Project Report

Hough Transform

CSE473/573: Computer Vision and Image Processing

(Fall 2016)

Instructor: Chang Wen Chen

Teaching Assistants: Radhakrishna Dasari, Shuang Ma

Sugosh Nagavara Ravindra

sugoshna@buffalo.edu

Person#: 50207357

Vaibhav Sinha

vsinha@buffalo.edu

Person#: 50208769

Submission Date: 12/16/2016

Table of Contents

Literature Survey

Hough transform is one of the robust algorithms for circle detection. To implement this we used Python programming language and OpenCV library. OpenCV is a library of programming functions mainly aimed at real-time computer vision. OpenCV provided us various functionalities including blurring of images (Gaussian Blur), edge detection (Canny) and much more not limited to the project.

Gaussian Blur is mainly used to blur the image and remove sharp noise in the images. This will be the input to Canny edge detector. Canny edge detector is robust algorithm to detect edges in an image. It has two values of threshold assigned to it, on optimization we found that 75 and 150 are the ideal thresholds for both.

Introduction

The Hough transform is a feature extraction technique used in digital image processing. The purpose of this technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out using accumulation arrays. Hough transform can be implemented to detect many arbitrary shapes in an image. In this project we implement Hough transform to detect circles in an image.

We were able to detect circles in the image but it also noise sensitive. To generalize this to all images we provide the user with a parameter to use. This parameter is mainly used to increase the sensitiveness of circle detection.

Implementation

We used Python and OpenCV library to implement the project. For both accumulator array implementation and the randomization implementation we have provided the threshold to tune in the beginning of the program. The user can tune them based on the noisiness, sensitiveness needed for particular image. The overview of the implementation is we traverse the image multiple times for different values of radius of circles and accumulate the hits at every pixel. The basic idea is that, the position where there is most number of hits, that point must be the centroid of the circle of that particular radius.

The algorithm used to implement Hough Circles is as follows:

Step 1: Read the image

Step 2: Gaussian Blur the image to reduce noise

Step 3: Edge detection using Canny Edge detector

Step 4: For R from 10 to MAX(rows,cols)/2

For every pixel f(x,y)==255

For theta in range(0,360) with a step size of 10

x = x0 – R*cos (theta)

y = y0 – R*sine(theta)

Accumulator[R][x][y]+=1

Step 5: To normalize values perform, Accumulator=Accumulator/MAX(Accumulator)

Step 6: Construct circles for every R,x,y if the threshold is above a certain value.

OpenCV functions used:

- imread (): imread() is used to read an image from the filesystem.
- imshow(): imshow() is used to output a particular image to the user.
- resize(): resize() is used to resize the image to a particular shape.
- GaussianBlur(): GaussianBlur() is used to blur the image to a specified kernel size.
- Canny(): Canny() is used to detect edges in the image. It takes on two threshold parameters, which on tuning we found out that 75 and 150 are the ideal thresholds.
- circle(): circle() is used to draw circles onto an image at a particular pixel x,y with a radius r.

Once the Accumulator array was obtained we had to normalize it to obtain a standard threshold value between 0 to 1. This threshold specifies the occlusion ration and in turn the sensitiveness of circle detection.

In order to not cloud too many circles at one particular pixel value we used a sliding window technique to find the most accumulated pixel in a 30x30 (30 was found to yield the best result) window and plot circle only at that position. Downside to this technique was we could not detect concentric circles. We fixed this by normalizing it again and checking if it was in the top 10%.

We found that this technique is very noise sensitive as it tries to find circles at every pixel. Hence we provide the user with the sensitiveness parameter to tune with for any image.

Software and Program development

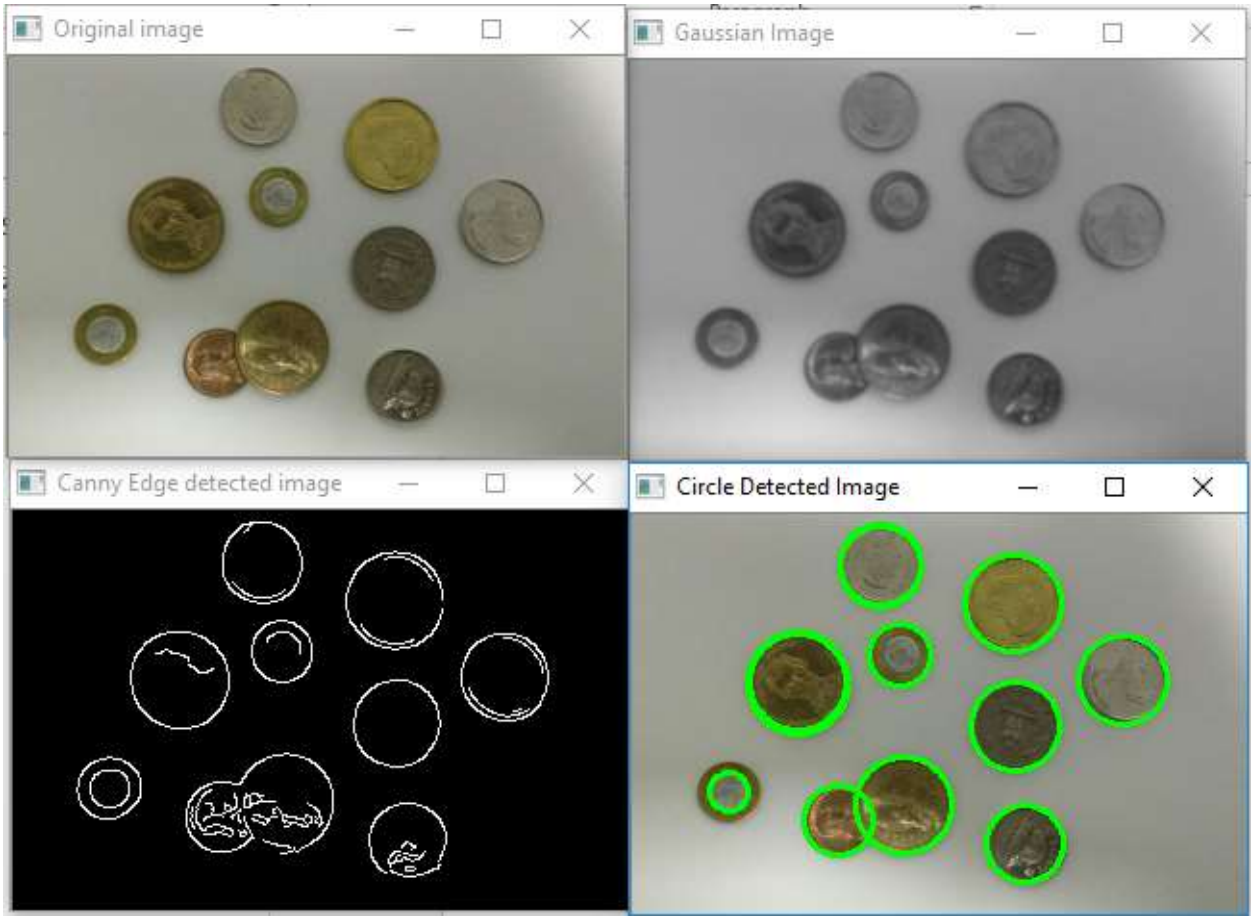
The project is divided into two parts, implementation of Hough transform using accumulator array and implementation of Hough transform by randomization. We found it ideal if one person implemented accumulator array implementation and the other implemented randomization implementation and that's how we divided the work.

We developed the code only using the references listed and the algorithm to implement using accumulator array was referred from the text book.

We learnt valuable lessons while implementing the algorithm which are but not limited to:

- Dimensionality constraints
- Noisy image handling
- Threshold tuning
- Space and time constraints
- Teamwork

Accumulator Array Results



As seen from the above image the coins are detected in the image and marked in green. We also see that the occluded coin is also detected.

There are few limitations to this approach such as:

- The computation of the accumulator array can be time consuming. Hence we tried to minimize this be increasing the theta step size from 1 to 10.
- This method is noise sensitive as it tries to detect circles at every foreground pixel. One of the solutions to this is to increase the kernel size of the GaussianBlur() but this is not an optimal solution.

We learnt a great deal of information while implementing Hough Transform. Even though we just implemented Hough transform to detect circles, it can be implemented to detect various features if there is a parametric equation to it such as, lines, curves, ellipses etc. There is only little modification needed to be made in this algorithm to detect other shapes is to change the circle equation to the shape of the object we need to detect.

Summary

In conclusion we can say that Hough transform is a robust and an efficient program which can be implemented on various image processing and computer vision tasks which might include any feature detections. We found pretty good accuracy on test images but we did face few limitations when it came to noisy images. To avoid this we have provided the user with parameters to tune to obtain the best possible efficiency to their task.

CSE573 was a valuable course and we learnt a great deal of information in past couple of months. Learning about various algorithms every week and practically implementing them in homework sets and projects were the best part. So we can conclude by saying the course was theoretically and practically very well constructed and this project helped us gain much more insight into the image processing world.

Bonus Presentation and Discussion

Overview

The algorithm works by first detecting edges using Canny Edge Detector and generating the binary image containing only the edges.

From the edge image, 4 points are selected at random. Then, using mathematical derivations, it is checked that if those 4 points belong to the same circle.

In case they belong to the same circle, it is further verified that is there a true circle lying there by counting the number of points lying on that circle in the edge image.

If 4 points are selected at random (x1, y1), (x2, y2), (x3, y3), (x4, y4)

We first make sure they are not collinear by checking the equation

$$(x2 - x1)(y3 - y1) - (x3 - x1)(y2 - y1) = 0$$

Then, first a circle passing through 3 points is found out using the formula

$$a_{123} = \frac{\begin{vmatrix} x_2^2 + y_2^2 - (x_1^2 + y_1^2) & 2(y_2 - y_1) \\ x_3^2 + y_3^2 - (x_1^2 + y_1^2) & 2(y_3 - y_1) \end{vmatrix}}{4((x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1))}$$

and

$$b_{123} = \frac{\begin{vmatrix} 2(x_2 - x_1) & x_2^2 + y_2^2 - (x_1^2 + y_1^2) \\ 2(x_3 - x_1) & x_3^2 + y_3^2 - (x_1^2 + y_1^2) \end{vmatrix}}{4((x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1))}.$$

After obtaining the center (a_{123}, b_{123}), the radius can be calculated by

$$r_{123} = \sqrt{(x_i - a_{123})^2 + (y_i - b_{123})^2}$$

for any $i = 1, 2, 3$.

After obtaining the center, it is then verified that does the 4th point lie on this circle

We verify that if that point satisfies the circle equation

$$d_{4\rightarrow 123} = (x_4 - a_{123})^2 + (y_4 - b_{123})^2 - r_{123}$$

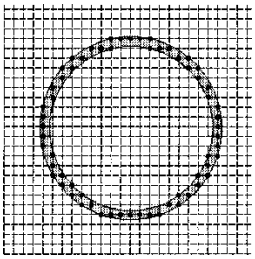


Figure 1 A digital circle

If v4 lies on the circle C123, the value of d4→123 in above Eq. is 0. Since the image is digital, it rarely happens that these edge pixels lie exactly on a circle. Therefore, the goal of circle detection is to detect a set of edge pixels which lie not exactly but roughly on a digital circle (see Fig. 1). For convenience, the set of edge pixels that form a digital circle is also called a circle and these edge pixels are called co-circular.

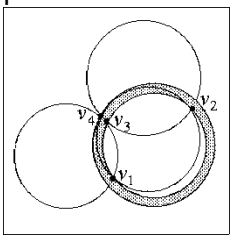


Figure 2 An example of four pixels in a digital circle

As shown in Fig. 2, v4 lies on the boundary of the circle C_{123}; then the value of $d_{4\rightarrow 123}$ in is very small. Therefore, above equation can be used to determine whether v4 lies on the circle C_{123} or not.

We use a threshold to check if the 4th point lies on the circle or not.

When two of the three agent pixels of the possible circle are too close, the possible circle may not be the true circle. As shown in Fig. 4, v1, v2, and v3 lie on a true circle (the bigger circle), but the circle (the smaller circle) determined by v1, v2, and v3 differs from the true circle. The undesirable case occurs when v2 and v3 are too close. To avoid this case, the distance between any two agent pixels must be greater than a given threshold Ta . If so, it means that the three agent pixels have strong evidence to be the representatives of the possible circle.

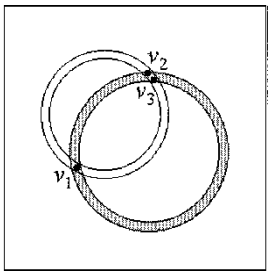


Figure 3 An undesirable case

After detecting a possible circle with center ($a_{i\,j\,k}$, $b_{i\,j\,k}$) and radius $r_{i\,j\,k}$, whether the possible circle is a true circle can be checked by the following evidence-collecting process

We count the points on the edges which lie on this circle. If points are greater than

$2\prod r$ * Tr where Tr is a threshold, then we declare it as a true circle.

Algorithm

From the above description, this section presents the formal RCD consisting of the following six steps.

Step 1. Store all edge pixels $v_i = (x_i , y_i)$ to the set V and initialize the failure counter f tobe0.LetT_f,T_{min},T_a,T_d,and T_r be the five given thresholds. Here, T_f denotes the number of failures that we can tolerate. If there are less than T_{min} pixels in V , we stop the task of circle detection. The distance between any two agent pixels of the possible circle should be larger than T_a. T_d and T_r are the distance threshold and ratio threshold, respectively. Moreover, let $|V|$ denote the number of edge pixels retained in V.

Step 2. If $f = T_f$ or $|V| < T_{min}$, then stop; otherwise, we randomly pick four pixels v_i,i =1,2,3,4,outof$V$.Whenv_i has been chosen, set $V = V - \{v_i\}$.

Step 3. From the four edge pixels, find out the possible circle such that the distance between any two of the three agent pixels is larger than T_a and the distance between the fourth pixel and the boundary of the possible circle is larger than T_d ; go to Step 4. Otherwise, put v_i, i =1,2,3,4,backto$V$;perform $f := f +1$;gotoStep2.

Step 4. Assume $C_{i\,j\,k}$ is the possible circle. Set the counter C to be 0. For each v_l in V , we check whether $d_{l \rightarrow i\,j\,k}$ is not larger than the given distance threshold T_d . If yes, $C := C + 1$ and take v_k out of V . After examining all the edge pixels in V , assume $C = n_p$, i.e., there are n_p edge pixels satisfying $d_{l \rightarrow ijk} \leq T_d$.

Step 5. If $n_p \geq 2\pi r_{i\,j\,k} T_r$, go to Step 6. Otherwise, regard the possible circle as a false circle, return these n_p edge pixels into V , perform $f := f + 1$, and go to Step 2.

Step 6. The possible circle $C_{i\,j\,k}$ has been detected as a true circle. Set f to be 0 and go to Step 2.

Implementation

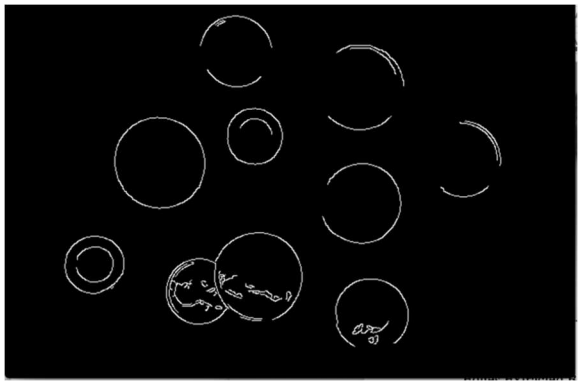
1. Image was read using opencv library in python.
2. After that, Gaussian smoothing was applied using cv2.GaussianBlur
3. Then, edge detection was carried out using Canny Edge detector with appropriate threshold - edges = cv2.Canny(img, 85,200)
4. Edge pixels extracted in a list.
5. 4 pixels extracted at random and checked if there is a true circle,
6. If there is a true circle, they it is marked on the image and points on the circle in the edge image extracted out.
7. If it is not a true circle, then the 4 points are added back to the edges set
8. If number of points in the edge set less that Tmin or number of iterations greater than Tf or, Stop and Display the image
9. Follow Steps 5 to 8 until completion

Results

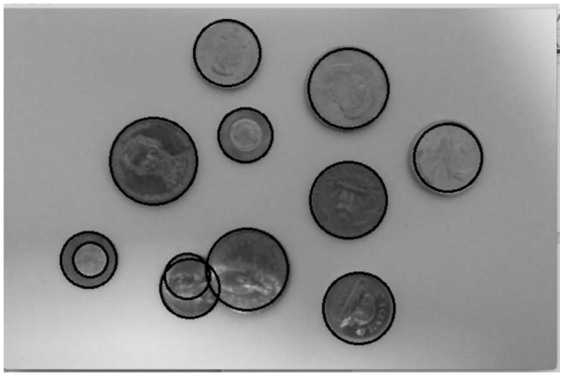
Original Image

Gaussian Image

Edges Extracted Binary Image



Circles Tagged



Conclusions

1. Final Result depends a lot on the kind of noise in the image.
2. A noisier image will have more false positives than a lesser noisy image.
3. The results in the image above are quite accurate, though there is an extra circle found out mainly because of noise in the edge which is forming an approximate semi-circle and rest of the circle is fulfilled by the real circle in that portion of the image. Hence, it is detected as well.
4. The extra circle detected can be eliminated by stricter thresholds but doing that makes the whole process computationally expensive, hence, a less strict thresholds are used.
5. Only library functions of OpenCV are used to eliminate noise and detect edges.
6. The thresholds found out are tuned for the image provided as input. There is no guarantee that these will work for all images as every image is different in terms of content, noise and edges.

Lessons learned from the bonus project

1. Seeing the results of a randomization algorithm in practice is quite encouraging.
2. The above procedure can be extended to identify other similar shapes like, ellipse, parabola, etc. Though, the number of parameters will increase.
3. The results depend a lot on the noise in the image and it is quite challenging to come up with a general threshold which can work for all images.

References

1.Yuen, H. K., et al. "Comparative study of Hough transform methods for circle finding." *Image and vision computing* 8.1 (1990): 71-77.

2. Chen, Teh-Chuan, and Kuo-Liang Chung. "An efficient randomized algorithm for detecting circles." *Computer Vision and Image Understanding* 83.2 (2001): 172-191.