|  |
| --- |
| Circle Detection Using the Hough Transform |

Input

The objective is to develop a program to find circular objects in an image using Hough transform techniques. Three test images on which to run the program were used

Preprocessing

The first stage in object detection is preprocessing. The source image itself must be “cleaned up” so that the resultant input image to the object detection program is in its best possible state (which is relative to the detection requirements). In the three given input images, the goal is to make the circles as distinct as possible from any other elements in the picture, avoiding noise, background objects, and occlusion. The most straightforward way of achieving this is to blur the image with an average filter to eliminate noise and then use a Canny edge detector to identify the edges of the circles.

To achieve the best possible looking images, a variety of methods were attempted. These include Gaussian smoothing followed by Canny detection; average smoothing, Gaussian smoothing, followed by Canny detection; average smoothing, Canny detection, followed by Gaussian smoothing, etc. The various combinations produced interesting results. Gaussian smoothing before edge detection produced less accurate circle detection than did average smoothing. Gaussian smoothing after Canny edge detection resulted in detection of 7/8 outer circles in circles1.bmp. Histogram equalization was also attempted before and/or after edge detection (in attempts to enhance contrast of circles with the background), but the results were poor. Surprisingly, the most effective technique was the simplest: average smoothing followed by Canny edge detection.

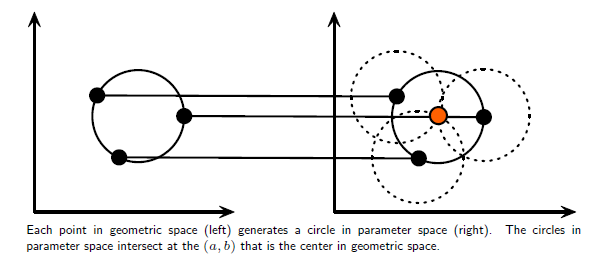
Circular Hough Transform

After the preprocessing stage comes the bulk of the work, the object detection stage itself. As the objects of interest are circles, the ideal method for detection is the Circular Hough Transform (CHT). The CHT works as follows:

The equation of a circle consists of three parts: the x-component of the center, the y-component of the center, and the radius. A circle can be described by the parametric equations:

x = a + Rcosθ  
y = b + Rsinθ

θ is sweeped through from 0 to 360 degrees to complete the full circle of radius R. These equations are used to trace circles of various values of R around each edge point of potential circles in an image. These circles are combined together in an accumulator, each contributing to the brightness of the true center of the pixel. The brightest pixel for different values of R corresponds to centers of different circles.



Algorithm

This is the section responsible for creating the accumulator (value of radius is given as an incrementer in a loop in the main function)

for(int i = 0;i < acc.rows;i++)

{

for(int j = 0;j < acc.cols;j++)

{

if (dst.at<uchar>(i,j) > 256/2)

{

for (int theta = 0; theta <= 360; theta++)

{

Xc = i + (radius \* cos(theta\*PI/180));

Yc = j + (radius \* sin(theta\*PI/180));

if(Xc < acc.rows && Yc < acc.cols && Xc > 0 && Yc > 0)

{

acc.at<uchar>(Xc,Yc) = (int)acc.at<uchar>(Xc,Yc) + 1;

}

}

}

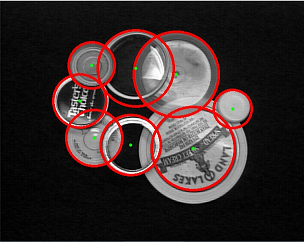
}

}

The algorithm is straightforward. It traverses each pixel. If the pixel is at least faintly white/gray, then it is most likely an edge, and is processed. The processing part has been explained above; each edge point creates a circle around itself that contributes to the true center of the target circle. The value of the true center is then incremented in the acc (accumulator) array.

The full program used is provided in the attached .cpp file. The above example excludes validation (addressed in Challenges section). The following images show the detected circles overlayed on circles1.bmp, circles2.bmp, and circles3.bmp.

Output



Challenges

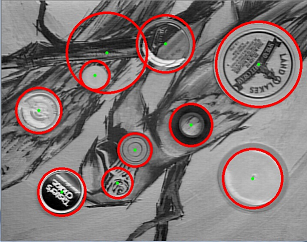
Several obstacles present themselves when using the Circular Hough Transform.

# Input Image

The input image itself poses several issues. Any occlusion or edge-rich background is bound to interfere with proper circle detection. The solution to this issue is to attempt many preprocessing steps before running the CHT algorithm on the image. As specified above, many techniques were tried, including Histogram Equalization, Gaussian blur, average blur, Canny edge detection, and several combinations of the above. The most effective tool, however, proved to be the minimum threshold parameter of the Canny edge detector. This allowed for maximum background edge reduction.

# Spurious Circles

The images given were plagued with incorrect circle detection. Large incorrect circles (see below image) were detected as a result of accumulation of a high number of votes relative to other points in the accumulator. This is solved by refining the process for choosing the best possible candidate center point for each radius value. Instead of merely checking if a certain candidate point is the largest, the sum of differences with said point and its neighbors are calculated. The center point with the largest sum of differences is the true center, as this corresponds to a sharp peak in intensity (in other words, local maximum), which is what is expected.

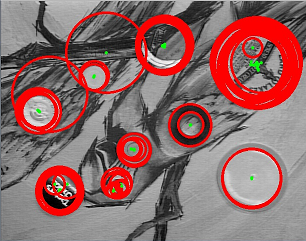


Small spurious circles are caused by a similar issue. Because small circles have small radii that are close to the outline of the circle itself, their centers become densely populated, causing several small circles to be formed (See below image). This can be solved by identifying a minimum radius threshold, which serves to prevent small circles from accumulating in the first place. The acceptability of this solution depends on the expected range of circle radii and the detection requirements at hand.



# Embedded Circles

The calculation of the sum of differences limited all accumulations to the true centers of the circles. However, this gave rise to yet another problem; several circles with radii close in value share the same center, and so repeatedly overwrite and underwrite the border of the circle, until a large portion of the circle itself is covered in detected circles. This issue is solved by storing center points and their brightness value in an array. The distance between a potential candidate center point and each previously stored center point is calculated. If the candidate center point is too close to any of the centers of the previously detected circles, it does not print a circle.



# Computational Complexity

Given all the problems above, each requires a costly solution. Add to that the cost of using them all together, and the result is a ridiculously slow program. This program is no different; due to the computational complexity of optimizations used, in addition to the increased complexity of parameters used in said optimizations, the delay is quite high. Unfortunately, reduction of complexity has not been very feasible; at least n2 operations are required to visit every pixel in the image. This is compounded with the cost of an accumulator created for each radius as well as validation operations.

# Other Attempts

In addition to the attempted filtering techniques, unsharp masking was used to try and sharpen the detected edges. However, this attempt did not seem to improve the edge detected image much. Further attempts have been made to refine the results from the accumulator during the post-processing stage. One such attempt was to store the center point for each accumulator in an array. After all the accumulators had been compiled, the array was sorted, and the 12 highest ranking (based on number of votes) center points were to be drawn on the source image. Theoretically, it should have worked, as the true center points are the brightest pixels in each of the accumulators. Practically, however, none of these came close, so the attempt was discarded. The added advantage of this particular method would have been to allow circle detection within other circles, and would eliminate the need for having a minimum distance between centers policy.