**Research Front in Computer Science – Lab Report**

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

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 which receives an image (jpeg/png) as an input and processes it in order 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 which indicates where the quadrilateral’s edges are (as appears 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

1. **Detailed Explanation**
   1. Computing gradient:  
      The image’s gradient details the rate of the change of color withing the picture, in order to produce it we firstly 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

* 1. Computing Hough-Space:  
     The Hough-Space gives information about lines withing the image space, each dot in the Hough-Space represents a line within the image space, and each point receives votes as 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, so points in the Hough-Space should receive high value 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 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:
     1. Every line that goes through a point:  
        First method was taking every high enough gradient point and increase votes for every point in the Hough-Space which represents a line that passes through said gradient.   
        Lines which go through many gradients will receive many votes  
        Time Complexity: O(n)
     2. Every line for every 2 points of gradient (This was suggested by Jihad):  
        Second method was taking every 2 high enough gradient points and increase the vote for the line which passes through them.   
        This method will drastically increase the number of votes for good candidates.  
        Time complexity: O(n²) – the increase in runtime is really felt even after optimization

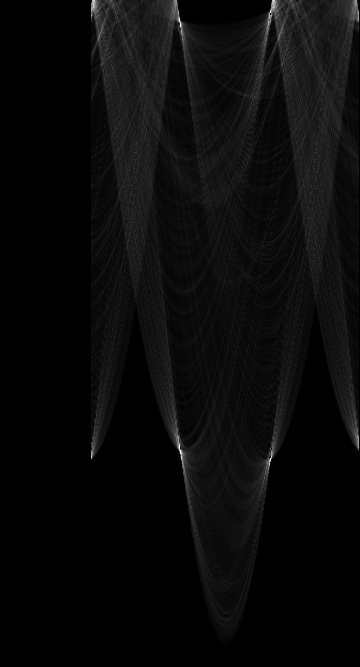
 

Figure 3.1: Hough-Space by O(n) Figure 3.2: Hough-Space by O(n²)   
 (Normalized in order to emphasize the difference of votes)

* 1. 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:
     1. By gradient quality:  
        Sum of high enough gradient values along the line, normalized to its length.
     2. By density:  
        Number of high enough gradient values along the line, normalized to its length.
     3. By frequency:  
        As a line which has smaller gaps between high enough gradient value points is probably 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.
     4. By histogram of gaps (this method was suggested by Jihad at the end):  
        Still requires implimantation