Detecting Lines by Hough Transform

Report for Lab Exercise 4 for the Computer Vision Course

A.P. van Ree (10547320)
T.M. Meijers (10647023)
University of Amsterdam
The Netherlands

December 8, 2014

1 Introduction

Written exercises: This report contains the theory on the Hough transform and it's applications that are discussed in Lab exercise 4: *Detecting Lines by Hough Transform.*

Programming exercises: The code is commented well enough so that every step is clear, and the structure of the code is well organized. The main file (*main.m*) is organized into sections which correspond to the structure in the assignment. The helper functions we have written (*houghLines.m, pointsOfLine.m, lineThroughPoints.m* and *hough.m*) are commented well enough to explain their working as well.

2 Theory on Hough Transform

Finding Lines using the Hough Transform

In this section, the assignment was the implementation of the function *hough.m*. This function uses the *canny.m* funtion we have written in a previous lab, to find all the edges. For every pixel that is seen as an egde, this function will calculate a range of combinations of θs and a ρs. These combinations describe all the possible lines that go though this pixel, where θ ($0 \leq \theta < \pi$) is the orientation of the line and ρ (- diameter of image$\leq \rho \leq$ diameter of image) is the distance from the origin of the image. To calculate ρ in the Matlab code the following formula is used:

$$\rho = x * \sin(\theta) - y * \cos(\theta)$$

. The formula for this in the Lecture Notes is the following:

$$r = z_1 * \cos(\theta) + z_2 * \sin(\theta)$$

The difference between these two forms is purely Matlab coördinates. One of the reasons for the transition is because Matlab uses coördinates as (y, x) and not (x, y), as it is in math. The other is that Matlab does not have its origin of axis where the two axes collide (left bottom of an image), but at the point where the y-axis begins (top left of an image). This is why the addition has turned into a subtraction.
Because of this flip of origin, ρ could be negative. When a *Hough-point* is e.g. somewhere in the left corner of the picture, x is small and y is big. Putting these in the equation will lead to a negative ρ.
The lines formed by certain values of θ and ρ occur more than others, for different pixels. These are the lines we are looking for, and the Hough-transform will show maxima in the number of occurrences of these combinations.

Using the Lines

When using the Hough-lines to do certain transformations, e.g. projective transformation, we can to use the cross product of the homogeneous coördinates of the lines acquired by the previous sections. With this cross product the intersection of two lines is calculated. The cross product of two 2D vectors, a x b, is equal to $||a||\ ||b|| \sin(\theta)$, where θ is the angle between the two lines. So this is dependent on the sine of the angle of the two lines. But seeing as we have two 3D vectors, the equation is different. The weight of the cross product is the last of the three rows, and is calculated by the two rows above. And as said before, this is dependent on the sine of the angle between the two lines. Thereby the weight is dependent on the sine of the angle between the two lines.
This works the same way for two points in homogeneous representation. If the cross product is taken of these two, the result is the line connecting them. The last row of this vector (the weight), is dependent on the sine of the angle that the points make if they make a right-angled triangle.

3 Conclusion

All of the assignments have been finished, except for the second method to be used in section six. Everything works as wanted. There was a problem with section five, but after a long session of debugging, it turned out to be a mix up of x and y (Matlab coórdinates). The rest went pretty smoothly.