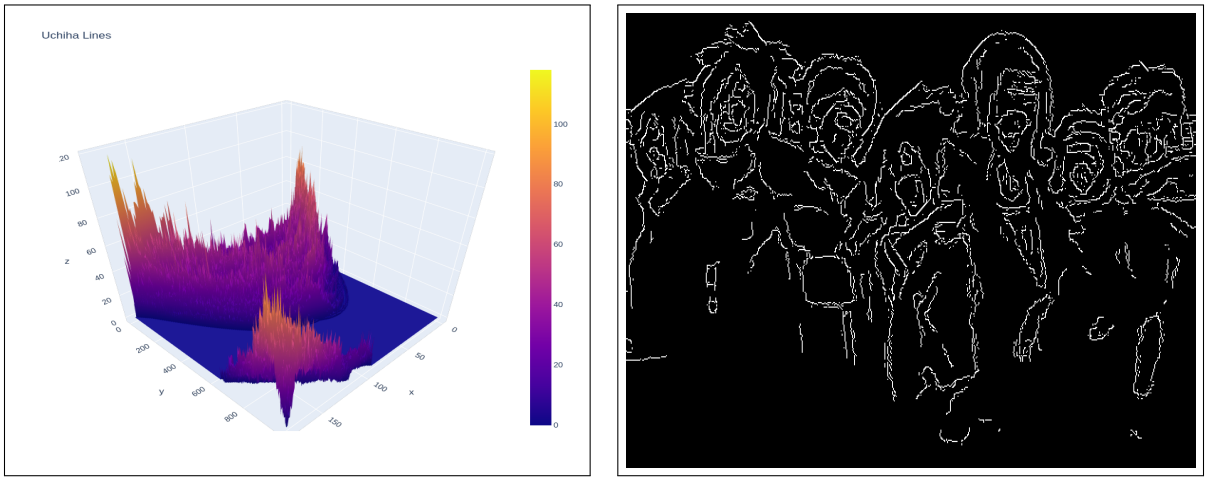
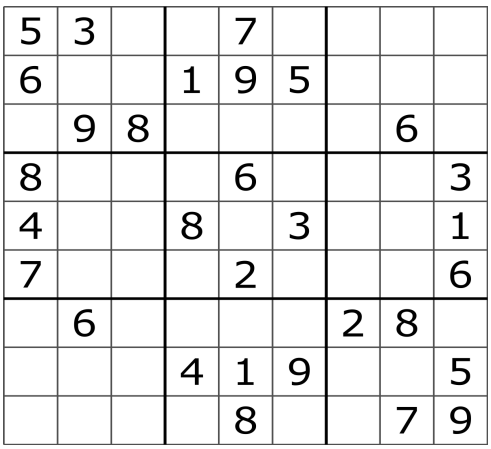
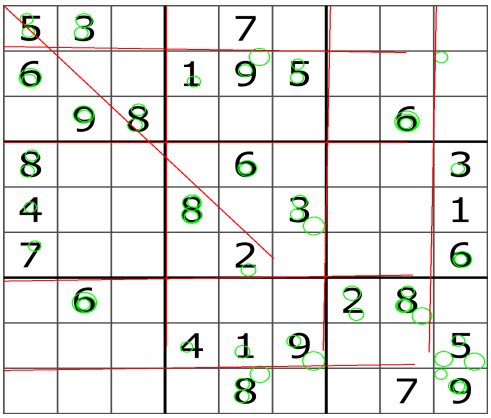


Figure 4.3: The 3D plot (left) got from the accumulator matrix when applied to the edge detected (right) version of the image of Uchiha clan..



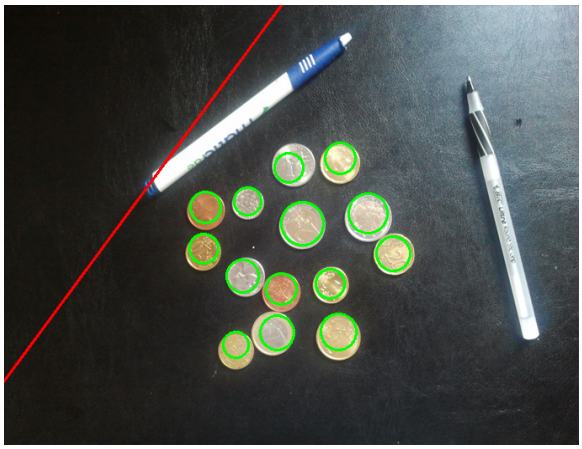
(a) An image of sudoku



(b) After applying Hough Circle Transform on the image

Figure 4.4: A figure containing the images by applying Hough Circle and Line transform on the image of sudoku. Picture Courtsey: wikimedia.

(a) An image of sudoku



(b) After applying Hough Circle Transform on the image

Figure 4.5: A figure containing the images by applying Hough Circle and Line transform on a test image.

References

[1] P. V. C. HOUGH, "Method and means for recognizing complex patterns," 1962. [Online https://patents.google.com/patent/US3069654A/en; accessed 21-February-2020].

[2] W. contributors, "Feature detection (computer vision)," 2020. [Online https://en.wikipedia.org/wiki/Feature_detection_(computer_vision); accessed 21-February-2020].

[3] J. B. Pal, "A deeper look into hybrid images," 2020. [Online https://arxiv.org/abs/2001.11302; accessed 21-February-2020].

[4] W. contributors, "Ridge detection," 2019. [Online https://en.wikipedia.org/wiki/Ridge_detection; accessed 21-February-2020].

[5] W. contributors, "Hough transform," 2020. [Online https://en.wikipedia.org/wiki/Hough_transform; accessed 21-February-2020].

[6] O. contributors, "Hough line transform," 2013. [Online https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html; accessed 21-February-2020].

[7] F. Masci, "Line detection by hough transformation," 2009. [Online http://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/HoughTrans_lines_09.pdf; accessed 21-February-2020].

[8] W. contributors, "Circle hough transform," 2019. [Online https://en.wikipedia.org/wiki/Circle_Hough_Transform; accessed 21-February-2020].

[9] O. contributors, "Hough circle transform," 2013. [Online https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghcircles/py_houghcircles.html; accessed 21-February-2020].

[10] T. Maharaj, "Project 2: Detecting lines and cirlces in images," 2020. [Online; accessed 21-February-2020].

[11] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," 1972.

Index

Appendix A

Codes

A.1 Python code corresponding to the Line and Circle Detection Algorithm

Here we build our own version of Hough Transform Line and Circle detection function of opencv by using numpy in Python3.

```
import numpy as np
import cv2 # You must not use any methods which has 'hough' in it!
from utils import  hough_peaks


def hough_lines_vote_acc(edge_img, rho_res=1, thetas= np.arange
    (0,180)):
     """
       Creating an Hough Vote Accumulator. The Generated grid will
    have thetas on
       X axis, and rhos on the Y axis.

       Args
       - edge_img: numpy nd-array of dim (m, n)
       - rho_res: rho resolution. Default is 1. This controls how
    dense your grid columns are
       - thetas: theta values.

       Returns
       - accumulator with votes for each grid point, thetas, and rhos

```

```
      HINTS:
      - I encourage you to try implementing this naively first, just be aware that
       it may take an absurdly long time to run. You will need to get a function
       that takes a reasonable amount of time to run so that I can verify
       your code works.
    """
    ############################
    ### TODO: YOUR CODE HERE ###


    # x*cos(theta) + y*sin(theta) max value sin,cos theta can take is 1
    max_rad = edge_img.shape[0] + edge_img.shape[1]
    accumulator = [[0 for x in range(180)] for y in range(max_rad)] #np.zeros((max_rad,max_rad))
    #print(accumulator)
    for x in range(edge_img.shape[0]):
        for y in range(edge_img.shape[1]):
            if edge_img[x][y] == 255:
                for theta in thetas:
                    # convert it into radians multiply by (np.pi/180)
                    rho = x*np.cos(theta*np.pi/180)+y*np.sin(theta*np.pi/180)
                    #print(rho)
                    #print("Type = ",rho," ",theta)
                    accumulator[int(rho.astype(int))][theta] += 1
    rhos = [x for x in range(max_rad)]
    accumulator = np.asarray(accumulator, dtype=np.uint8)


    ### END OF STUDENT CODE ####
    ############################
    return accumulator, thetas, rhos


def hough_circles_vote_acc(edge_img, radius):
    """
```

```
    Creating an Hough Vote Accumulator. The Generated grid will have
    x coordinate of the center of cirlce on
    X axis, and y coordinate of the center of cirlces on the Y axis.

    Args
    - edge_img: numpy nd-array of dim (m, n)
    - radius: radius of the circle

    Returns
    - accumulator with votes for each grid point

    HINTS:
    - I encourage you to try implementing this naively first, just be aware that
     it may take an absurdly long time to run. You will need to get a function
     that takes a reasonable amount of time to run so that I can verify
     your code works.
    """
    ############################
    ### TODO: YOUR CODE HERE ###

    # Naive version of the accumulator needs 3 dimensional matrix
    # Hough Gradient Method will give faster results

    accumulator = [[0 for x in range(edge_img.shape[1])] for y in range(edge_img.shape[0])]
    for x in range(edge_img.shape[0]):
        for y in range(edge_img.shape[1]):
            if edge_img[x][y] == 255:
                    for theta in range(0,360):
                        b = x - radius * np.cos(theta*np.pi/180)    # x     r * cos(t * PI / 180);
                        a = y - radius * np.sin(theta*np.pi/180)    # y     r * sin(t * PI / 180);
                        try:
                            # wild cards!
```

```
                            accumulator[int(abs(b))][int(abs(a))] += 1
                        except:
                            pass
     accumulator = np.array(accumulator, dtype=np.uint8)


     ### END OF STUDENT CODE ####
     ############################
     return accumulator



def find_circles(edge_img, radius_range=[1,2], threshold=100, nhood_size=10):
     """
       A naive implementation of the algorithm for finding all the circles in a range.
       Feel free to write your own more efficient method [Extra Credit].
       For extra credit, you may need to add additional arguments.


       Args
       - edge_img: numpy nd-array of dim (m, n).
       - radius_range: range of radius. All cicles whose radius falls in between should be selected.
       - nhood_size: size of the neighborhood from where only one candidate can be chosen.

       Returns
       - centers, and radii i.e., (x, y) coordinates for each circle.

       HINTS:
       - I encourage you to use this naive version first. Just be aware that
        it may take a long time to run. You will get EXTRA CREDIT if you can write a faster
        implementaiton of this method, keeping the method signature (input and output parameters)
        unchanged.
```

```
    """
    n = radius_range[1] - radius_range[0]
    H_size = (n,) + edge_img.shape
    H = np.zeros(H_size, dtype=np.uint)
    centers = ()
    radii = np.arange(radius_range[0], radius_range[1])
    valid_radii = np.array([], dtype=np.uint)
    num_circles = 0
    for i in range(len(radii)):
        H[i] = hough_circles_vote_acc(edge_img, radii[i])
        peaks = hough_peaks(H[i], numpeaks=10, threshold=threshold,
                            nhood_size=nhood_size)
        if peaks.shape[0]:
            valid_radii = np.append(valid_radii, radii[i])
            centers = centers + (peaks,)
            for peak in peaks:
                cv2.circle(edge_img, tuple(peak[::-1]), radii[i]+1,
(0,0,0), -1)
        #  cv2.imshow('image', edge_img); cv2.waitKey(0); cv2.
destroyAllWindows()
        num_circles += peaks.shape[0]
        print('Progress: %d%% - Circles: %d\033[F\r'%(100*i/len(radii
), num_circles))
    print('Circles detected: %d          '%(num_circles))
    centers = np.array(centers)
    return centers, valid_radii.astype(np.uint)
```

Generated using Jimut's LaTeX, Version 1.4. Department of Computer Science, Ramakrishna Mission Vivekananda Educational and Research Institute, Kolkata, India.

This thesis was generated on Wednesday 11th March, 2020 at 9:38pm.