Introduction

- We’ll get to edge detection in more detail in the future…

- For now, let’s assume you have an edge image
  - Binary image:
    - 0 = not edge
    - 1 (or 255) = edge point
Example of Edge Image
How Do We Find Straight Lines?

**Brute force approach:**
- For every pair of points, find all subsets of points close to the line formed by those points

- So: \( n(n-1)/2 \sim n^2 \) lines to check
  - \( (n)(n(n-1))/2 \sim n^3 \) comparisons of every point to all lines

- There is a better way, however…
HOUGH TRANSFORM FOR LINES
Hough Transform Introduction

- Consider a point \((x_i, y_i)\) in an image

- General equation of a line in **slope-intercept** form:

  \[ y_i = ax_i + b \]

- Infinitely many lines pass through \((x_i, y_i)\)
Parameter Space

- Right now, $x$ and $y$ are the variables while $a$ and $b$ are fixed $\rightarrow$ in $xy$ space

$$y_i = ax_i + b$$

- What if $x$ and $y$ were fixed and $a$ and $b$ were the variables? This would be in parameter space:

$$b = -x_i a + y_i$$
Parameter Space

- Every point \((a', b')\) in parameter space is the equation of a single line in \(xy\) space
  - \(a' = \) slope
  - \(b' = \) intercept

\[
y = a'x + b'
\]
Parameter Space

- We said before that there are infinitely many lines (in xy) that go through a given point \((x_i, y_i)\)

- In parameter space, this means that \((a, b)\) varies while \((x_i, y_i)\) remains fixed
  - However, given a value of \(a\), only certain values of \(b\) will work
  - Example: if \((x = 5, y = 3)\), and \(a = 2 \rightarrow b\) has to equal

\[
b = -x_i a + y_i = -(5)a + 3 = -5(2) + 3 = -10 + 3 = -7
\]
So, in parameter space, all of the lines that could go through \((x_i, y_i)\) are represented by a single line in parameter space:

\[
\begin{align*}
    y &= ax + b \\
    b &= -x_ia + y_i
\end{align*}
\]
Quick Recap

- Single line (variable $x$ and $y$)
- Point $(a,b)$ in parameter space
- All the lines going through a fixed $(x,y)$
- Single line in parameter space (variable $a$ and $b$)
Two Points in xy Space

- Given two points \((x_i, y_i)\) and \((x_j, y_j)\) in xy space, only one line goes through them
  - All lines going through \((x_i, y_i)\) \(\Rightarrow\) \(b = -x_i a + y_i\)
  - All lines going through \((x_j, y_j)\) \(\Rightarrow\) \(b = -x_j a + y_j\)
  - Single line in xy space \(\Rightarrow\) one point \((a', b')\) in parameter space

- Intersection of \(b = -x_i a + y_i\) and \(b = -x_j a + y_j\) in parameter space \(\Rightarrow\) line in xy space that goes through both points

\[
egin{align*}
b &= -x_i a + y_i \\
b &= -x_j a + y_j
\end{align*}
\]

\((a', b')\) in parameter space

Line that goes through both points

\(b' = x a' + y\)
Original Problem

- We need to find the lines in the image

- What if we considered a set of possible lines?
  - A set number of values of a and b → a **grid in parameter space**
  - Have a counter for each pair (a,b)
  - If a point in xy space goes through a given line (a,b), increment counter
Problem

- $a$ is the slope of the line $\rightarrow$ approaches infinity as line becomes vertical.

- *Alternative:* use polar coordinates!

\[ x \cos \theta + y \sin \theta = \rho \]
Meaning of \((\rho, \theta)\)

- The meaning of the parameters:
  - \(\rho =\) distance of line from origin
  - \(\theta =\) angle between X axis and line connecting line to origin

\[ x \cos \theta + y \sin \theta = \rho \]
Polar Form vs. Implicit Form

- The vector orthogonal to the line is given by \((\cos \theta, \sin \theta) \Rightarrow \text{line’s NORMAL}\)
- So, we can turn this:
  \[x \cos \theta + y \sin \theta = \rho\]
- …into this:
  \[Ax + By = \rho\]
  \[Ax + By - \rho = 0\]
  \[Ax + By + C = 0\]
- …which is effectively the implicit form of a line!
  - We can also use this form, if \(P = (x,y)\):
    \[P \cdot N - \rho = 0\]
Hough Transform for Lines

- Subdivide $\rho \theta$ parameter space into accumulator cells
  - $-90^\circ \leq \theta \leq 90^\circ$
  - $-D \leq \rho \leq D$
  - $\theta_{\text{inc}} = \text{increment of } \theta$
  - $\rho_{\text{inc}} = \text{increment of } \rho$
- where $D$ = max distance between opposite corners of image
- Initially set all accumulator cells to zero
- For every non-background point $(x_k, y_k)$ in xy plane:
  - Cycle through all values of $\theta$ (incrementing by $\theta_{\text{inc}}$)
    - Solve for $\rho \rightarrow$ round to nearest $\rho$ cell
    - Increment accumulator cell $(\rho, \theta)$
Example: Hough Transform

- Let’s say we have the following 3 points and our accumulator array:
Example: Hough Transform

- \((0,0)\)
  
  \[
  0 \cos \theta + 0 \sin \theta = \rho \\
  \rho = 0 \\
  \]
  
  - \(\theta = 0^\circ\)
    
    \[
    \rho = 0 \\
    \]
  
  - \(\theta = 90^\circ\)
    
    \[
    \rho = 0 \\
    \]
Example: Hough Transform

(1,0)

\[ 1 \cos \theta + 0 \sin \theta = \rho \]

\[ \rho = \cos \theta \]

- \( \theta = 0^\circ \)
  \[ \rho = \cos 0 = 1 \]

- \( \theta = 90^\circ \)
  \[ \rho = \cos 90 = 0 \]
Example: Hough Transform

- $(1, 1)$

\[ 1 \cos \theta + 1 \sin \theta = \rho \]
\[ \rho = \cos \theta + \sin \theta \]

- $\theta = 0^\circ$
  \[ \rho = \cos 0 + \sin 0 \]
  \[ = 1 + 0 = 1 \]

- $\theta = 90^\circ$
  \[ \rho = \cos 90 + \sin 90 \]
  \[ = 0 + 1 = 1 \]
Example: Hough Transform

- Final accumulator →

- Most likely lines:
  - $\rho = 0, \theta = 90^\circ$
    \[
    x \cos 90 + y \sin 90 = 0
    \]
    \[
    y = 0
    \]
  - $\rho = 1, \theta = 0^\circ$
    \[
    x \cos 0 + y \sin 0 = 1
    \]
    \[
    x = 1
    \]
Computational Complexity

- Accuracy determined by size of increments for $\theta$ and $\rho$

- Complexity: linear in $n$ (number of non-background points)
  - Before: $n^3$
Problem with Previous Example

- What about THIS line???

- We didn’t get this line BECAUSE our accumulator array did not have sufficient values
  - Specifically, we would have needed $\theta = 135^\circ$

- Similar problem occurs if we don’t have small enough increments for $\theta$
OpenCV Hough Lines

- Two different versions:
  - HoughLines() → standard Hough Transform
    - Gives $\theta$ and $\rho$ for each line
  - HoughLinesP() → uses more efficient version
    - Also gives endpoints of lines
OpenCV: HoughLines()

- void HoughLines( InputArray image, 
  OutputArray lines, 
  double rho, double theta, 
  int threshold)

  - **image** → input image
  - **lines** → lists of lines → vector<Vec2f>, each with (ρ, θ)
  - **rho** → increment size for ρ in pixels (usually 1)
  - **theta** → increment size for θ in radians (usually CV_PI/180.0)
  - **threshold** → minimum accumulator value

- HoughLinesP() has very similar parameters
Output of HoughLines()
Output of HoughLinesP()
HOUGH TRANSFORM EXTENDED
Hough Transform Extended

• The Hough Transform is applicable to any function of the form $g(v,c) = 0$
  ◦ where:
    • $v = \text{vector of coordinates}$
    • $c = \text{vector of coefficients}$
Hough Transform on Circles

- Given the formula of a circle:
  \[(x - c_1)^2 + (y - c_2)^2 = c_3^2\]

- We can use the Hough Transform to detect circles

- Accumulator cells are 3D now: \((c_1, c_2, c_3)\)
OpenCV: HoughCircles()

- void HoughCircles(InputArray image, OutputArray circles, int method, double dp, double minDist, double param1=100, double param2=100, int minRadius=0, int maxRadius=0)

- **image** ➔ input image
- **circles** ➔ vector<Vec3f> of circles, with (x,y,radius)
- **method** ➔ has to be set to cv::HOUGH_GRADIENT
- **dp** ➔ inverse ratio of accumulator resolution to image resolution (1 = same)
- **minDist** ➔ minimum distance between detected circles
- **param1** ➔ higher threshold for Canny edge detection
- **param2** ➔ minimum accumulator value
- **minRadius, maxRadius** ➔ minimum and maximum radius
Output of Hough Circles