ECE 661
Purdue Univer.
Avi Kak
Lecture 20

Hough Transform for Extracting Low-Level Features in Images

Reference: Kosaka and Kak, "Fast Vision-Guided Mobile Robot Navigation using Model-Based Reasoning and Prediction of Uncertainties," CVGIP-Image Understanding, 1992

- Our indoor environments tend to be rich in straight-line features. Hallway corners and the outlines of doors, windows, and picture frames are some of the sources of such features in the camera images of indoor scenes.
- Therefore, it should not be surprising that straight-line features have played a significant role in the more successful vision-based frameworks used by indoor mobile robots for self-localization and for navigation.
- As one example of how such frameworks work, the robot extracts the approximately vertical (remember the perspective effects) straight-line features from the images and compares them to an expectation map of such features that is derived from a geometric model of the environment. The robot knows roughly where it is through the use of odometry. However, the longer the robot relies on just odometry, the greater the error in its sense of where exactly it is located. [This is particularly the case if the robot must turn frequently to avoid collisions with people and obstacles.] Comparing the line features extracted from the images with the same features in the expectation map allows the robot to localize itself exactly.
- Using straight-line features for the purpose mentioned above requires an algorithm that can extract such features reliably and efficiently from images. Hough Transform has emerged as the algorithm of choice for that.

Hough Transform – The Basic Idea

- Let's say that an image contains some <u>disconnected</u> pixels on a line that can be described by the following equation:

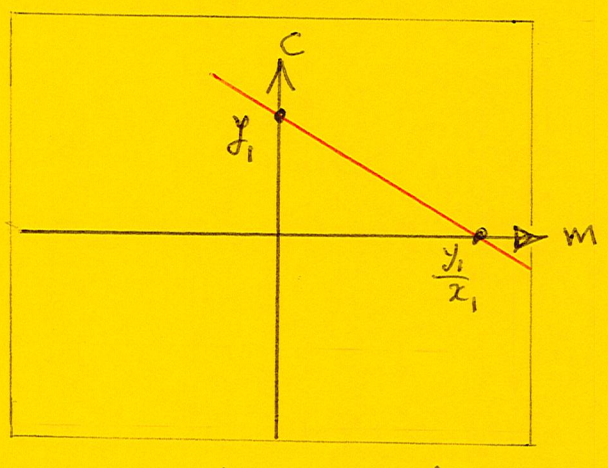
$$y = mx + c$$

where (x, y) represent the coordinates of a pixel on the line.

- For a specific pixel (x_1, y_1) on this line, we can rewrite the above equation as

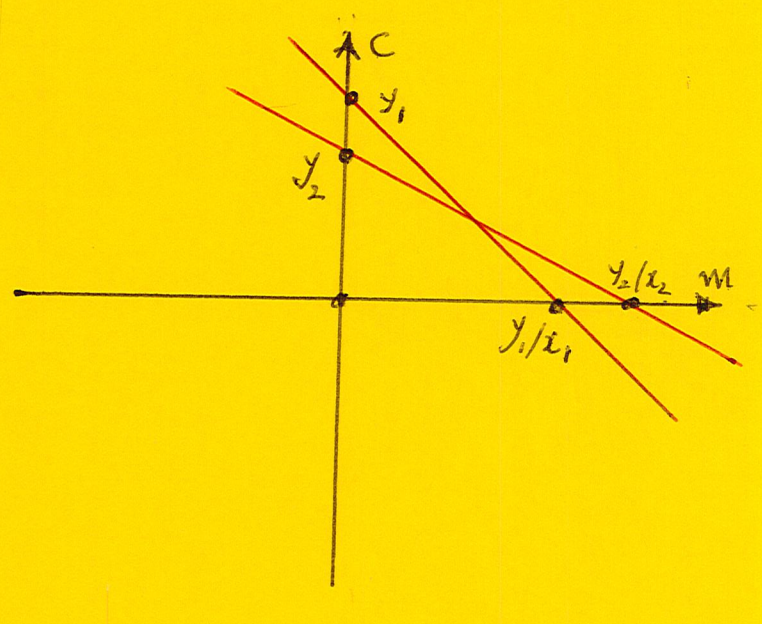
$$c = -x_1 m + y_1$$

Since (x_1, y_1) are fixed, this defines a line in the parameter space (m, c):



What the line in the parameter space tells is that every straight line through the pixel (x_1, y_1) in the image will have its (m, c) values constrained as shown by the red line at left.

- Now consider another pixel (x_2, y_2) on the same line in the image. The (m, c) values for all possible straight-line features through this pixel will also fall on a line of their own in the parameter space. As a result, we will now have two liner in the parameter space:



- The depiction at right speaks to the following approach for detecting straight-line features in an image even if the pixels on the lines are not connected:

① For the purpose of feature extraction assume that you have applied some sort of an edge detector (which could be a Laplacian operator) to the image and binarized its output;

② Divide the parameter space (m, c) into bins;

③ In a raster scan of the binarized image in Step 1, when you encounter a non-zero pixel, increment the bin counts in all of bins that correspond to that pixel. [These would be along the sort of red lines shown above];

④ Threshold the 2D parameter-space histogram thus constructed for discovering the straight-line features in your original image.

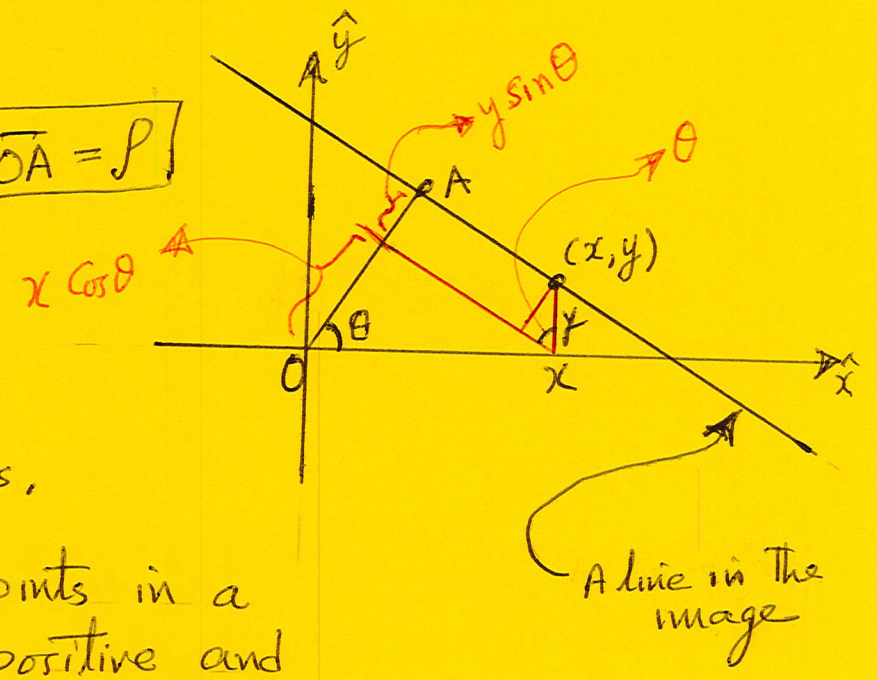
- The parameter space (m, c) is convenient as a first introduction to the Hough Transform. But it cannot be used in an actual implementation of the algorithm because the m values can become arbitrarily large for image lines parallel to the image y-axis.

Using (ℓ, θ) Space for Hough Transform

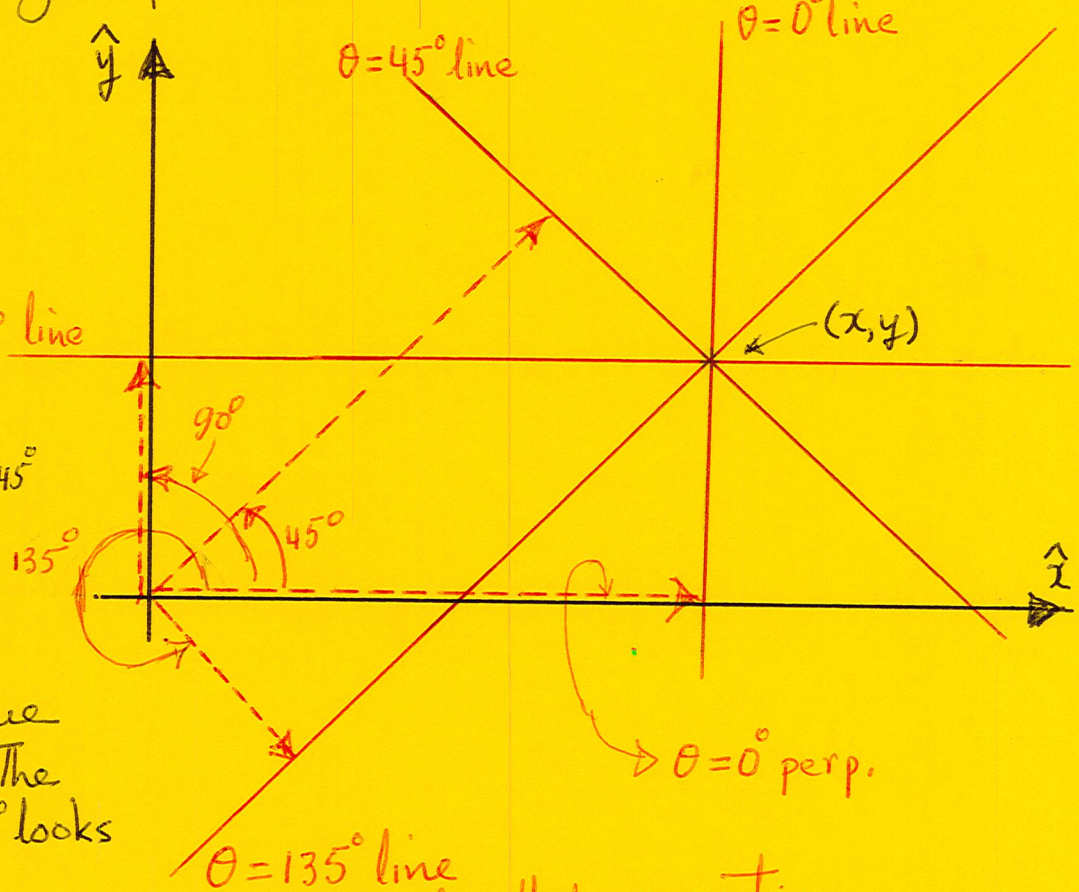
- As mentioned at the bottom of the previous page, an (m, c) based implementation of the Hough transform is not likely to be very useful.
- What works is using the (ℓ, θ)-parameter space for a polar coordinate representation of the lines. As shown, we can write for a line

$$\ell = x \cos\theta + y \sin\theta$$

$\overline{OA} = \ell$

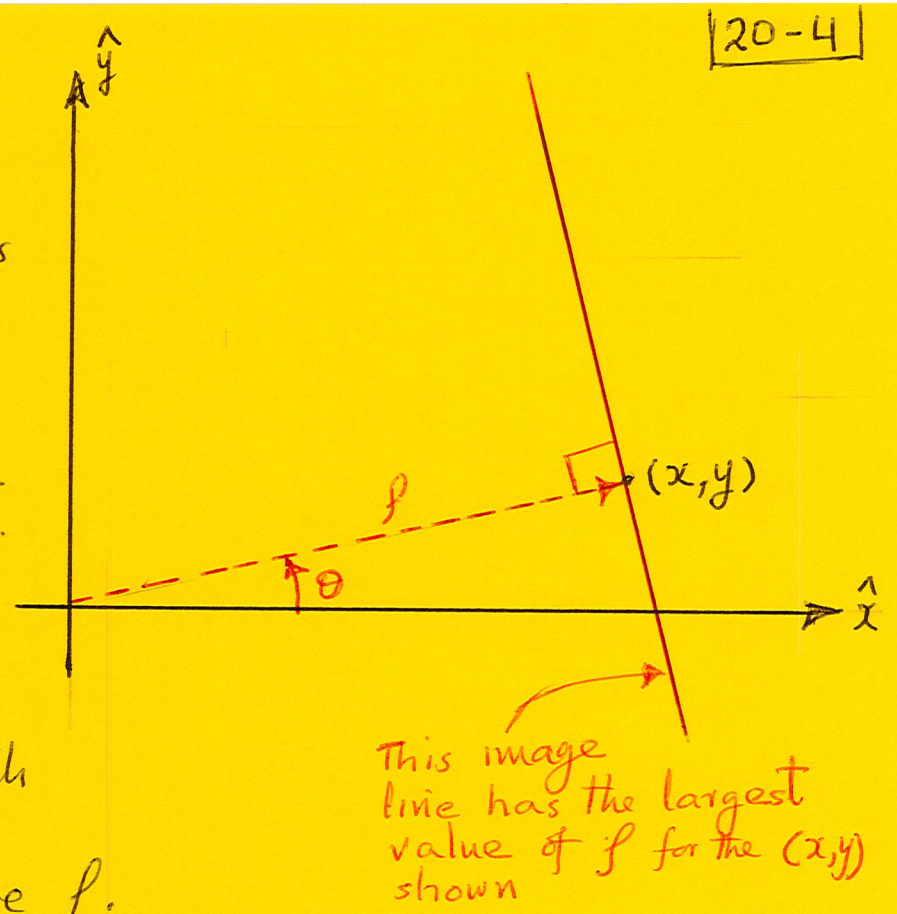


- Note that the more commonly known polar representation (ℓ, θ) for points in a plane is NOT the same as the (ℓ, θ) representation for the lines.
- When we use (ℓ, θ) representation for points in a plane, the radial distance ℓ is always positive and θ goes typically from 0 to 2π or from $-\pi$ to π.
- To see the range of values for θ in the (ℓ, θ) representation of a line, we need to visualize all the lines that can pass through a given pixel (x, y) in the image plane.
- For the pixel (x, y) shown at right, it is easy to visualize the image lines for $\theta = 0^\circ, 45^\circ$, and 90°. For all these three lines, ℓ is positive. When $\theta = 0^\circ$, $\ell = x$, and when $\theta = 90^\circ$, $\ell = y$. When $\theta = 45^\circ$, $\ell = x \cos 45^\circ + y \sin 45^\circ$



- At around $\theta = 120^\circ$, the value of ℓ will be close to 0. And when θ increases beyond that, the value of ℓ will become negative. [The image line that is shown for 135° looks like it should be for 315°.]
- From the construction shown it is clear that θ in the semi-closed interval $[0, \pi)$ will cover all possible lines passing through the pixel (x, y). While ℓ will be positive (or zero) for some of these image lines, it will be negative for others. We can make the same claim for θ in the interval $[-\pi/2, \pi/2)$.
- That leaves open the question of what range to use for the ℓ values.

- To get a fix on the range of values to use for ρ, for the choice of the pixel (x,y) shown at right, notice that ρ takes on its largest value for an image line that is perpendicular to the line joining the origin with that pixel. The value of ρ_{max} for this pixel is $\rho_{max} = \sqrt{x^2+y^2}$.



- As for the smallest value for ρ, it will obviously be negative. And there will exist at least one pixel for which the magnitude of the negative ρ will be as large as the largest positive ρ.

- Based on the above observations, we set the range of values to use for ρ to $[-\rho_m, \rho_m]$ where $\rho_m = \sqrt{x_{max}^2 + y_{max}^2}$

- To summarize: ① For detecting the straight-line features in an image, we first apply an edge detector to the image. Usually the Canny operator gives the best results. If necessary, we then binarize the edge image. ② Subsequently, we declare a 2D array of accumulator cells for the purpose of constructing a 2D histogram in the (ρ,θ)-parameter space. ③ We set the range of ρ and θ values as discussed above. ④ We raster scan the binarized image and and increment the bin counts in the 2D histogram as dictated by the coordinates of each non-zero pixel encountered. ⑤ The histogram thus constructed is called the **Hough Transform** of the edge image. We scan the Hough Transform and locate all its **local maxima**. The (ρ,θ) values where these maxima are located correspond to **ALL** the straight-line features in your original image.

- The last step mentioned above — finding the local maxima in the (ρ,θ) histogram — is more **difficult than it sounds**. Histograms are always noisy. So the local maxima must be located in the presence of that. What adds to the difficulty is the 2D nature of the search.

Useful → When noise is not a big issue, you can call `argsort()[-N:]` on the flattened version of your histogram array for the index values for N largest peaks

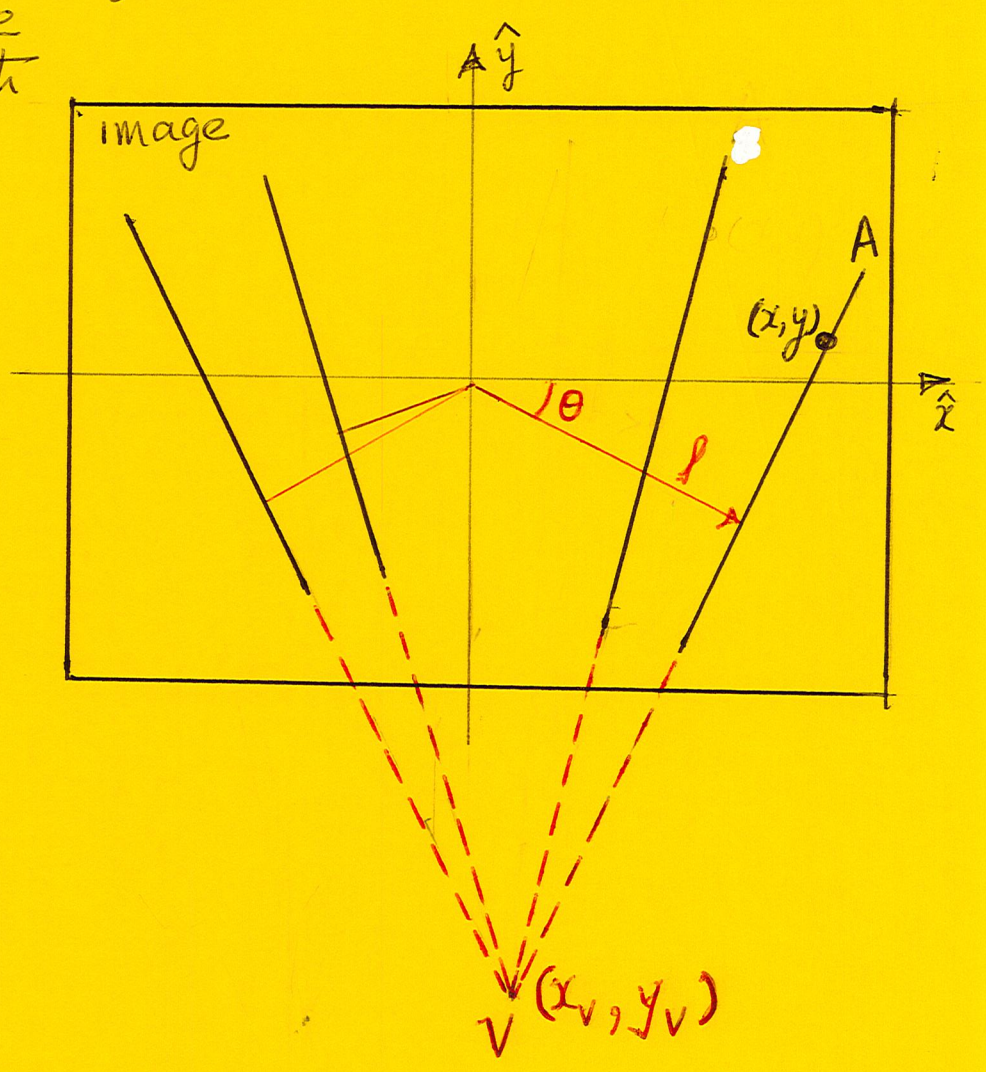
Speeding Up Hough for Real-Time Indoor Mobile Robot Vision

- For indoor mobile robotics, it is possible to convert the 2D search for the peaks in the (ρ,θ)-histogram into a 1-D search. At any position of the robot in, say, the hallways of a building, the images of all the vertical lines in the physical space around the robot will meet at a **vanishing point** (x_v, y_v) in the

imaging plane of the camera. This is shown in the figure below.

- Consider the image line represented by the line VA in the figure. This line will be represented by the cell (ρ,θ) in the Hough space, with ρ and θ as shown. Every pixel (x,y) on line VA will satisfy

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



- As you would expect, this equation will be satisfied by the vanishing point also. Therefore, the (ρ,θ) parameters for line OA must satisfy:

$$\rho = x_v\cos\theta + y_v\sin\theta$$

- Keeping in mind the fact that the perspective effect shown in the figure is greatly exaggerated, the values of θ for the different vertical lines, although different, will all be close to zero. Therefore, we can write

$$\rho \simeq x_v + y_v\theta$$

which is a straight line in the (ρ,θ) parameter space.

- The observation made above suggests the following approach for constructing a 1-D histogram for detecting the vertical lines in the physical space around the robot:

① In a raster scan of the edge image, when we encounter a non-zero pixel at coordinates (x,y), with the assumption that the pixel falls on a line that is the image of a vertical line in the scene, we calculate the θ associated with that putative image line by

$$\theta = \tan^{-1}\frac{y-y_v}{x-x_v} + \frac{\pi}{2}$$

which follows from the geometry of the lines shown in the figure above.

② In this manner we construct a θ-histogram from all the non-zero pixels in the edge image.

③ We retain the N bins with the highest counts.

④ We associate with each θ bin its ρ value as given by

$$\rho = x_v + y_v\theta$$

- The vanishing point coordinates (x_v, y_v) can be thought of as the calibration parameters. You estimate them in advance.

- See Section 6 of the Kasaka & Kak paper [PDF pages 23 through 27] for a detailed explanation of how to make the Hough extraction of line features fast in the context of indoor mobile robotics

Hough Transform for Extracting Circles in Images

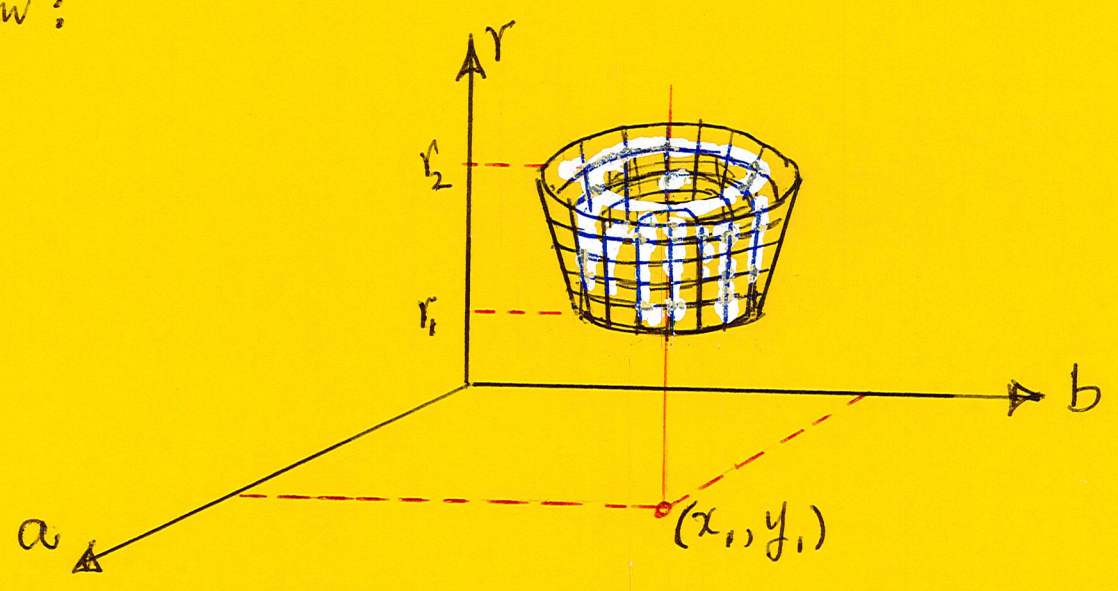
- The Hough method can also be used to extract circles of arbitrary radius in images
- A circle of radius r that is centered at the pixel coordinates (a, b) is described by

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

- Since we have 3 unknowns – a, b, r – we must allow each edge pixel in a raster scan to vote for the bins in a 3-dimensional parameter space whose axes stand for a, b, and r.
- Let's say an edge pixel is at the coordinates (x_1, y_1). It will cast a vote for the bins in the (a, b, r) space that satisfy the equation:

$$(a-x_1)^2 + (b-y_1)^2 = r^2$$

- If all the ~~xx~~ circles you want to extract have their radii between r_1 and r_2, then the pixel at (x_1, y_1) will cast its vote for the bins as shown below:



Generalizing Hough to the Extraction of Conics in Images

- If you wanted to use the Hough method to extract conics of arbitrary shape and at arbitrary locations from an image, you would need to use the following equation that describes conics in general:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$

- Now each pixel you encounter in the edge image will need to cast its votes in a 6-dimensional parameter space.
- Let's say your raster scan sees a pixel at (x_1, y_1). Now you must search through all possible ranges for each of the six parameters in order to locate the bins that would need to be incremented. As you can imagine, this can be computationally very expensive.