Hough Transform and Shapes

Rosa Yuliana Gabriela Paccotacya Yanque

1 Fitting Lines

Once we have the contours of an image, what follows is to detect and match lines to the points in an image. There are many techniques to extract linear descriptors from the curves in an image. One is the *successive approximation* (Fig. 1) that takes two endpoints and join them with a line, then recursively divide till having polylines vertices that matches the curve. In case a smoother representation is needed a interpolation could be done. However, real-world lines are more complex. Another technique is the Hough Transform that will be detailed below.

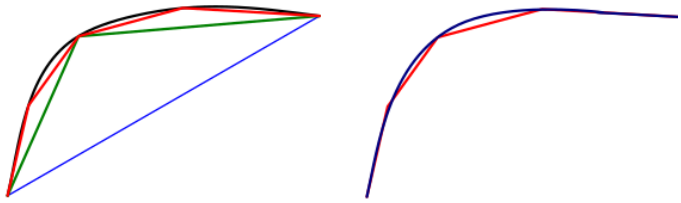

Figure 1. Successive Approximation. At the left, in blue the first step:joining to endpoints, and then in red and green, its division recursively. At the right, in red the last polyline generated an in blue an interpolation is done

2 The Hough Transform

The Hough Transform(HT) is a general procedure that fits a structure to a set of tokens (it can be a set of pixels to curves of a specified shape e.g. a line to a set of points, circles to points in the plane or spheres or ellipsoids to 3D data). Basically what it does is "to record all the structures on which each token lies and then look for structures that get many votes" [1].

For example in its simplest case, if we would like to group points that lie on lines, we take each edgel(a point in a recognized edge) and calculate all possible lines that could pass through it, then we repeat this process for each point keeping a register of lines and counting when a line is repeated. In the end, the line or lines that are present will be the ones with more votes.

To explain more detailed how the Hough Transform works [2], we will continue with the simplest case, detecting straight lines. But, before we can vote for a line, we should find an adequate way to represent it. When we use a space based on parameters each point on it corresponds to a line in the xy-space A line in a xy-space.

A line is represented by the general formula:

$$y_1 = ax_1 + b \tag{1}$$

being the parameters a(slope) and b(bias). So, let's change the equation to be based on these parameters $b = -ax_1 + y_1$. In (Fig. 2), we see the representation of a line having two points (x_i, y_i) and (x_j, y_j) in xy-space and its consequent representation in ab-space, in the parameterized space we see there is a point (a', b') of intersection. This point represents the values for a(slope) and b(intercept) in the general equation in a xy-space for both points, moreover, for all the lines generated from the points in the line in xy-space are going to be intersected in the same point (a', b') in the parameterized space.

The disadvantage of this new space(slope and bias) is that it is not limited, because when we have vertical lines, the value for the slope tends to infinity, and in computers, we can't store infinity values. Therefore, a more suitable space will be one based on the normal representation of the line:

$$x\cos(\theta) + y\sin(\theta) = \rho \tag{2}$$

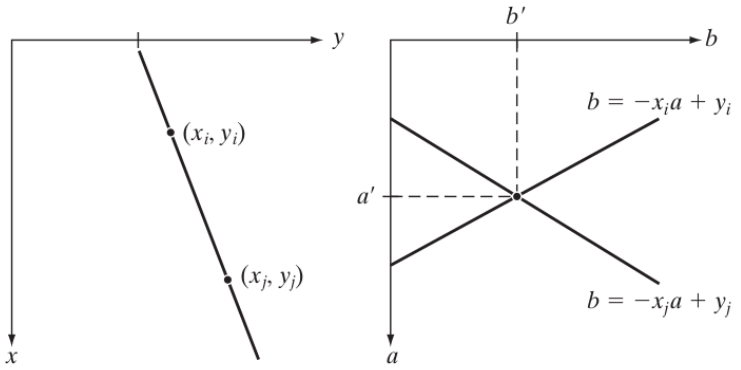

Figure 2. Left: points in a xy-space. Right: points in a parameterized space. Source:[2]

Figure 3 exemplifies how we can get these values from the xy-space, and its representation in the new space based on its parameters, here the sinusoidal curves $x_i \cos(\theta) + y_i \sin(\theta) = \rho$ and $x_j \cos(\theta) + y_j \sin(\theta) = \rho$ are composed of points where each one represents a possible line in any direction from x_i, y_i and x_j, y_j respectively. So, we know these two points belong to the same line and we can confirm it in the $\rho\theta$ space where both curves intersect at the point (ρ', θ'), being this the one that represents the line in xy-space. Lastly, we see the matrix accumulator for the parameters. The values for each axis are determined by the subdivision values for θ.

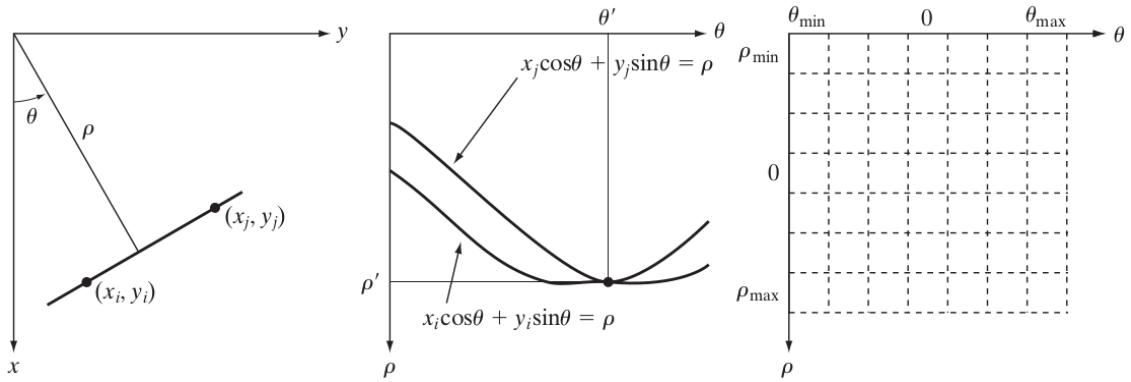

Figure 3. Left: Parameterization in the xy-space. Middle: Sinusoidal curves in the $\rho\theta$ space. Right: Accumulator(matrix). Source:[2]

j

Then the algorithm is as follows [3]:

1. Get an edge image
2. Discretize the space parameters in finite intervals to create the accumulator M.
3. Initialize the accumulator in zero.
4. For each edgel(x,y), calculate the ρ and θ values that satisfies the line.
5. Increase the accumulator in M(ρ,θ).
6. When finishing passing through all the edgels an calculating the parameters for each one, we can find in the accumulator matrix the max values(peaks) that suggests a strong evidence of lines in the image

A technique to reduce the parameters numbers is using the gradient information(direction of the borders).Figure 4 shows how the Hough Transform performs detecting lines in some real samples.

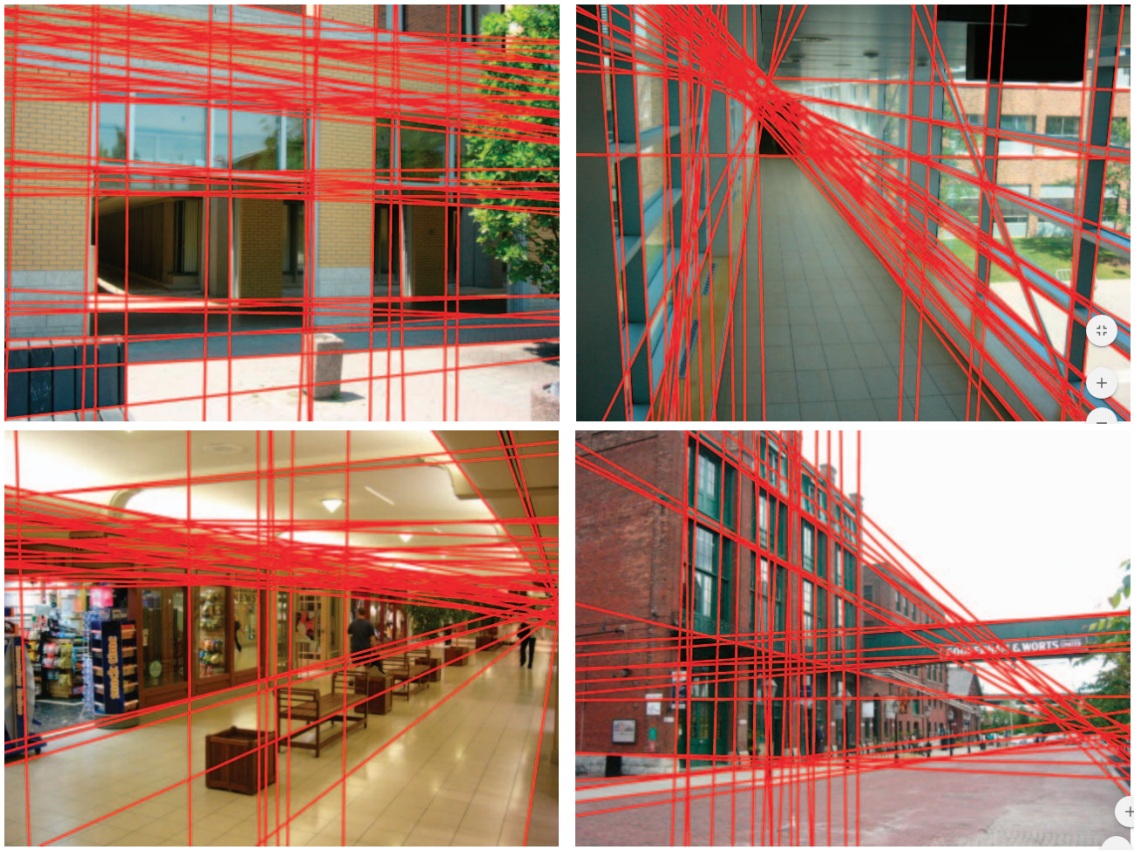

Figure 4. Detecting lines in real scenarios Source:[4]

The Hough Transform can map any shape(circles, rectangles, ellipsoids,cars, etc.) that is parameterized. Moreover, the generalized HT proposed by Ballard in 1981 can work even with shapes that does not have an equation that describes it.

There are several approaches based on HT because of its main advantages[4]:

- Flexibility: tokens can be edgels, interest points, image patches, or image regions.
- Simplicity of the learning procedure.

2.1 What could happen when using the Hough Transform?

There are some problems that could happen when working with the Hough Transform:

- Dimension for the accumulator array increases as we change the structure to fit e.g. when we fit lines to points it has dimension two, but for circles to plane is 3D, moreover, for general ellipsoids in 3D, 10 dimensions are needed
- Grid size for the accumulator, it is difficult to guess the change rate for the parameters
- It can generate structures from points that do not belong to an edge like noise or texture pixels due to the Hugh Transform connects broken up tokens that are close to some structures

However, these problems can be avoided for example using smooth filters to reduce the number of tokens unneeded like textures or lightning to ensure high-contrast edges.

2.2 Real Life Applications

The Hough Transform is applied to solve some vision problems[5]:

- Vanishing points(where parallel lines converge) to calibrate a camera
- 3-D Object Detection
- Motion detection, where vectors of direction are formed
- Automatic recognition in medical imaging, e.g. detect diastolic and systolic diameters of the carotid artery
- Biometric Authentication
- Remote sensing, to recognize roads, rivers, railways, hedges or forest rides
- Robot Navigation where the position and the orientation of a mobile robot are estimated
- Optical Character Recognition for image processing, e.g. to detect text lines on hand-written pages

3 Other Algorithms for Fitting Lines

There is another algorithm that also makes lines detection, the the RANdom SAmple Consensus (RANSAC) algorithm, this chooses randomly pairs of edgels to form a hypothetical line and counts how many other edgels could pass through this line. then, the winner lines are the ones with more matching edgels.

The literature [6] mentions there is not a consensus about which algorithm works best because it is going to depend on the problem we are working on and that it is a good idea to test all the approaches and select the one who has a better performing.

References

[1] D. A. Forsyth and J. Ponce, "A modern approach," *Computer vision: a modern approach*, 2003, Second Edition.

[2] R. C. Gonzalez and P. Wintz, "Digital image processing," 2008, Third Edition.

[3] H. Pedrini and W. R. Schwartz, *Análise de imagens digitais: princıépios, algoritmos e aplicações*. Thomson Learning, 2008.

[4] O. Barinova, V. Lempitsky, and P. Kholi, "On detection of multiple object instances using hough transforms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 9, pp. 1773–1784, 2012.

[5] P. Mukhopadhyay and B. B. Chaudhuri, "A survey of hough transform," *Pattern Recognition*, vol. 48, no. 3, pp. 993–1010, 2015.

[6] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.