Research Front in Computer Science – Lab Report

Submitted by: Ron Ziskind, Roi Tiefenbrunn
Leading Researcher: Jihad El-Sana

URL to project in Git-Hub: Project_Hazit

1. Goal: CV algorithm – Accurately finding the vertices of a rectangle within the image

As Jihad's research focuses on computer vision algorithms and image processing, we were tasked to produce an algorithm that receives an image (jpeg/png) as an input and processes it to return the vertices of a convex quadrilateral within the image.
Optional information: The algorithm may receive, in addition to the image itself, a mask indicating where the quadrilateral's edges are (as in Figs. 1.1 and 1.2).

High-level approach:

1. Compute the image's gradient (and apply the mask if exists)
2. Produce a Hough-Space based on the gradient
3. Based on the Hough-Space's local extrema, pick candidates for the quadrilateral's edges
4. Score the candidates in order to come up with the best edges defining line equations
5. Return lines' intersections

Programming language: Python using opencv2, NumPy libraries

Figure 1.1: Given Image

Figure 1.2: Given mask

2. Detailed explanation

2.1. Computing gradient:

The image's gradient, see Fig. 2.1, details the rate of the change of color within the picture. In order to produce it, we first convert the image to grayscale and pass it through a low pass filter to reduce noise.

OpenCV's GaussianBlur function passes the grayscale image through a low pass filter and the Laplacian function computes the gradient.

Figure 2.1: Fig 1.1's gradient

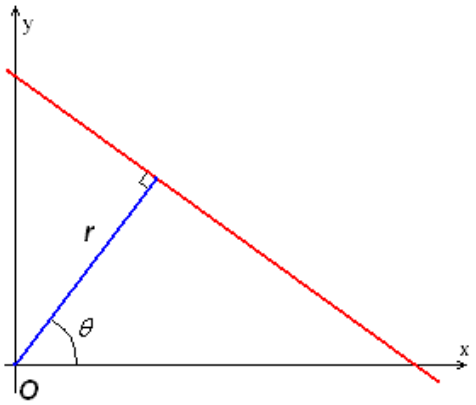


Figure 2.2: r, Θ represent the line in the Hough-Space

2.2. **Computing Hough-Space**:

The Hough-Space gives information about lines within the image space: each dot in the Hough-Space represents a line within the image space, see Fig. 2.2, and each point receives votes as a value to determine which line is a better candidate. As an edge of a quadrilateral produces a drastic change of color in the picture, we expect the gradient at the edge to be significantly higher than at other locations. Therefore, points in the Hough-Space should receive a high amount of votes in proportion to the number of high-value gradients the line they represent passes through.

In order to limit the number of points in the Hough-Space which may receive a value, we base the Hough-Space on the picture's polar space – as the longest radius possible is the image's diagonal and we can quantize the angle to 360 degrees. Two methods of computing the Hough-Space were considered:

2.2.1. <u>Taking every possible line that goes through a point:</u>

The first method was to take every high-enough gradient point and increase votes for every point in the Hough-Space which represents a line that passes through the mentioned gradient point.

As a result, the more gradients are in a line, the higher is the number of votes the line receives, see Fig. 2.3.

Time Complexity: O(n)

2.2.2. <u>Taking a line for every two points (This was suggested by Jihad):</u>

The second method was to take every two high-enough gradient points and increase the vote for the line which passes through them, see Fig. 2.4. This method will drastically increase the number of votes for good candidates.

Time complexity: $O(n^2)$ – the increase in runtime is surely felt even after optimization.

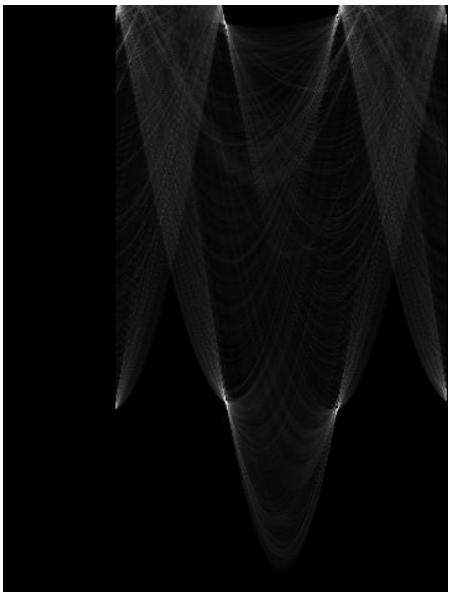


Figure 2.3: Hough-Space by O(n)

Figure 2.4: Hough-Space by $O(n^2)$
(normalized in order to emphasize the difference of votes)

2.3. **Choosing candidates**:

After calculating the Hough-Space, we select the top 20 valued indices. However, these aren't necessary the best lines. Noise-heavy or blurred images may produce many local extrema in the Hough-Space computation. A way to determine the quality of the lines these extrema represent is needed, in order to choose the best possible options.

We came up with several different ways:

2.3.1. By gradient quality:

Sum of high-enough gradient values along the line, normalized by its length.

2.3.2. By density:

A number of low-enough gradient values along the line, normalized by its length. Here, we look for the minimum scores.

2.3.3. By frequency:

As a line that has smaller gaps between high-enough gradient value points is most likely a better line than one with one big gap, this scoring method gives a negative score to a gap proportionally to its length.

The line's score will be the sum of scores of its gaps, normalized by its length.

2.3.4. By histogram of gaps (this method was suggested by Jihad at the end):

Similar to the frequency, we consider a line to be better when its gaps are minimal. In this method, we "punish" the score of the line more aggressively. For a gap of the length of N, we decrease the score by 2^N. Normalized by the length of the line.

2.4. **Choosing the lines out of the candidates**:

The previous step may result with lines which are very similar in nature, basically representing the same edge. It is needed to make sure that the top-score, final lines we pick, in our case 4 lines, represent different edges. For this purpose, we checked the uniqueness of the lines using the next approaches:

2.4.1. By polar representation:

Given two lines are considered not unique if, when in polar representation, their θ's are almost the same (rounded to 3 digits after the decimal point, in radians) and their radii are close enough.

2.4.2. By average distance:

Given two lines are considered not unique if the average distance is too small.
The average distance is calculated using functions representing both lines, using iterations throughout the full length of the lines.

2.4.3. By average distance calculated with an integral:

Given two lines are considered not unique if the average distance is too small. Unlike the method above, the average is calculated by the integral value between the two lines, in other words, the area, normalized by the line's length.

2.4.4. By slope while considering the average distance between the lines:

Given two lines are considered not unique if their slopes are similar and the average distance between them is small enough.
Because we expect to find a quadrilateral, we can assume that the similar slopes indicate that the average distance is significant.

- We note that in some cases 4 lines are not found in the first chunk of lines. In these cases, we load more chunks of lines, in accordance with their value in the Hough-Space.

3. Some technical details

An important goal of ours was to have a convenient way to compare all the different methods. This was essential for understanding the performance of each method. To achieve this, we created a special way of work, in which all combinations of the different methods are calculated for each run of the program. The heavy calculations such as computing the Hough-Space are divided into different threads, each per a method of computation.

In addition, it was important for us to save these calculations in an organized way.

We created a nested folder for each combination of methods, as can be seen in Fig. 3.1. This is done dynamically in run time and doesn't require any preparation of folders. It is worth mentioning that in addition, we saved results of heavy calculations, e.g., the Hough-Space, done as a part of the whole process. This makes it possible to "start" the program from the line detection stage and not wait every time for the Hough-Space calculation.

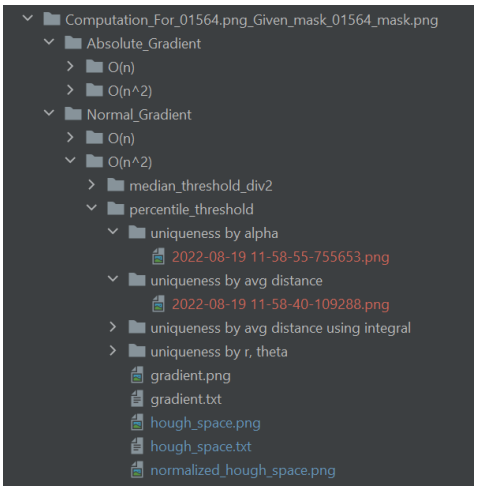
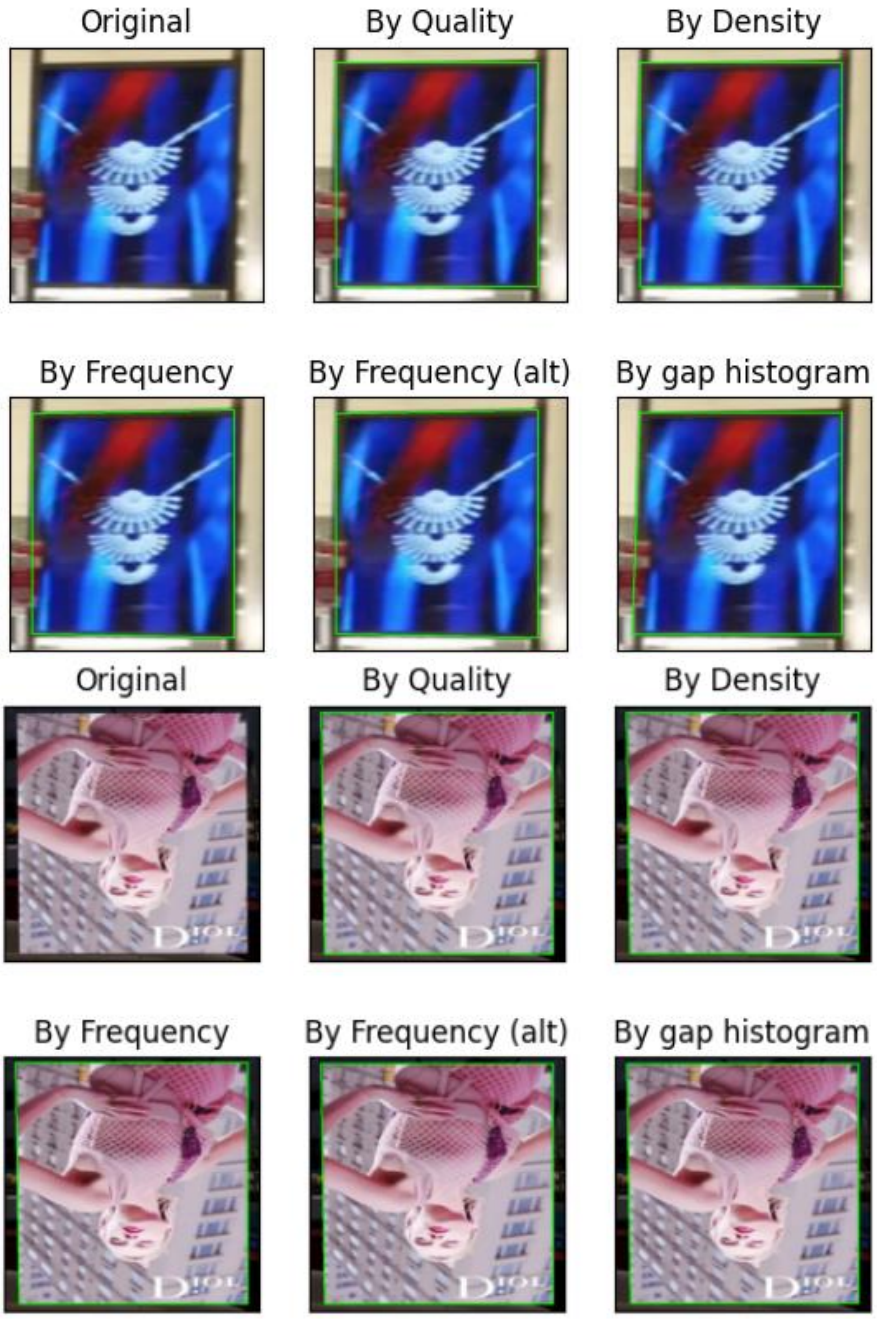


Figure 3.1: nested folders for saving the results

Results

Images with a given mask:



Original

By Quality

By Density

By Frequency

By Frequency (alt)

By gap histogram

Original

By Quality

By Density

By Frequency

By Frequency (alt)

By gap histogram

Images without a mask:

Original

By Quality

By Density

By Frequency

By Frequency (alt)

By gap histogram

Original

By Quality

By Density

By Frequency

By Frequency (alt)

By gap histogram

Conclusions

While testing the many methods we got to the next conclusions:

1. Gradient: Using absolute value doesn't improve results. This is because the Laplacian returns a result in which a high gradient is always next to a corresponding negative value, this happens since a positive gradient represents a transition from bright to dark and the negative gradient represents the opposite.
   In addition, it is worth mentioning that although the final results of the program are quite similar with or without the absolute value, when using the absolute value, we process the double the points which results the double run time is two times higher.
2. Hough-Space: The second method mentioned for computing the Hough-Space is more accurate but much less efficient. It's worth mentioning that in some cases, the first method which is far more efficient returns results that are similar to the second method.
3. Scoring method: The best method is the last one mentioned in the scoring methods section. In this method, we punish the lines score more significantly for high gaps. Therefore, because good lines have low gaps, this seems to be the best method.
4. Line uniqueness: The best method is the last one mentioned in the line uniqueness method section. In this method, we compare the lines using there cartesian representation, considering both the slope and the average distance between the lines. This metric is much better than other possibilities mentioned

References

Hough-Space: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html

Laplacian: https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html