Project name: Historic Aerials Index Project

Date: Oct 8, 2021-Dec 26, 2021

Project Co-leaders: Jaime Martindale, Jim Lacy

Project participants: Gengkui Liu, Rachel Ren

Context and Challenge

Historical maps play an important role in Wisconsin's history, as they record the landscape change through years. However, these valuable physical maps are aging and becoming more expensive to maintain. We need to find an effective method to preserve these map indexes, and make them easy to access at little to no cost. The challenge is, digitizing features of interest in the historical map index has so far been a manual task. Manual extracting spatial features is a repetitive and time-intensive task when dealing with a collection of tens of thousands of maps. As a result, we try to use computer vision and machine learning techniques to modernize this process, and let map users easily access the information they need.

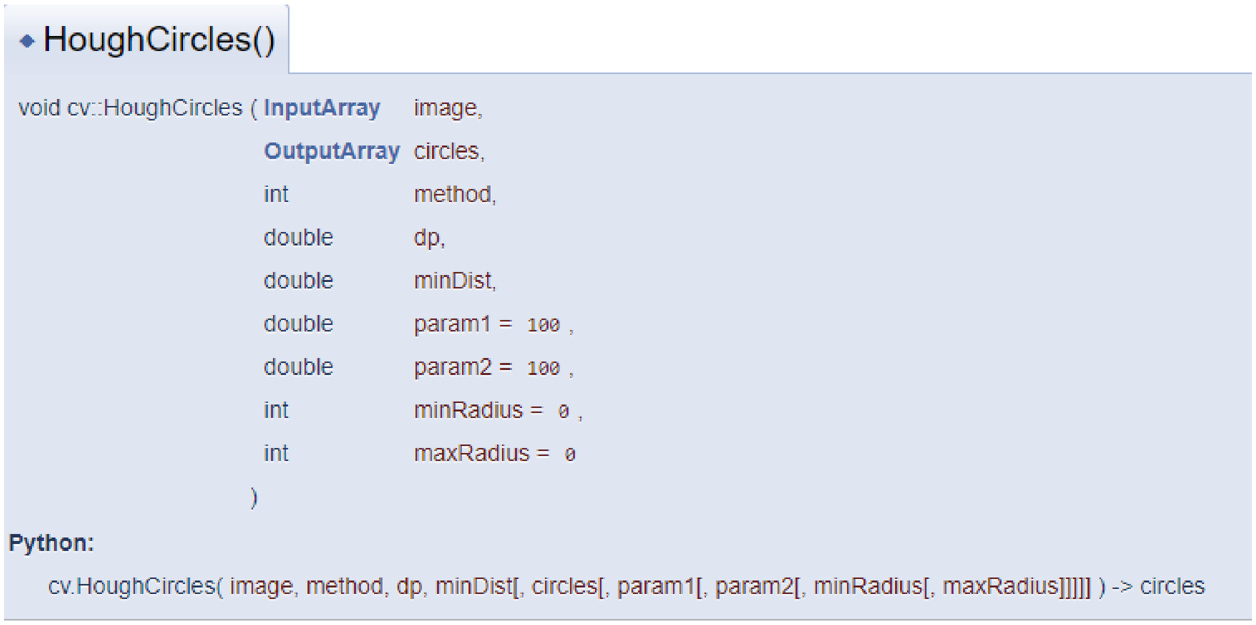
Methods

We were inspired to use the OpenCV library for this project because we found this "map vectorizer" project led by the New York Public Library lab[1]. This map vectorizer project is to modernize the points, lines, and polygons from the historic paper maps, which is very similar to what we need to do here. We got a rough idea of where to start after we read the paper[2] for this map vectorizer project, so we started to learn some computer vision, especially object detection knowledge in the OpenCV documents.

[1] https://github.com/nypl-spacetime/map-vectorizer
[2] http://imap.mauriciogiraldo.com/doc/map-poly-paper.pdf

The OpenCV is an open source Python library that solves computer vision related problems. We choose to use the OpenCV library because this library can detect objects in the image. As our project aims to identify and extract dots from the georeferenced aerial photos, we can use HoughCircles() function in the OpenCV library to detect the circles around the dots as well as the centers of all circles. If all dots were successfully circled out, the center of the circle would be the center of the dot, and the spatial coordinates of each dot would be generated by the HoughCircle() function.

HoughCircle() function finds circles in a grayscale image using hough transformation. There are eight parameters in this function, and the parameter names and data types are specified in the screenshot below[3]:



◆ HoughCircles()

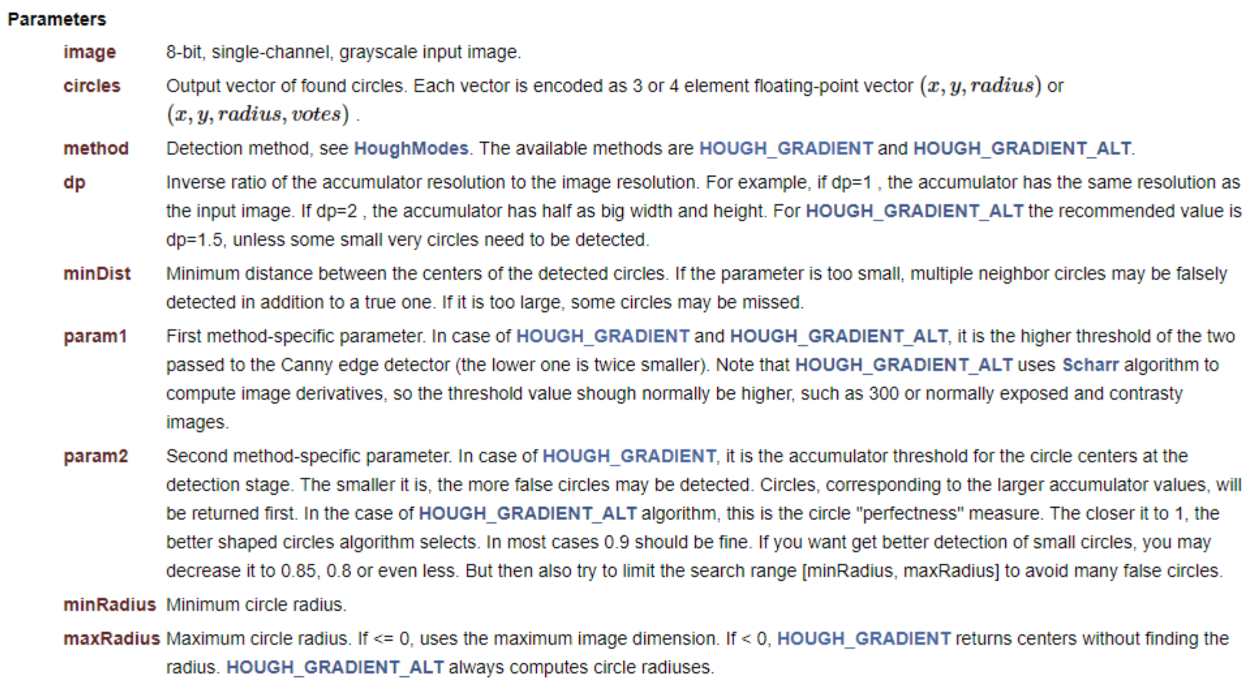
```
void cv::HoughCircles ( InputArray   image,
                        OutputArray  circles,
                        int          method,
                        double       dp,
                        double       minDist,
                        double       param1 =  100 ,
                        double       param2 =  100 ,
                        int          minRadius =  0 ,
                        int          maxRadius =  0 
                      )
```

Python:

```
cv.HoughCircles( image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]]] ) -> circles
```

Each parameter is explained in the screenshot below:

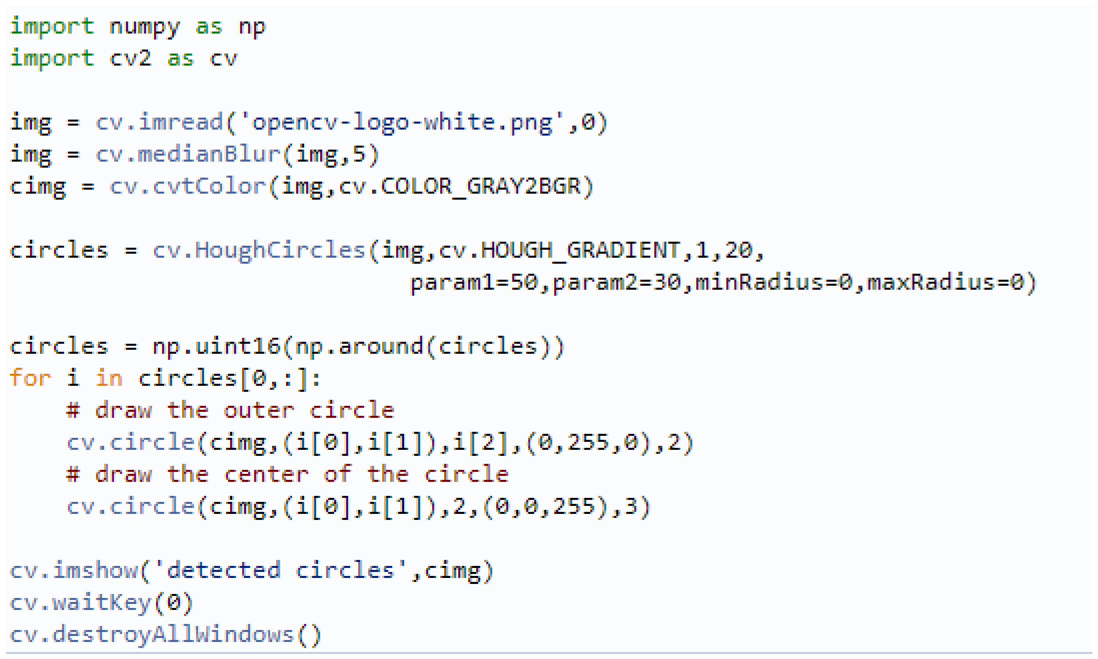
[3] https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d



Parameters

image	8-bit, single-channel, grayscale input image.
circles	Output vector of found circles. Each vector is encoded as 3 or 4 element floating-point vector $(x, y, radius)$ or $(x, y, radius, votes)$.
method	Detection method, see HoughModes. The available methods are HOUGH_GRADIENT and HOUGH_GRADIENT_ALT.
dp	Inverse ratio of the accumulator resolution to the image resolution. For example, if dp=1 , the accumulator has the same resolution as the input image. If dp=2 , the accumulator has half as big width and height. For HOUGH_GRADIENT_ALT the recommended value is dp=1.5, unless some small very circles need to be detected.
minDist	Minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.
param1	First method-specific parameter. In case of HOUGH_GRADIENT and HOUGH_GRADIENT_ALT, it is the higher threshold of the two passed to the Canny edge detector (the lower one is twice smaller). Note that HOUGH_GRADIENT_ALT uses Scharr algorithm to compute image derivatives, so the threshold value shough normally be higher, such as 300 or normally exposed and contrasty images.
param2	Second method-specific parameter. In case of HOUGH_GRADIENT, it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first. In the case of HOUGH_GRADIENT_ALT algorithm, this is the circle "perfectness" measure. The closer it to 1, the better shaped circles algorithm selects. In most cases 0.9 should be fine. If you want get better detection of small circles, you may decrease it to 0.85, 0.8 or even less. But then also try to limit the search range [minRadius, maxRadius] to avoid many false circles.
minRadius	Minimum circle radius.
maxRadius	Maximum circle radius. If <= 0, uses the maximum image dimension. If < 0, HOUGH_GRADIENT returns centers without finding the radius. HOUGH_GRADIENT_ALT always computes circle radiuses.

In the OpenCV library tutorials document, we find the original code that can be used to detect dots in the photo index[4]:



```
import numpy as np
import cv2 as cv

img = cv.imread('opencv-logo-white.png',0)
img = cv.medianBlur(img,5)
cimg = cv.cvtColor(img,cv.COLOR_GRAY2BGR)

circles = cv.HoughCircles(img,cv.HOUGH_GRADIENT,1,20,
                            param1=50,param2=30,minRadius=0,maxRadius=0)

circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv.circle(cimg,(i[0],i[1]),2,(0,0,255),3)

cv.imshow('detected circles',cimg)
cv.waitKey(0)
cv.destroyAllWindows()
```

[4] https://docs.opencv.org/4.x/da/d53/tutorial_py_houghcircles.html

What we need to do based on this original code, is to read in our own image, turn it into a grayscale image, and tune the parameters such that the HoughCircles() function can correctly detect as many dots as possible. For example, we have a photo index in the year of 2003, and we tailored the code for this photo index to successfully detect the majority of dots (please find the detection result in the "results" section):

```
import numpy as np
import cv2 as cv
#read in the original image
img = cv.imread('2003_01.jpg',0)
#use median blur method to remove the noise
img = cv.medianBlur(img,5)
cimg_target = cv.cvtColor(img,cv.COLOR_GRAY2BGR)
circles = cv.HoughCircles(img,cv.HOUGH_GRADIENT,1,60,
                            param1=50,param2=35,minRadius=15,maxRadius=45)
if circles is not None:
    print(circles,len(circles[0]))
    circles = np.uint16(np.around(circles))
    for i in circles[0,:]:
        # draw the outer circle
        cv.circle(cimg_target,(i[0],i[1]),i[2],(0,255,0),2)
        # draw the center of the circle
        cv.circle(cimg_target,(i[0],i[1]),2,(0,0,255),3)
    #rescale the generated image to fit in the computer screen
    scale_percent = 30
    width_cimg_target = int(cimg_target.shape[1] * scale_percent / 100)
    height_cimg_target = int(cimg_target.shape[0] * scale_percent / 100)
    dim_cimg_target = (width_cimg_target, height_cimg_target)

    resized_cimg = cv.resize(cimg_target, dim_cimg_target)
    cv.imshow('detected circles',resized_cimg)
else:
    print(circles)
cv.waitKey(0)
cv.destroyAllWindows()
```

Results

Each historic map is different in some way, which increases the difficulty of detection and the uncertainty of the result. Before we perform dot detection analysis on the historic map, we firstly draw some circles in Photoshop, and then test how successful the HoughCircles() function can detect these circles.

There are three types of circles:

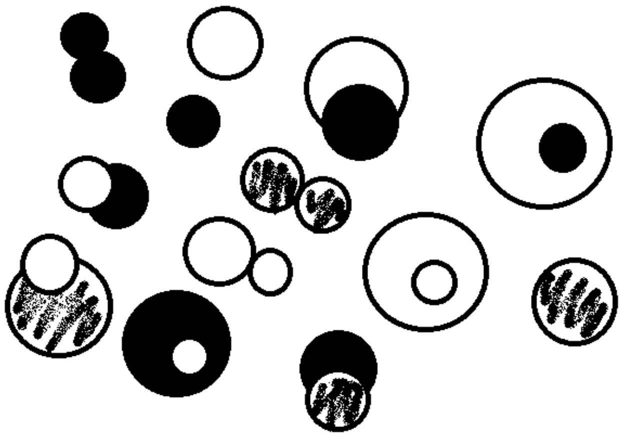
1. solid circle 2. hollow circle 3. shaded circle.

There are three types of spatial relationships between circles:

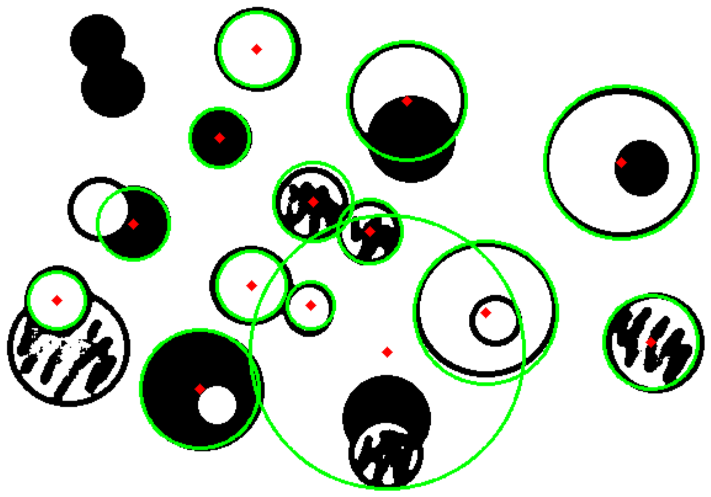
1. One circle touches the other circle

2. One circle partially overlaps with the other circle

3. One circle is completely within the other circle



HoughCircles() function detects circles:

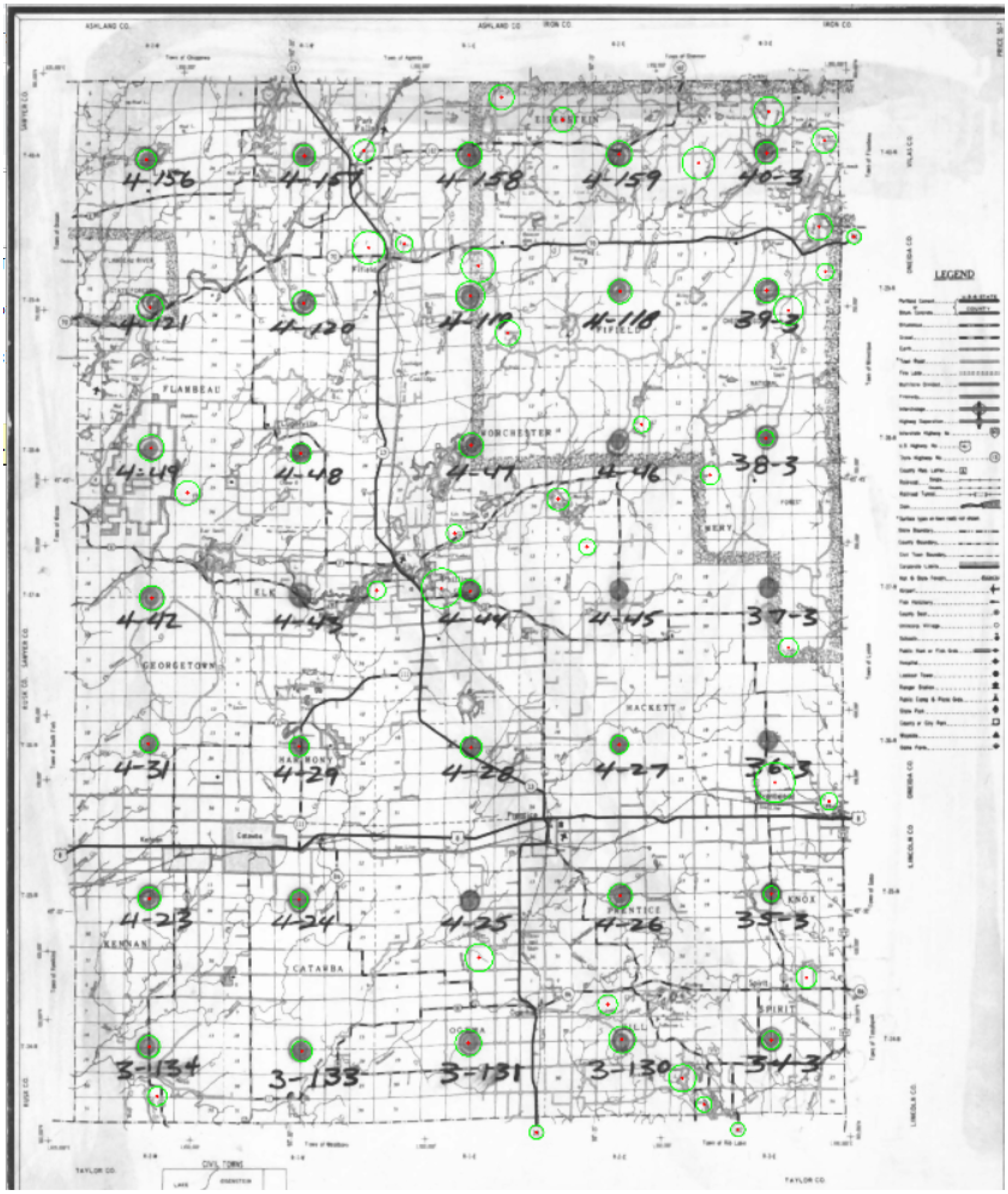


The parameters we use to detect the dots above are shown here:

```
circles = cv.HoughCircles(img,cv.HOUGH_GRADIENT,1,50,
                          param1=50,param2=37,minRadius=0,maxRadius=0)
```

HoughCircles() catches 13/23 circles in green and identifies their centers in red. We may notice that this function fails to detect circles that are completely within other circles. In addition, it seems to falsely detect a circle that does not exist. However, dots in the aerial photo index represent the locations where the camera on the plane took photos along a straight line, so rarely two dots would completely overlap with each other.

We proceed to detect dots on the real photo index. This photo index is in the year of 2003 and it contains clear dots. The detected dots are circled out in green color:



The parameters we use to detect the dots in the above map are shown below:

```
circles = cv.HoughCircles(img,cv.HOUGH_GRADIENT,1,60,
                              param1=50,param2=35,minRadius=15,maxRadius=45)
```

When the dots are very clear and there are no overlaps or no touches, the HoughCircles() function does a great job. 29/35 dots are successfully detected, but some circles that should not exist also get generated by the HoughCircles() function.

To reduce the numbers of the falsely generated circles, we can try:

- Cutting the image into multiple smaller size parts would increase the accuracy of detecting the circles.
- Find the vertical and horizontal linear coordinates of these points. If the points are not in the vertical and horizontal lines, then they are errors.

Next Step

- Find a method to generate the x and y coordinates for all the correctly detected circles. This is because we do not need circles, instead, we need the centers of the circles and their corresponding coordinates.
- Use a proper machine learning model to automatically detect dots in the other map index. Some papers suggest using the Conventional Neural Network (CNN) algorithm, but feel free to experiment with other models. Please refer to the **Air Photo Index Research** folder in the shared Google drive.

Helpful Resources

These are the StackOverflow links that help us understand the concepts, fix the code, etc. They may be helpful for the future process as well so we list them out.

- https://stackoverflow.com/questions/34706092/how-exactly-does-dp-parameter-in-cvhoughcircles-work

- https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga564869aa33e58769b4469101aac458f9
- https://stackoverflow.com/questions/33544411/in-the-circle-hough-transform-what-is-the-inverse-ratio-of-accumulator-resoluti/33545904#33545904
- https://stackoverflow.com/questions/54377418/why-does-one-circle-go-undetected-by-houghcircles-how-does-mindist-work
- https://stackoverflow.com/questions/43852023/detecting-small-circles-using-houghcircles-opencv
- https://stackoverflow.com/questions/59105602/what-do-the-parameters-in-hough-circle-signify-and-how-to-know-what-values-to-us