Alona Aliyev

# Finding the longest pair of parallel lines in images

## The Approach:

In image processing theory, to find lines in image, we need to:

1. Find all edges in the image (using the Canny algorithm)
2. Find what edges are lines (using Hough transform)
3. Find pairs of parallel lines and compare their lengths

**Edges:** To understand where the edges are in an image, we need to look how the pixels' colors change. An Edge in an image, is a pixel where the gray value suddenly changes, so we need to detect the changes, and this is done by finding the gradient of the image:

$$the\ gradient\ of\ an\ image: \nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

$$gradient\ magnitute: \left||\nabla f|\right| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

$$therefor, in\ digital\ images, the\ gradient\ of\ a\ pixel\ (x,y)\ is:$$

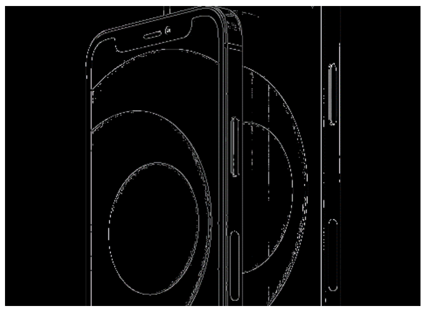
$$\frac{\partial f}{\partial x}[x,y] = f[x+1,y] - f[x,y]\ , \qquad \frac{\partial f}{\partial y}[x,y] = f[x,y+1] - f[x,y]$$

There are a couple of algorithms for edge detecting – Prewitt, Sobel, Laplacian and Canny, I will be using the Canny algorithm since it is the most commonly used and it's highly effective.

The Canny algorithm:

1. **Blur the picture using a gaussian filter** – this cancel noises in the image and smooths the function, the details vary according to $\sigma$.
2. **Calculate the magnitude of gradients** – helps finding where the colors change the most
3. **Preform Non-Maximum suppression** – checks if pixel is local maximum along gradient direction.
4. **Hysteresis Thresholding –** using lower and upper bounds, we include possible edge pixels that are adjacent to edge pixels. If pixel's gradient is lower than the low threshold, than it is not an edge. If is higher than the high threshold, than it is an edge. Otherwise, the pixel's gradient compares to its neighbors, following the assumption that an edge pixel will be connected to another edge pixel.
   - The thresholds' values define how many details will be detected.

As a result, we get a binary image, where white pixel means an edge, and black means not an edge.



**Lines:** To find lines in a binary Image, we use the Hough transform.

In 2d world, every linear line corresponds to a point in Hough space:

$$if\ the\ line\ is\ y = mx + b, then\ the\ point\ in\ the\ Hough\ space\ is\ (m, b)$$

Moreover, every point in the image space corresponds to a line in Hough space.

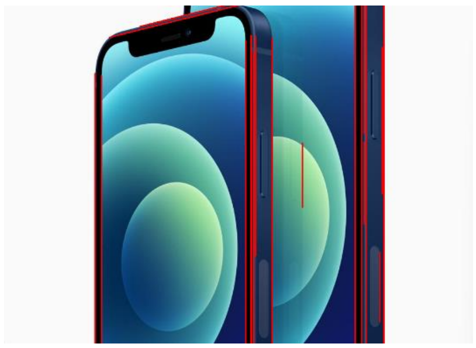
In Hough transform, each line is represented by $(r, \theta)$, where $r$ is its distance from the origin, and $\theta$ is the angle the perpendicular to this line makes with the x axis.

$the\ line\ is\ r = x\ cos\theta + y\ sin\theta$ .

This representation allows to represent vertical lines too (which are a problem in $y = mx + b$ representation).

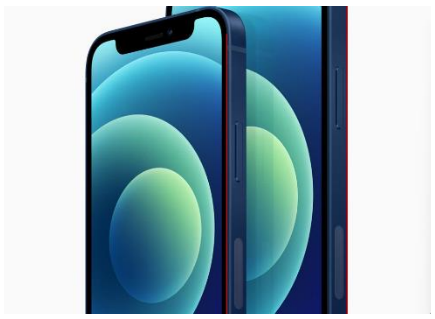
The Hough Transform Algorithm: (works on binary edge image)

1. **Initialize** – define an array H, where rows are r values and columns are $\theta$ values (0-180) and initialize it to zeros.
2. **Voting** – for each edge point $I[x, y]$ in the image:
```
$for\ \theta = 0\ to\ \theta = 180$:
        $H[r, \theta] += 1$
```
3. **Find Maximum** – finding the values of $r$ and $\theta$ with the highest values
4. **Conduct line** – represent the line $r = xcos\theta + ysin\theta$



**Finding parallel lines:** after the Hough transform, we have an array representing all the lines, now we need to find the lines with the same $\theta$ , this will promise us that the lines are parallel.

**Finding the longest pair:** for each pair, we can calculate the length and compare this to the other pairs.



- For this question, I will compare the lengths of the shortest lines in each pair.
- To improve results, we can add Erosion and Dilation to remove more noise and emphasize the lines.

Alona Aliyev

**My Implementation:**

- Using OpenCV, NumPy, matplotlib.pyplot, math
- The values I chose for parameters may change from picture to picture depending on desired results.

1. **Finding Edges:** the function **EdgeDetedtion(image)**:
    - Smoothing the picture, using the function $\boldsymbol{cv2.GaussianBlur(img, kernel\ size, sigmaX, sigmaY)}$, for this implementation, I chose SIGMA = 0.8 and kernel size = [3,3]
    - Finding lower and upper thresholds – we compute the median of the image (np.median) and define lower and upper bounds according to this and the sigma.
    - Appling the Canny algorithm, using the function $\boldsymbol{cv2.Canny(img, lower, upper)}$.

    Return value: a binary image that represents the edges.

2. **Find Lines:** the function **FindLines(binaryImage):**
    - Appling erosion and dilation to the binary image, using the functions $\boldsymbol{cv2.getStructuringElement(shape, size,\ point)}$, $\boldsymbol{cv2.dilate(im, structureElement, iterations\ )}$ $\boldsymbol{cv2.Erode(im, structureElement, iterations)}$ I chose, cv2.MORPH_RECT as the shape, (3,3) as the size, and default value as the number of iterations.
    - Apply the Hough transform, using the function $\boldsymbol{cv2.HoughLinesP(im, rho, theta, threshold, srn, stn\ ),}$ for this implementation, I chose rho – to be a resolution of 1 pixel for r, theta – is from 0 to pi (in radians), threshold – 150 (we need at least 150 votes to be a line) and srn stn to be their defaults values, this function preforms the Hough transform and returns start and end points of each line.
    - Computing the angle each of the lines creates with x-axis using $\boldsymbol{math.atan2}()$, and the line's length using $\boldsymbol{np.linalg.norm}()$
    - Creating an array of all lines, their length and the angle it makes with the x-axis (easier to use latter)

    Return value: an array $[n, 6]$ that represents the lines, columns 0-3 – start point and end point, columns 5 – the angle the line creates with the x-axis, column 6 – line's length.

3. **Finding the longest pair:**
    - Sort the array according to the angle – this ensures that the parallel lines are next to each other.
    - Initialize the chosen longest lines to zeros.
    - For each possible angle (-90,90):

        - Sort the lines with this angle according to length and compare the second longest to the shortest of the chosen lines. If it longer, update chosen lines.
    - If a pair of parallel lines was found, draw it on the image using $\boldsymbol{cv2.line(im, ptr1, ptr2, color, thickness)}$**,** and return image. Otherwise, print message and return 0.

    Return value: an image, where the two longest parallel lines are drawn.

4. **Main:** showing the image, using matplotlib and OpenCV.

Final result: