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 Fig 1).
   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

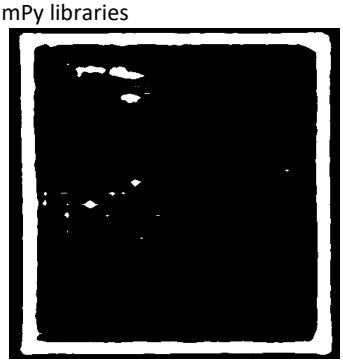

Figure 1.2: Given Mask

2. **Detailed Explanation**

   2.1. Computing gradient:

   The image's gradient 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: Fig 1.1's gradient

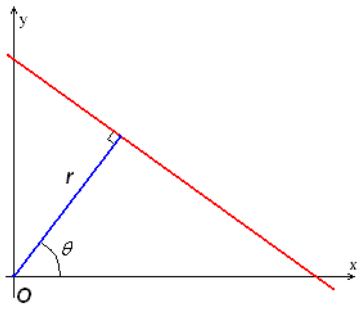

Figure 3: r, Θ represent the line in the Hough-Space

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, 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 high, 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 it 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 we're considered:

2.2.1. Every line that goes through a point:

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

as a result the more valued gradians in a line, the higher the number of votes the line receives.

Time Complexity: O(n)

2.2.2. Every line for every 2 points of gradient (This was suggested by Jihad):

The second method was taking every 2 high enough gradient points and increasing the vote for the line which passes through them.

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

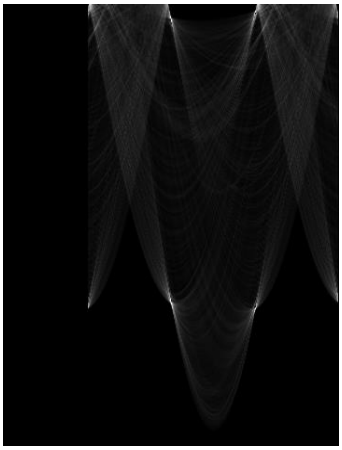

Figure 3.1: Hough-Space by O(n)

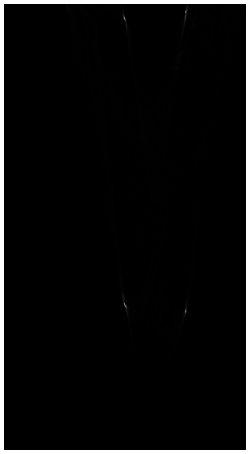

Figure 2.2: Hough-Space by O(n²)
(Normalized in order to emphasize the difference of votes)

2.3. Choosing candidates:

Noise heavy or blurry 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 many different ways:

2.3.1. By gradient quality:

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

2.3.2. By density:

A number of low enough gradient values along the line, normalized to its length. When 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 to its length.

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

Still requires implementation

2.4. Choosing the lines out of the candidates:

A Quadrilateral's edge might be considered as similar but different lines, resulted from the Hough-Space. It is needed to make sure that the top scored, 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 ways:

2.4.1. By polar representation:

Given two lines, we consider them **not** unique if when in polar representation, their $\theta's$ are almost the same(rounded to 3 digits after the zero, in radians) and their radiuses are close enough.

2.4.2. By average distance:

Given two lines, we consider them **not** unique if the average distance is too low. 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, we consider them **not** unique if the average distance is too low. In differ from the above method, the average is calculated by the integral value between the two lines, in other words, the Area, normalized to the line's length

2.4.4. By Incline while considering the radius of the polar representation:

Given two lines, we consider them not unique if their incline is similar and their radiuses are close enough.

Because we expect to find a quadrilateral, we can have the assumption that the incline in suitability to the radius, differs significantly.

Results: