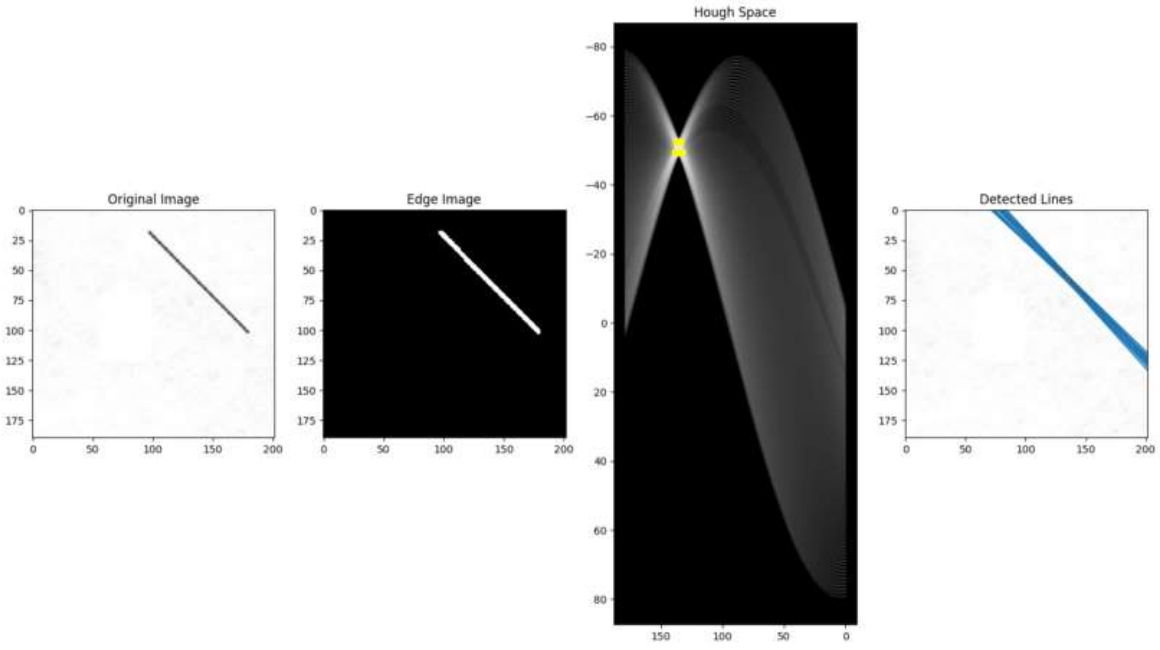
Lines Detection with Hough Transform

An algorithm to find lines in images

Socret Lee
May 2 · 6 min read ★



Line Detection using the Hough Transform Algorithm

> ***Note:*** *You can read the Chinese version of this article here.*

I. Motivation

Recently, I found myself having to incorporate a document scanner feature into an app. After doing some research, I came across an article written by Ying Xiong who was a member of the Dropbox's machine learning team. The article explains how the Dropbox's machine learning team implemented their document scanner by

highlighting the steps they went through and the algorithms used in each step (Xiong, 2016). Through that article, I learnt about a method called the Hough Transform and how it can be used to detect lines in images. Hence, in this article, I would like to explain the Hough Transform algorithm and provide a "from-scratch" implementation of the algorithm in Python.

II. The Hough Transform

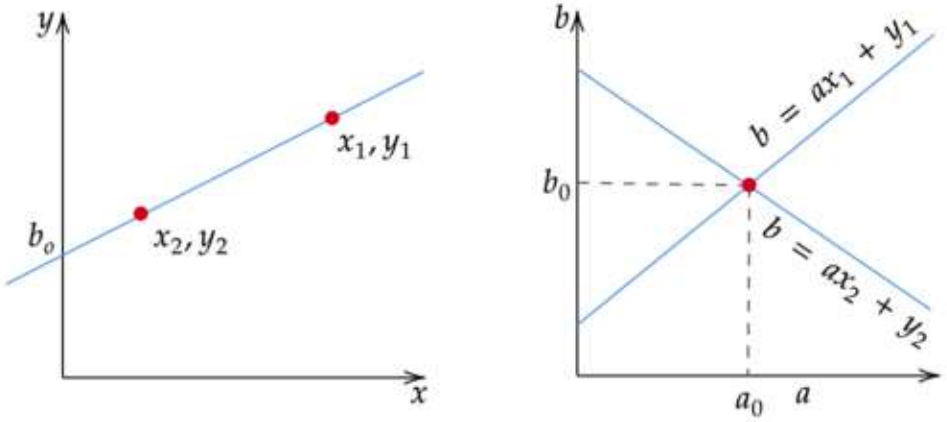
The Hough Transform is an algorithm patented by Paul V. C. Hough and was originally invented to recognize complex lines in photographs (Hough, 1962). Since its inception, the algorithm has been modified and enhanced to be able to recognize other shapes such as circles and quadrilaterals of specific types. In order to understand how the Hough Transform algorithm works, it is important to understand four concepts: edge image, the Hough Space and the mapping of edge points onto the Hough Space, an alternate way to represent a line, and how lines are detected.

Edge Image

Canny Edge Detection Algorithm. Source: AI Shack

An edge image is the output of an edge detection algorithm. An edge detection algorithm detects edges in an image by determining where the brightness/intensity of an image changes drastically ("Edge Detection — Image Processing with Python", 2020). Examples of edge detection algorithms include: Canny, Sobel, Laplacian, etc. It is common for an edge image to be binarized meaning all of its pixel values are either a 1 or a 0. Depending on your situation, either a 1 or a 0 can signify an edge pixel. For the Hough Transform algorithm, it is crucial to perform edge detection first to produce an edge image which will then be used as input into the algorithm.

The Hough Space and the Mapping of Edge Points onto the Hough Space



Mapping from edge points to the Hough Space.

The Hough Space is a 2D plane that has a horizontal axis representing the slope and the vertical axis representing the intercept of a line on the edge image. A line on an edge image is represented in the form of $y = ax + b$ (Hough, 1962). One line on the edge image produces a point on the Hough Space since a line is characterized by its slope a and intercept b. On the other hand, an edge point (x_i, y_i) on the edge image can have an infinite number of lines pass through it. Therefore, an edge point produces a line in the Hough Space in the form of $b = ax_i + y_i$ (Leavers, 1992). In the Hough Transform algorithm, the Hough Space is used to determine whether a line exists in the edge image.

An Alternate Way to Represent a Line

$$m = \frac{rise}{run} = \frac{y_2 - y_1}{x_2 - x_1}$$

m = slope

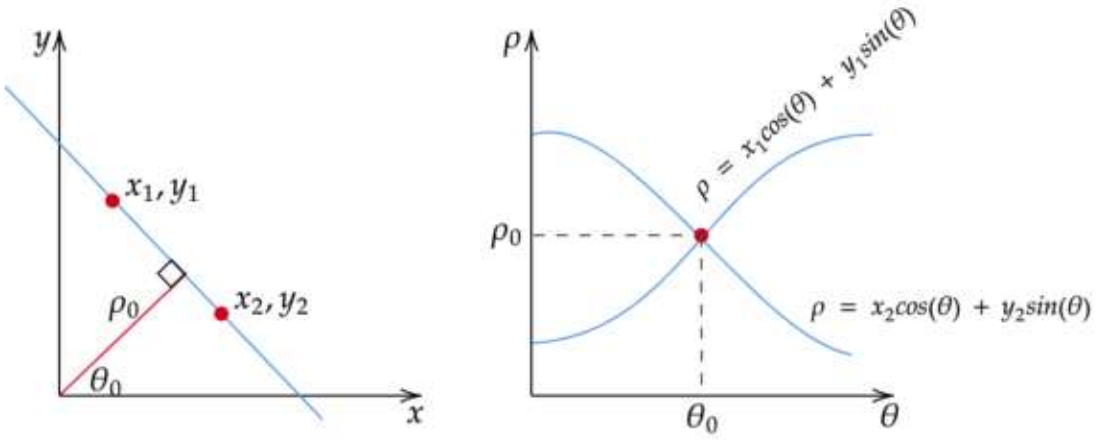
(x_1, y_1) = first point

(x_2, y_2) = second point

The equation to calculate a slope of a line.

There is one flaw with representing lines in the form of $y = ax + b$ and the Hough Space with the slope and intercept. In this form, the algorithm won't be able to detect

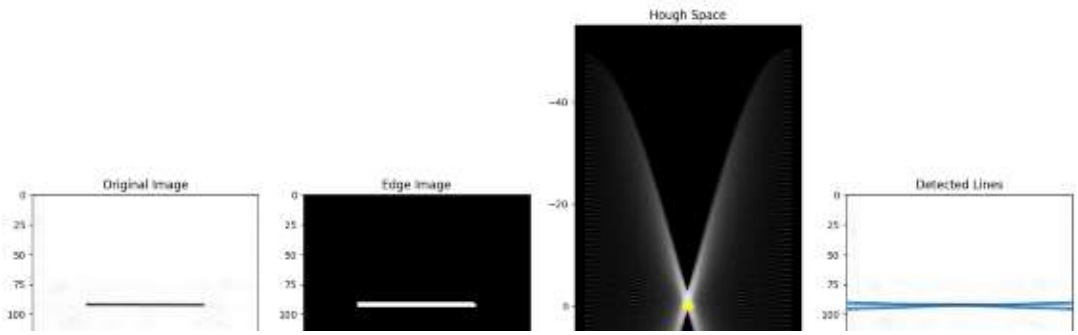
vertical lines because the slope a is undefined/infinity for vertical lines (Leavers, 1992). Programmatically, this means that a computer would need an infinite amount of memory to represent all possible values of a. To avoid this issue, a straight line is instead represented by a line called the normal line that passes through the origin and perpendicular to that straight line. The form of the normal line is $\rho = x\,cos(\theta) + y\,sin(\theta)$ where ρ is the length of the normal line and θ is the angle between the normal line and the x axis.

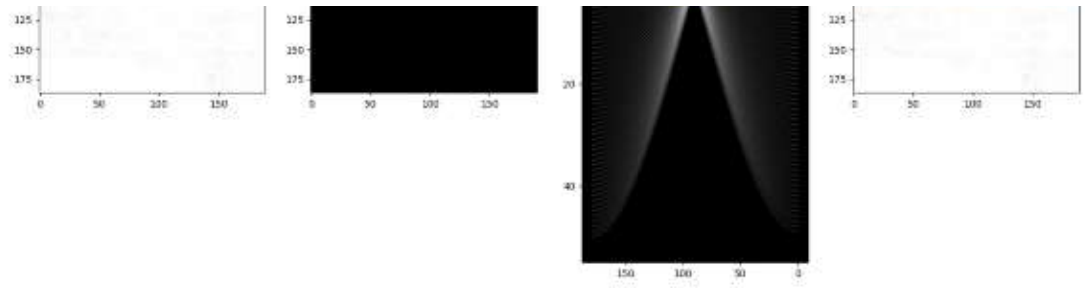


An alternative representation of a straight line and its corresponding Hough Space.

Using this, instead of representing the Hough Space with the slope a and intercept b, it is now represented with ρ and θ where the horizontal axis are for the θ values and the vertical axis are for the ρ values. The mapping of edge points onto the Hough Space works in a similar manner except that an edge point (x_i, y_i) now generates a cosine curve in the Hough Space instead of a straight line (Leavers, 1992). This normal representation of a line eliminates the issue of unbounded value of a that arises when dealing with vertical lines.

Line Detection





The process of detecting lines in an image. The yellow dots in the Hough Space indicate that lines exist and is represented by the θ and ρ pairs.

As mentioned, an edge point produces a cosine curve in the Hough Space. From this, if we were to map all the edge points from an edge image onto the Hough Space, it will generate a lot of cosine curves. If two edge points lay on the same line, their corresponding cosine curves will intersect each other on a specific (ρ, θ) pair. Thus, the Hough Transform algorithm detects lines by finding the (ρ, θ) pairs that has a number of intersections larger than a certain threshold. It is worth noting that this method of thresholding might not always yield the best result without doing some preprocessing like neighborhood suppression on the Hough Space to remove similar lines in the edge image.

III. The Algorithm

1. Decide on the range of ρ and θ. Often, the range of θ is [0, 180] degrees and ρ is [-d, d] where d is the length of the edge image's diagonal. It is important to quantize the range of ρ and θ meaning there should be a finite number of possible values.

2. Create a 2D array called the accumulator representing the Hough Space with dimension (*num_rhos*, *num_thetas*) and initialize all its values to zero.

3. Perform edge detection on the original image. This can be done with any edge detection algorithm of your choice.

4. For every pixel on the edge image, check whether the pixel is an edge pixel. If it is an edge pixel, loop through all possible values of θ, calculate the corresponding ρ, find the θ and ρ index in the accumulator, and increment the accumulator base on those index pairs.

5. Loop through all the values in the accumulator. If the value is larger than a certain threshold, get the ρ and θ index, get the value of ρ and θ from the index pair which can then be converted back to the form of $y = ax + b$.

IV. The Code

Non Vectorized Solution

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as mlines


def line_detection_non_vectorized(image, edge_image, num_rhos=180, num_thetas=180, t_count=220)
  edge_height, edge_width = edge_image.shape[:2]
  edge_height_half, edge_width_half = edge_height / 2, edge_width / 2
  #
  d = np.sqrt(np.square(edge_height) + np.square(edge_width))
  dtheta = 180 / num_thetas
  drho = (2 * d) / num_rhos
  #
  thetas = np.arange(0, 180, step=dtheta)
  rhos = np.arange(-d, d, step=drho)
  #
  cos_thetas = np.cos(np.deg2rad(thetas))
  sin_thetas = np.sin(np.deg2rad(thetas))
  #
  accumulator = np.zeros((len(rhos), len(rhos)))
  #
  figure = plt.figure(figsize=(12, 12))
  subplot1 = figure.add_subplot(1, 4, 1)
  subplot1.imshow(image)
  subplot2 = figure.add_subplot(1, 4, 2)
  subplot2.imshow(edge_image, cmap="gray")
  subplot3 = figure.add_subplot(1, 4, 3)
  subplot3.set_facecolor((0, 0, 0))
  subplot4 = figure.add_subplot(1, 4, 4)
  subplot4.imshow(image)
  #
  for y in range(edge_height):
    for x in range(edge_width):
      if edge_image[y][x] != 0:
        edge_point = [y - edge_height_half, x - edge_width_half]
        ys, xs = [], []
        for theta_idx in range(len(thetas)):
          rho = (edge_point[1] * cos_thetas[theta_idx]) + (edge_point[0] * sin_thetas[theta_idx
          theta = thetas[theta_idx]
          rho_idx = np.argmin(np.abs(rhos - rho))
          accumulator[rho_idx][theta_idx] += 1
```

```
            ys.append(rho)
            xs.append(theta)
        subplot3.plot(xs, ys, color="white", alpha=0.05)

  for y in range(accumulator.shape[0]):
    for x in range(accumulator.shape[1]):
      if accumulator[y][x] > t_count:
        rho = rhos[y]
        theta = thetas[x]
        a = np.cos(np.deg2rad(theta))
        b = np.sin(np.deg2rad(theta))
        x0 = (a * rho) + edge_width_half
        y0 = (b * rho) + edge_height_half
        x1 = int(x0 + 1000 * (-b))
        y1 = int(y0 + 1000 * (a))
        x2 = int(x0 - 1000 * (-b))
        y2 = int(y0 - 1000 * (a))
        subplot3.plot([theta], [rho], marker='o', color="yellow")
        subplot4.add_line(mlines.Line2D([x1, x2], [y1, y2]))

  subplot3.invert_yaxis()
  subplot3.invert_xaxis()

  subplot1.title.set_text("Original Image")
  subplot2.title.set_text("Edge Image")
  subplot3.title.set_text("Hough Space")
  subplot4.title.set_text("Detected Lines")
  plt.show()
  return accumulator, rhos, thetas


if __name__ == "__main__":
  for i in range(3):
    image = cv2.imread(f"sample-{i+1}.png")
    edge_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    edge_image = cv2.GaussianBlur(edge_image, (3, 3), 1)
    edge_image = cv2.Canny(edge_image, 100, 200)
    edge_image = cv2.dilate(
        edge_image,
        cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)),
        iterations=1
    )
    edge_image = cv2.erode(
        edge_image,
        cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)),
        iterations=1
    )
    line_detection_non_vectorized(image, edge_image)
```

Vectorized Solution

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as mlines


def line_detection_vectorized(image, edge_image, num_rhos=180, num_thetas=180, t_count=220):
  edge_height, edge_width = edge_image.shape[:2]
  edge_height_half, edge_width_half = edge_height / 2, edge_width / 2
  #
  d = np.sqrt(np.square(edge_height) + np.square(edge_width))
  dtheta = 180 / num_thetas
  drho = (2 * d) / num_rhos
  #
  thetas = np.arange(0, 180, step=dtheta)
  rhos = np.arange(-d, d, step=drho)
  #
  cos_thetas = np.cos(np.deg2rad(thetas))
  sin_thetas = np.sin(np.deg2rad(thetas))
  #
  accumulator = np.zeros((len(rhos), len(rhos)))
  #
  figure = plt.figure(figsize=(12, 12))
  subplot1 = figure.add_subplot(1, 4, 1)
  subplot1.imshow(image)
  subplot2 = figure.add_subplot(1, 4, 2)
  subplot2.imshow(edge_image, cmap="gray")
  subplot3 = figure.add_subplot(1, 4, 3)
  subplot3.set_facecolor((0, 0, 0))
  subplot4 = figure.add_subplot(1, 4, 4)
  subplot4.imshow(image)
  #
  edge_points = np.argwhere(edge_image != 0)
  edge_points = edge_points - np.array([[edge_height_half, edge_width_half]])
  #
  rho_values = np.matmul(edge_points, np.array([sin_thetas, cos_thetas]))
  #
  accumulator, theta_vals, rho_vals = np.histogram2d(
      np.tile(thetas, rho_values.shape[0]),
      rho_values.ravel(),
      bins=[thetas, rhos]
```

```
    )
    accumulator = np.transpose(accumulator)
    lines = np.argwhere(accumulator > t_count)
    rho_idxs, theta_idxs = lines[:, 0], lines[:, 1]
    r, t = rhos[rho_idxs], thetas[theta_idxs]

    for ys in rho_values:
      subplot3.plot(thetas, ys, color="white", alpha=0.05)

    subplot3.plot([t], [r], color="yellow", marker='o')

    for line in lines:
      y, x = line
      rho = rhos[y]
      theta = thetas[x]
      a = np.cos(np.deg2rad(theta))
      b = np.sin(np.deg2rad(theta))
      x0 = (a * rho) + edge_width_half
      y0 = (b * rho) + edge_height_half
      x1 = int(x0 + 1000 * (-b))
      y1 = int(y0 + 1000 * (a))
      x2 = int(x0 - 1000 * (-b))
      y2 = int(y0 - 1000 * (a))
      subplot3.plot([theta], [rho], marker='o', color="yellow")
      subplot4.add_line(mlines.Line2D([x1, x2], [y1, y2]))

    subplot3.invert_yaxis()
    subplot3.invert_xaxis()

    subplot1.title.set_text("Original Image")
    subplot2.title.set_text("Edge Image")
    subplot3.title.set_text("Hough Space")
    subplot4.title.set_text("Detected Lines")
    plt.show()
    return accumulator, rhos, thetas


if __name__ == "__main__":
    for i in range(3):
      image = cv2.imread(f"sample-{i+1}.png")
      edge_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
      edge_image = cv2.GaussianBlur(edge_image, (3, 3), 1)
      edge_image = cv2.Canny(edge_image, 100, 200)
      edge_image = cv2.dilate(
          edge_image,
          cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)),
          iterations=1
      )
```

```
    )
    edge_image = cv2.erode(
        edge_image,
        cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)),
        iterations=1
    )
    line_detection_vectorized(image, edge_image)
```

V. Conclusion

To conclude, this article showcased the Hough Transform algorithm in its simplest form. As mentioned, this algorithm can extend beyond detecting straight lines. Over the years, many improvements have been made to this algorithm that allow it to detect other shapes such as circles, triangles, and even quadrilaterals of specific shapes. This resulted in many useful real world applications ranging from document scanning to lane detection in self-driving cars. I'm confident that many more amazing technologies will be powered by this algorithm in the foreseeable future.

References

Edge Detection — Image Processing with Python. (2020, February 16). Retrieved from https://datacarpentry.org/image-processing/08-edge-detection/

Hough, P. V. C. (1962). *Method and means for recognizing complex patterns* (U.S. Patent 3069654). Retrieved from https://patentimages.storage.googleapis.com/9f/9f/f3/87610ddec32390/US3069654.pdf

Leavers, V. F. (1992). Preprocessing, *Shape Detection in Computer Vision Using Hough Transform* (pp. 39–64). doi:10.1007/978–1–4471–1940–1

Lin, C. (2018, December 17). *Tutorial: Build a lane detector*. Towards Data Science. Retrieved from https://towardsdatascience.com/tutorial-build-a-lane-detector-679fd8953132

Mukhopadhyay, P., & Chaudhuria, B. B. (2015). A survey of Hough Transform. *Pattern Recognition*. *48*(3), 993–1010. Retrieved from https://doi.org/10.1016/j.patcog.2014.08.027

Smith, B. (2018, September 21). *Hough Circle Transform*. ImageJ. Retrieved from https://imagej.net/Hough_Circle_Transform

Sodha, S. (2017). *An Implementation of Sobel Edge Detection*. projectrhea.org. Retrieved from https://www.projectrhea.org/rhea/index.php/An_Implementation_of_Sobel_Edge_Detection

The Hough Transform. (n.d.). Retrieved from https://aishack.in/tutorials/hough-transform-basics/

Xiong, Y. (2016). *Fast and Accurate Document Detection for Scanning*. Retrieved from https://dropbox.tech/machine-learning/fast-and-accurate-document-detection-for-scanning