Shape detection, Hough transform

1st Mikel-ange Barros
computer science department
University Claude Bernard Lyon 1
Lyon, France
mikel-ange.barros@etu.univ-lyon1.fr

2nd Zacharia Beddalia
computer science department
University Claude Bernard Lyon 1
Lyon, France
zacharia.beddalia@etu.univ-lyon1.fr

3rd Thomas Scouflaire
computer science department
University Claude Bernard Lyon 1
Lyon, France
thomas.scouflaire@etu.univ-lyon1.fr

***Abstract*—Image analysis is a field based on a set of tools made for object recognition in images. One of the few areas of expertise of image analysis is shape detection.In this paper we will present our approach to well-known detection methods, the Hough transforms. Then we will compare it to the method implemented by one of the most important image analysis libraries: OpenCV.**

***Index Terms*—image analysis, Hough Transform.**

I. Introduction

Image analysis is a branch of computer science with a growing number of different use, will it be in the field of video games, medecine or data protection. In all these fields the detection of recognizable patterns is an important part of the analytical work and allows the development of a large number of techniques. It is with this in mind that our work is carried out by implementing the hough transform for circles and lines.

Hough Transforms are a set of accumulator-based pattern detection methods. In theory, any shape can be detected, as long as the curve defining it is known and that the correct accumulator is defined. In practice, however, this form of detection is often limited to two objects: lines and circles. And this is what we are going to present to you today.

A. Principle of the Hough transform

Whether for segments or circles, the principle is the same and can be broken down into three steps.

- an accumulator is defined with the dimensions necessary for detection.
- For each point of the image corresponding to a border, all the shapes (circle or line) passing through this point are calculated and the accumulator is filled with the number of times a line passes through a point.
- The points are filtered by keeping only the local maxima that exceed a certain value.

We can refine the result with a fourth step that will sort each value and keep the n maximum values found.

These steps will be described in more detail in the definitions of the different algoithms.

II. Our algorithm

A. Data structures

In order to achieve the hough transformation we have defined a data structure consisting of two elements:

- a dynamic array of vectors containing information on the detected shapes
- an accumulator adapted to the parameter of the curve describing the shape you want to detect

B. Preprocessing

In order to be able to do shape detection, we must first pass our image through an outline For this we will use our own algorithm which is detailed in the paper [MZT20].

C. Hough transform for lines

As explained above, the hough transform on lines is done by counting for each line the number of pixels being contours of the image are passing through that line.

In our algorithm the accumulator used is a 1D array of size nbRho * nbTheta. Indeed, to detect the lines, we work with polar coordinates. The equation of a line with cartesian coordinates is $ax + bx + c$. The fact that it is described with three parameters is not ideal. However, its equivalent in polar coordinates is $x.\cos(\theta) + y.\sin(\theta) = \rho$ where we only have two parameters, which will result in a reduction of the algorithm complexity.

Once this has been defined the algorithm used is rather simple and can be broken down into 4 steps.

- The accumulator is reset to 0.
- All lines passing through each pixels of the image belonging to a contour are calculated in polar coordinates.
- These calculated polar coordinates correspond to a cell of our accumulator and since several line can pass through the same pixel, for each of these lines we increment the value corresponding in the accumulator.
- For all the values of the accumulator the local maxima are taken which are higher than a threshold defined by the user.
- A straight line is defined as the ρ and θ corresponding to each of the selected values.

Thanks to this algorithm we define a set of lines that represent the most expressed lines on our image. However we would like to be able to limit them to the shapes on our image. For this the simplest method is to start from the centre of our line and run our line in both directions as long as we are on a border pixel.And as soon as we are no longer on a border pixel we define that we are no longer on a representative line and we limit our line to the ends thus found. (fig 2)

Sample results:

(a) Result on simple shapes (b) Result on complex shapes

Fig. 1. Comparison of the results on different images

(a) Result on simple shapes

Fig. 2. with contour limitation

It can be noticed that if the results are interesting on simple shapes, they are not very representative of what the image is for complex shapes . (fig 2) And this is why we have implemented a second algorithm.

D. Alternative Hough transform algorithm for lines

The second algorithm is quite similar to the first, but the changes made are significant enough to be detailed.

- First of all for this algorithm the accumulator is a 2D matrix.
- Then, as in the first algorithm, we calculate for a contour pixels all the straight lines passing through this pixel and we accumulate the values as in the first algorithm.
- Instead of looking for local maxima we look to see if the line with the highest accumulated value passing through this point is greater than a threshold value.
- If this is not the case we move on to the next pixel.
- If this is the case, we move along the line in both directions until we encounter a pixel without a border, an edge or that we have exceeded a certain distance fixed by the user.
- Finally, it is checked whether the length of the detected segment is greater than a chosen threshold and, if so, it is kept.
- And then we move on to the next pixel.

This algorithm is very similar to the previous one, the difference being that the line detection is done for all points, not only for local maxima. Moreover, the length tracking being done during the analysis and not in post processing makes the algorithm much more efficient on complex images. (fig 3)

(a) Result on simple shapes (b) Result on complex shapes

Fig. 3. Comparison of the results on different images

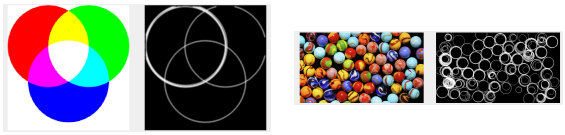
E. Hough transform on circles

The circle detection algorithm is very similar to the first line detection algorithm except for two details:

- First of all for this algorithm the accumulator is a 3D matrix.
- Here we accumulate the values in relation to the center of the circle (a polar coordinates parameter)and the radius of the circle, no longer in relation to the couple (ρ, θ) defining a straight line.

In order to lighten the calculation, a minimum and a maximum radius are passed manually by the user.

This algortihm gives fairly good results. (fig 4)



(a) Result on simple shapes (b) Result on complex shapes

Fig. 4. Comparison of the results on different images

F. threshold influence

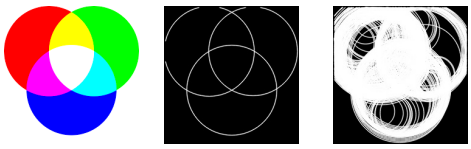
The definition of the threshold is a very important part of these different algorithms. Indeed, it makes it possible to eliminate certain parasitic lines (fig 5,7). However, it can also have a big impact on the result and can erase an important element. (fig 6)

(a)base image (b)threshold=150 (c)threshold=50

Fig. 5. threshold on the first line algorithm

(a)base image (b)threshold=50 (c)threshold=10

Fig. 6. threshold on the second line algorithm



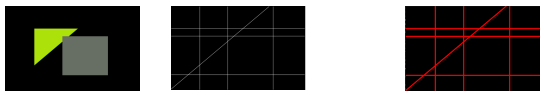
(a)base image (b)threshold=100 (c)threshold=50

Fig. 7. threshold on the circle algorithm

G. Comparative results

Now that we have been able to test the efficiency of our algorithms, let's compare their efficiency with OpenCV's algorithms. The OpenCV test implementation is based on the article [OCVH] and [OCVHC]

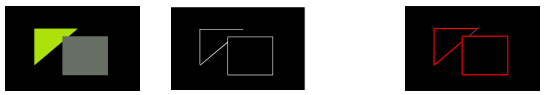
As we can see, the results obtained are very similar to those of OpenCV. It can be assumed that the difference in results may be due to the edge detection method used in the different tests. Indeed, where we use our own edge detection algorithms, opencv uses its implementation of the canny method.



(a) base image (b) our algorithm (c) OpenCV Implementation

(d) base image (e) our algorithm (f) OpenCV Implementation

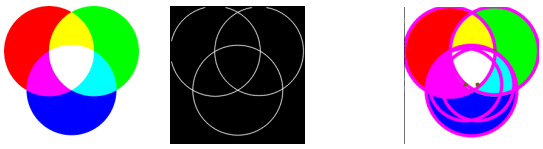
Fig. 8. first line algorithm



(a) base image (b) our algorithm (c) OpenCV Implementation

(d) base image (e) our algorithm (f) OpenCV Implementation

Fig. 9. second line algorithm



(a) base image (b) our algorithm (c) OpenCV Implementation

(d) base image (e) our algorithm (f) OpenCV Implementation

Fig. 10. circle algorithm

H. Limit of our algorithm

The algorithm put in place has two major problems:

- First of all, it is far too dependent on user settings which makes it difficult to use in a general case and often requires a lot of testing and tweaking to find the right settings. In order to correct certain problems we could set up an automatic threshold method.
- We can only extract simple shapes, which limits the usefulness of our algorithms.

III. Conclusion

We can notice that each parameters influences the detection of contours and that it is difficult to find parameters to obtain a perfectly smooth and usable result. However, hough transform algorithms are powerful and efficient algorithms that allow simple shapes to be detected in a short time. Once generalised to a large number of shapes, these algorithms can be very useful in many areas.

Acknowledgement

Thanks to Mrs. Saida for this project and for her help.

Reference

[MZT20] Edge detections methods, Barros Mikel-ange, Beddalia Zacharia, Scouflaire Thomas, Octobre 2020.
[OCVHL] Hough line tutorial
[WPHTC] Wikipedia Hough Circle Transform Page
[WPHTL] Wikipedia Hough Line Transform Page
[OCVHC] Hough circle tutorial