Hough Line Transform:

* Technique used to detect features in an image (lines, shapes, etc.)
* Best performance is after pre-processing (edge-detection -> canny edge detection)
* How it works:
  + A line can be represented with the equation ; therefore, a point can also represent a line (in the appropriate plane).
  + Similarly, a line (call it *line1*) can be represented in polar coordinates with the equation ; consider a *line2* that is perpendicular to *line1* and passes through the origin. ρ represents the (perpendicular) distance from the origin to the intersection of *line1* and *line2* and θ represents the angle formed with the x-axis+ and *line2*. This way, a line can also be represented as a point (ρ,θ) (in Hough space).
    - We will stick with polar notation because unlike the equation in the Cartesian plane, this equation can also parametrize vertical lines.
    - *line2* is called the *normal* line
    - It’s important to remember that we are searching for the parameters of a line; we do not know (ρ,θ), but we do know all the pairs.
  + Given a random point , we can plot the curve in the Hough plane (ρ,θ). This curve (a sinusoid graph) represents the infinite number of lines that pass through point because each point on this line is of the form (ρ,θ), which itself parametrizes a line.
  + Plotting multiple pairs in the Hough space will create a respective number of curves. The intersection point of these curves (if any) are the parameters of a line which crosses through all the pairs. The number of curves required to cross in order to classify the intersection point as a desirable “line” is called the *threshold value*.
    - It is very unlikely all the curves of the edge pixels graphed will intersect at a point. As a result, we create a value which can be translated as, “if this number of curves intersect, we can assume this line probably exists in the image.”
  + **An important note:** only edge points are plotted in the Hough space; in other words, only pixels of interest (which can be acquired after edge detection) are being used in this algorithm.
  + The above points can be implemented using an accumulator. An accumulator can be described as a matrix which can classify a *finite* number of lines, depending on its dimensions, ρ and θ. Usually, θ is between [0, π] radians and ρ is [-d, d], where d is the length of the image’s diagonal. For each edge pixel, a ρ value is calculated for some θ value and then the corresponding (ρ,θ) cell is incremented. We iterate through all θ values for this edge point.
  + After iterating through all edge points, we look at our accumulator. All cells with a value above the *threshold* are considered lines.

Example:

Given the line , let’s see if we can generate this equation from some points on the line. We will use the following three pairs of points:

Let’s now create our curves in polar form:

* ⇒
* ⇒
* ⇒

The curve contains all the parameter pairs (ρ, θ) of lines which pass through the point . As a result, the intersection point of these 3 curves will give us a parameter pair for a line which passes through all 3 points. Because all the points came from the same line, we know *all* the curves will intersect at points (plural because they are periodic functions), but generally, we use some *threshold value* to determine a sufficient number of intersecting curves. We restrict our answers to the interval [0,2π].

**Intersection Point:**

(radians)

We now have the parameters of the line of interest in polar form. To find the slope, we can use the following equation:

⇒

Remember, this is the slope of the *normal* line. Because the *normal* line is perpendicular to the original line, the slope of the original line can be easily computed. With this, we almost have the entire equation of the original line:

To find , we can use ρ. ρ is the distance from the origin (0,0) to the intersection of the *normal* line and the original line. We can write the intersection point as ; therefore,

We can write the intersection point in a different manner with only one variable; because the *normal* line passes through the intersection point, the y-coordinate can be rewritten as (the equation of the *normal* line is ). As a result,

We have two values for the x-coordinate of our intersection point, either in quadrant 1 or in quadrant 2. Since , we know our intersection point must be in quadrant 2. Therefore, . Using the *normal* line equation, we obtain . The intersection point is .

With this information, we can complete the equation of the original line:

Equation of original line:

This is the line we used to generate our 3 points.

Congrats!!!