Hough transforms

Adapted from: [Juan Cruz Martinez]

What is Hough space?

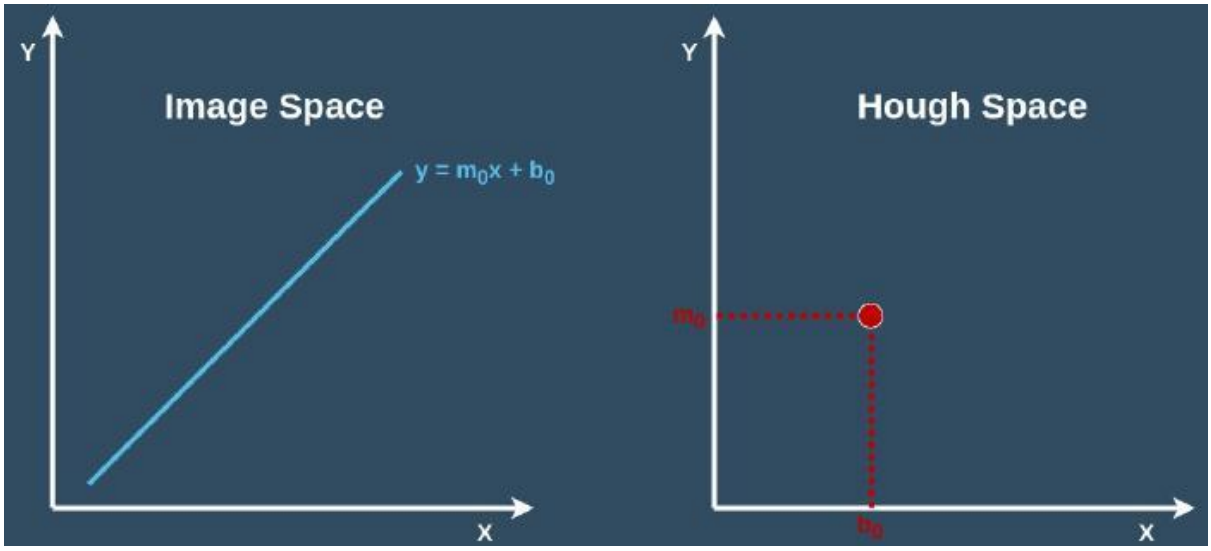
Before we start applying Hough transform to images, we need to understand what a Hough space is, and we will learn that in the way of an example.

Parameter space

When we work with images, we can imagine the image being a 2d matrix over some x and y coordinates, under which a line could be described as $y = mx + b$

Parameter space

But in parameter space, which we will call Hough space, I can represent that same line as m vs b instead, so the characterization of a line on image space, will be a single point at the position m-b in Hough space.

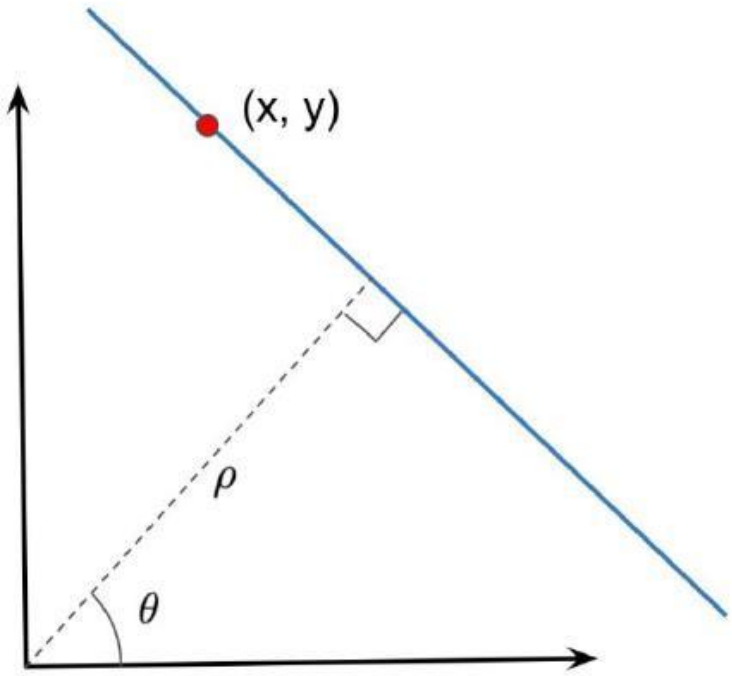


Hough Space

But we have a problem though, with $y = mx + b$, we cannot represent a vertical line, as the slope is infinite. So we need a better way parametrization, polar coordinates (rho and theta).

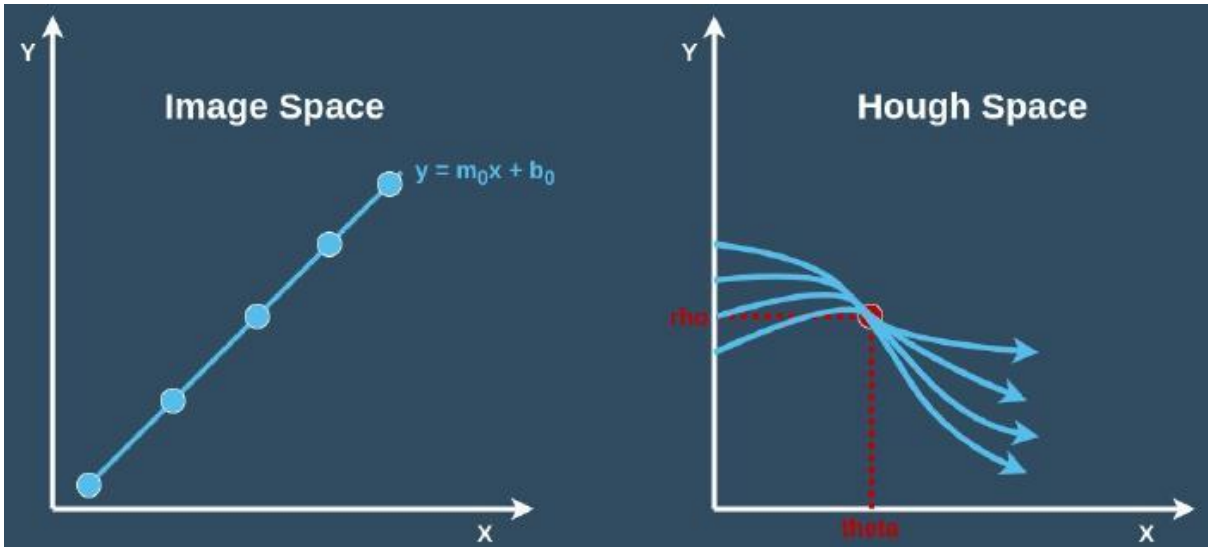
Hough space

- rho: describes the distance of the line from the origin
- theta: describes the angle away from horizontal



Line Polar Coordinates

One very important observation though, is what happens when we take multiple points around a line, and we transform into our Hough space.



Dots and Line relation in Hough Space

A single dot on image space translates to a curve on Hough space, with the particularity that points among a line on image space will be represented by multiple curves with a single touchpoint.

And this will be our target, finding the points where a group of curves intersects.

Summary:

The Hough transform is a way of finding the most likely values which represent a line (or a circle, or many other things).

You give the Hough transform a picture of a line as input. This picture will contain two types of pixels: ones which are part of the line, and ones which are part of the background.

For each pixel that is part of the line, all possible combinations of parameters are calculated. For example, if the pixel at co-ordinate (1, 100) is part of the line, then that could be part of a line where the gradient (m) = 0 and y-intercept (c) = 100. It could also be part of m = 1, c = 99; or m = 2, c = 98; or m = 3, c = 97; and so on. You can solve the line equation y = mx + c to find all possible combinations.

Each pixel gives one vote to each of the parameters (m and c) that could explain it. So you can imagine, if your line has 1000 pixels in it, then the correct combination of m and c will have 1000 votes.

The combination of m and c which has the most votes is what is returned as the parameters for the line.

What is Hough transform?

Hough transform is a feature extraction method for detecting simple shapes such as circles, lines, etc in an image.

Detecting lines using OpenCV

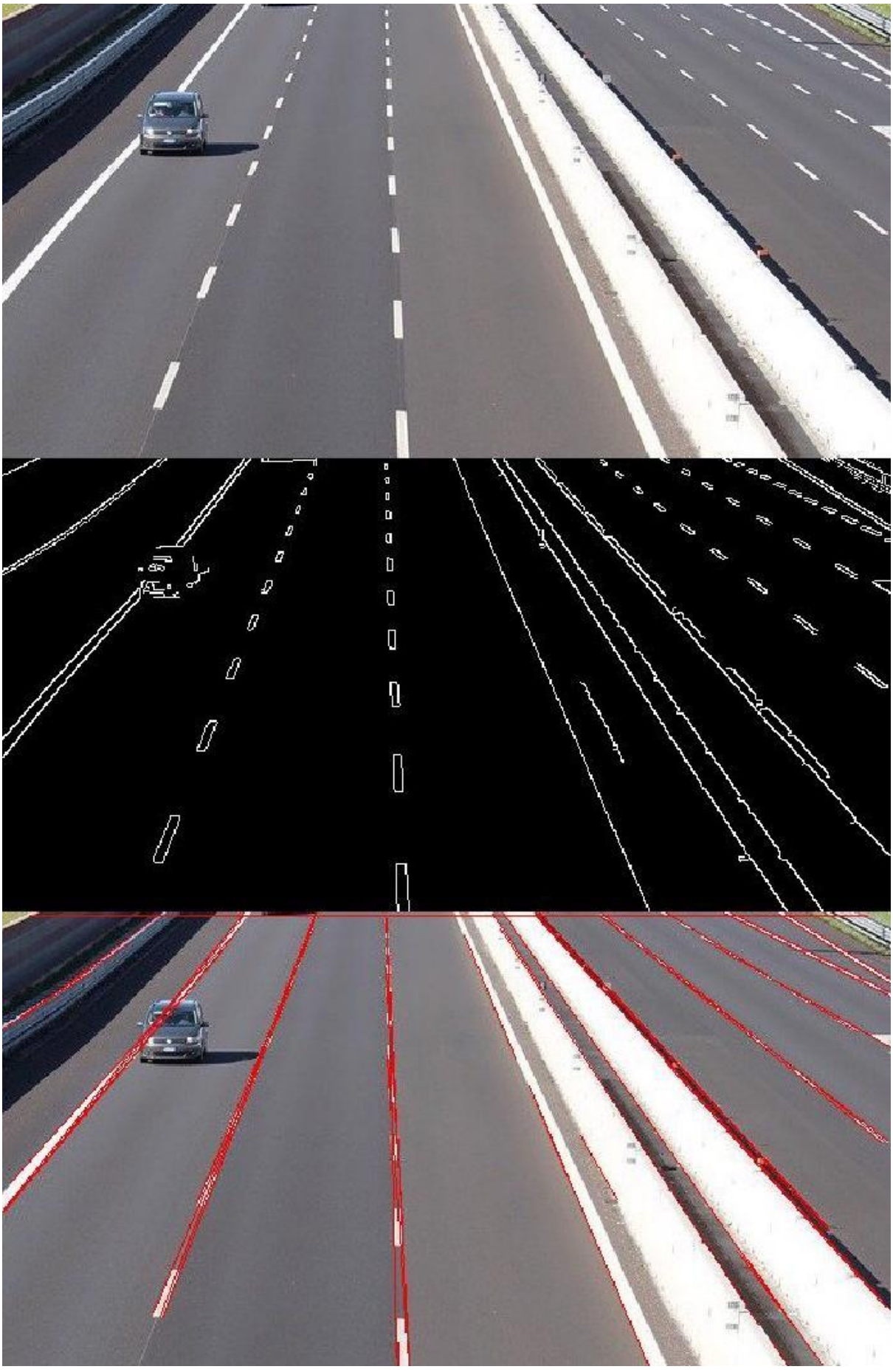
In OpenCV, line detection using Hough Transform is implemented in the functions HoughLines and HoughLinesP (Probabilistic Hough Transform). We will focus on the latter.

The function expects the following parameters:

- image: 8-bit, single-channel binary source image. The image may be modified by the function.
- lines: Output vector of lines. Each line is represented by a 4-element vector (x_1, y_1, x_2, y_2) , where (x_1,y_1) and (x_2, y_2) are the ending points of each detected line segment.
- rho: Distance resolution of the accumulator in pixels.
- theta: Angle resolution of the accumulator in radians.
- threshold: Accumulator threshold parameter. Only those lines are returned that get enough votes
- minLineLength: Minimum line length. Line segments shorter than that are rejected.
- maxLineGap: Maximum allowed gap between points on the same line to link them.

Example:

```
# Read image
img = cv2.imread('lanes.jpg', cv2.IMREAD_COLOR)
# Convert the image to gray-scale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Find the edges in the image using canny detector
edges = cv2.Canny(gray, 50, 200)
# Detect points that form a line
lines = cv2.HoughLinesP(edges, 1, np.pi/180, max_slider, minLineLength=10,
maxLineGap=250)
# Draw lines on the image
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 3)
# Show result
cv2.imshow("Result Image", img)
```



Line Detection Example

It is very important that we actually use an edge only image as parameter for the Hough Transform, otherwise the algorithm won't work as intended.

Detecting circles using OpenCV

The process goes about the same as for lines, with the exception that this time we will use a different function from the OpenCV library. We will use now HoughCircles, which accepts the following parameters:

- image: 8-bit, single-channel, grayscale input image.
- circles: Output vector of found circles. Each vector is encoded as a 3-element floating-point vector (x, y, radius) .
- circle_storage: In C function this is a memory storage that will contain the output sequence of found circles.
- method: Detection method to use. Currently, the only implemented method is CV_HOUGH_GRADIENT , which is basically 21HT
- dp: Inverse ratio of the accumulator resolution to the image resolution. For example, if dp=1 , the accumulator has the same resolution as the input image. If dp=2 , the accumulator has half as big width and height.
- minDist: Minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.
- param1: First method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the higher threshold of the two passed to the Canny() edge detector (the lower one is twice smaller).
- param2: Second method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.
- minRadius: Minimum circle radius.
- maxRadius: Maximum circle radius.

Remember the parameters needs to be different, as we can't describe a circle with the same parametrization we used for lines, and instead, we need to use an equation like:
(x - x0)^^2 + (y - y0)^^2 = r^^2

Code:

```
# Read image as gray-scale
img = cv2.imread('circles.png', cv2.IMREAD_COLOR)
# Convert to gray-scale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Blur the image to reduce noise
img_blur = cv2.medianBlur(gray, 5)
# Apply hough transform on the image
circles = cv2.HoughCircles(img_blur, cv2.HOUGH_GRADIENT, 1,
img.shape[0]/64, param1=200, param2=10, minRadius=5, maxRadius=30)
# Draw detected circles
if circles is not None:
    circles = np.uint16(np.around(circles))
```

```
    for i in circles[0, :]:
        # Draw outer circle
        cv2.circle(img, (i[0], i[1]), i[2], (0, 255, 0), 2)
        # Draw inner circle
        cv2.circle(img, (i[0], i[1]), 2, (0, 0, 255), 3)
```

Note that compared to the previous example, we are not applying here any edge detection function. This is because the function HoughCircles has inbuilt canny detection.

And the result:

Circle Detection Example

Conclusion

Hough Transform is an excellent technique for detecting simple shapes in images and has several applications, ranging from medical applications such as x-ray, CT and MRI analysis, to self-driving cars. If you are interested in knowing more about Hough space, I recommend that you actually run the code, try different configurations by yourself, and that you check out the OpenCV documentation for additional information.