Convolution, Edge Detection and Hough Transform

Bastien Ruivo
Baptiste GENEST

Abstract

In this report we detail how we implemented convolution, Canny Edge Detection, and an improvement of the classic Hough Line Transform.

Remark. *In the following report, we will use the following notation:* $I(x) = ||\nabla Grayscale(image)(x)||$

1 Convolution

1.1 Basic filters

2 Edge Detection

2.1 Active Contour

The active contour method is an alternative method to contour detection, the main idea is to build a curve that follows the contour of the object. The evolution of the curve is done by trying to minimize the following energy functionnal:

$$E(\gamma) = \int_0^1 ||\dot{\gamma}(t)||^2 + \alpha||\ddot{\gamma}(t)||^2 + \beta V(\gamma(t))dt \tag{1}$$

The first term penalizes length, the second one penalizes curvature. Finally V is a potential function that penalizes the curve for being in a region that is likely not in image contour.

We use :

$$V(x) = \frac{1}{1 + ||\nabla I(x)||} \tag{2}$$

The variationnal gradient of the energy gives the following evolution PDE:

$$-\frac{\partial \gamma}{\partial t} = \ddot{\gamma} + \alpha \ddddot{\gamma} + \beta \nabla V(\gamma) \tag{3}$$

2.1.1 Discretization scheme

We use a finite difference scheme to discretize the spatial derivatives. We use the classic laplacian tridiagonal matrix to discretize the second derivative. And the 5-points stencil to discretize the fourth derivative.

The time discretization is done using the Implicit Euler scheme to ensure stability and larger time steps:

$$(I - dt(L + \alpha F))\gamma^{n+1} = \gamma^n + -\beta dt \nabla V(\gamma^n) \tag{4}$$

We use Eigen sparse matrix structure to store the differential operators and factorize the linear system with sparse LL decomposition once to be able to solve it at each time step efficiently.

Figure 1: Evolution of active contour method

2.2 Canny Edge Detection

2.2.1 Noise reduction

First, we apply a gaussian filter to reduce the noise in the image.

2.2.2 Gradient calculation

We calculate the gradient of the image using the Sobel operator.

2.2.3 Non-maximum suppression

We suppress all the pixels that are not local maximum in the gradient direction. Removing all the pixels that are not local maximum in the gradient direction hence checking:

$$I(x + \frac{\nabla I}{||\nabla I||}) \geq I(x) \wedge I(x - \frac{\nabla I}{||\nabla I||}) \geq I(x) \tag{5}$$

For values of I out of the pixel grid we use bilinear interpolation.

2.2.4 Hysteresis thresholding

To visit efficiently the image, we use a queue to store the pixels that are above the high threshold, then use a BFS to visit all the pixels that are above the low threshold and connected to the pixels in the queue.

To automatically define meaningful and robust thresholds, we use statistical quantiles of the gradient magnitude, namely:

$$\text{low threshold} = \text{quantile}_I(\alpha)$$
$$\text{high threshold} = \text{quantile}_I(1 - \alpha)$$

For typical values of α, like $\alpha = 0.2$. This approach allows to automatically select statistically significant variations.

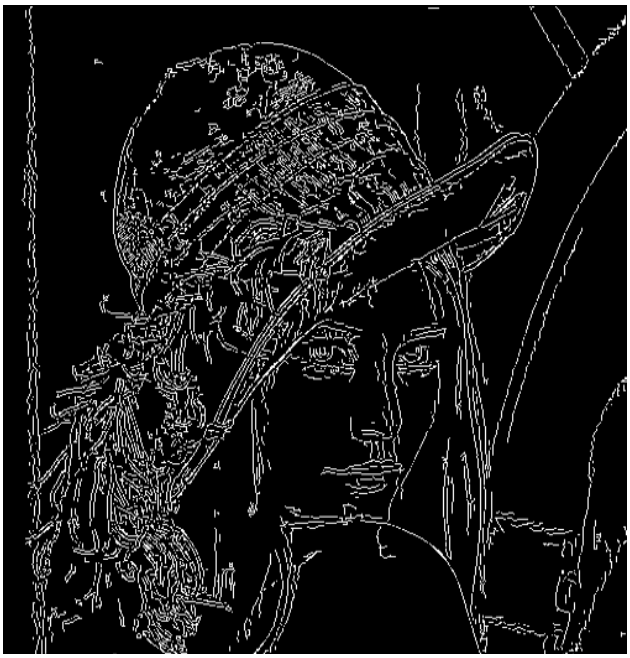

Figure 2: Result of Canny Edge Detection on the lena image

3 Hough Line Transform

3.1 Classic Hough Line Transform flaws

The main arguments we oppose to the classic Hough Line Transform are:

- The need to use a large array for θ and ρ discretization in order to be precise.
- The need to discretize the parameters in itself is a problem, most values of θ and ρ are not relevant.

- With large arrays, local maxima are too numerous and too close to each other, the algorithm is not able to find the best fitting line.

We show how the will to correct such flaws led to an alternative and efficient algorithm.

3.2 Using gradient information and pixel location

First of all, even if the original algorithm uses the norm of the gradient to detect contours, the orientation of the gradient is not used while it carries an important information: if there is a line going through a pixel, it must be orthogonal to the gradient.

Hence instead of making cuntour-likely pixels vote for a line, we consider the following 4D-point cloud:

$$\mathcal{P}_{I^\alpha} = \{(\rho, \theta, x, y) \in \mathbb{R}^4 \mid I(x,y) > \alpha, \theta = \angle \nabla I(x,y), \rho = \cos(\theta)x + \sin(\theta)y\} \tag{6}$$

Keeping the pixel coordinates will be important later to choose the best fitting line and clamping it.

3.3 Density Based Clustering

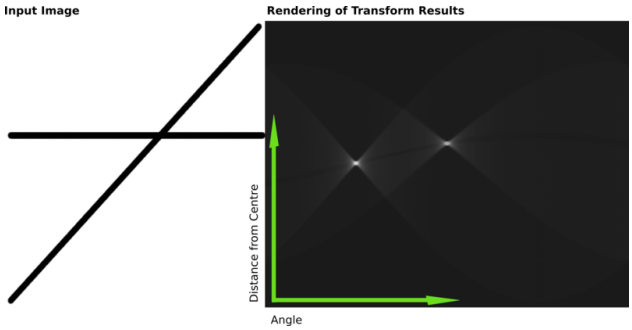

Figure 3: Classic Hough Transform, lines are detected by finding local maxima in the accumulator.

As seen in the above figure, the goal is to find the brightest spots. Storing the votes in the accumulator reduces the problem to finding local maxima but in our case the bright regions corresponds to regions of high density in the line space given by the gradients of the image. Hence, to locate such dense regions we use a density based clustering algorithm : DBSCAN.

We do not describe the algorithm here but note that it relies on two parameters: ε and $minPts$.

The ε parameter is the radius of the ball centered around a point to define its neighborhood, and $minPts$ is the minimum number of points in the neighborhood of a point for it to be considered a core point.

To define a meaningful neighborhood, we need to define a robust distance between two pixel-line pairs.

3.3.1 Distance between pixel-line pairs

The distance we need to define must be 0 when the points describe the same line and the two pixels are on the shared line, and be high typically when the two lines are perpendicular.

We propose the following distance to respect these properties:

$$d((l_1, x_1), (l_2, x_2)) = \frac{||\Pi_{l_1}(x_2) - x_2||^2 + ||\Pi_{l_2}(x_1) - x_1||^2}{(d_1 \cdot d_2)^2} \tag{7}$$

where $\Pi_l(x)$ is the orthogonal projection of x on l and d_i is the direction vector of the ith line.

One can easily check that if $l_1 = l_2$ then $\Pi_{l_i}(x_j) = x_j$ hence $d((l_1, x_1), (l_2, x_2)) = 0$ and if l_1 and l_2 are perpendicular then $d_1 \cdot d_2 = 0$, implying $d((l_1, x_1), (l_2, x_2)) = \infty$.

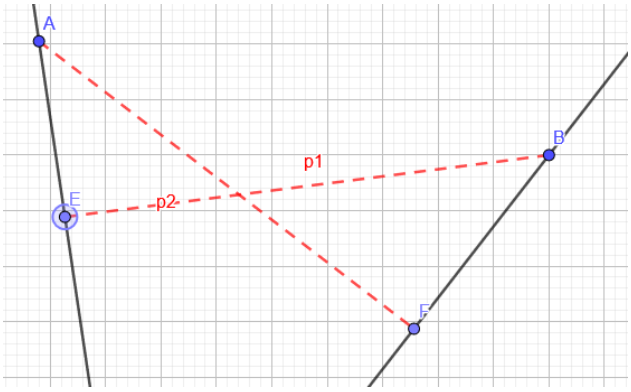

Figure 4: Distance between pixel-line pairs

3.3.2 Heuristic for clustering parameters tuning

We fix the value of ε to a value that seems reasonable, and then we tune find $minPts$ such that $t\%$ of all points have $minPts$ in their ε-neighborhood.

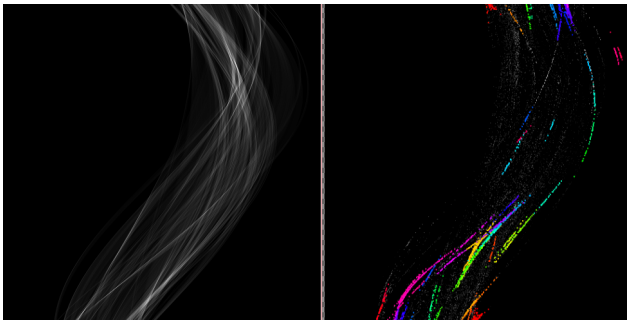

Figure 5: Hough Transform accumulator vs DBSCAN clustering

3.4 Choosing cluster representative

Once all clusters are determined, we need to determine the line that represents each cluster the best. One could try to pick the barycenter of the points in the cluster but its a bad idea since we cannot interpolate angles that way.

3.4.1 Linear regression

Hence we use the fact that we kept the pixel coordinates in the point to determine the best fitting line.

We use PCA to find the least square regression line of the points in the cluster.

Note that also gives a natural way to define a score to measure how good the clustering is:

$$\text{score} = \sum_j \lambda^j_{\text{min}} \tag{8}$$

Where λ^j_{min} is the smallest eigenvalue of the covariance matrix of the pixels in cluster j. If all the points in the cluster are on the same line, $\lambda^j_{\text{min}} = 0$,hence a low score means a good line detection.

3.4.2 Clamping line

Finally we can determine easily when the line starts and ends by clamping the regression line between the two parameters:

$$\begin{cases} t^j_{min} = \min_i\{d_j \cdot x_i \mid (l_i, x_i) \in C_j\} \\ t^j_{max} = \max_i\{d_j \cdot x_i \mid (l_i, x_i) \in C_j\} \end{cases} \tag{9}$$

Where d_j is the direction vector of the regression line of C_j.

Which corresponds to determining the two points that are the border of the projection of the cluster along the regression line.

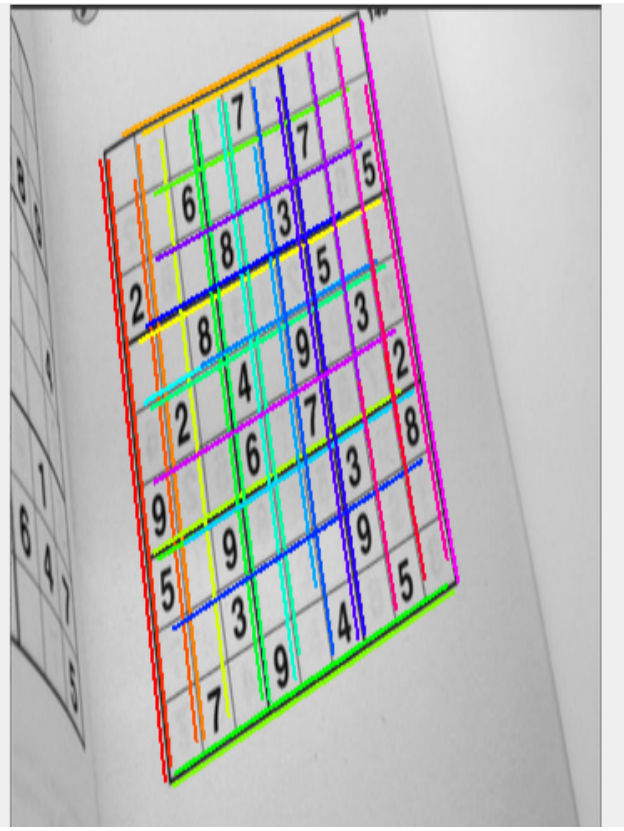

Figure 6: Results on sudoku image

Note that the result heavely depends on the parameters of the algorithm, we can find arbitrary good results by tuning the values of ε and $MinPts$ but the automatic tuning of those parameters proposed here appears to work fine on most images.