CSE585/EE555: Digital Image Processing II

Computer Project # 1:

Mathematical Morphology: Hit-or-Miss Transformation

Yifei Xiao, Ling Zhang, Jiaming Chai

Date: 01/31/2020

A. Objectives

In this project we will learn the following topics:

1. How to use basic MATLAB functions to process images.
2. How to clean image noises using the basics learned in the class.
3. How to implement Hit-or-Miss transform.

B. Methods

Since the image given is not a pure binary image, the first step is to convert it to black-and-white image. By using *rgb2gray* function, we can convert the original image into a gray image. Then for each pixel, we set a threshold equals 128. By comparing each pixel to threshold, we can convert the image to pure binary image. But still it is not ready for hit-or-miss transform yet. In order to reach the final goal, we take the following steps as shown in Figure 1. We think using a flow chart serves the purpose of our algorithm. All the steps are discussed in the following sections.

To run the project, just run *main.m* file. Although it can give the final result, it has too many figures that MATLAB may sometimes add things to wrong figure. We recommend running section by section.

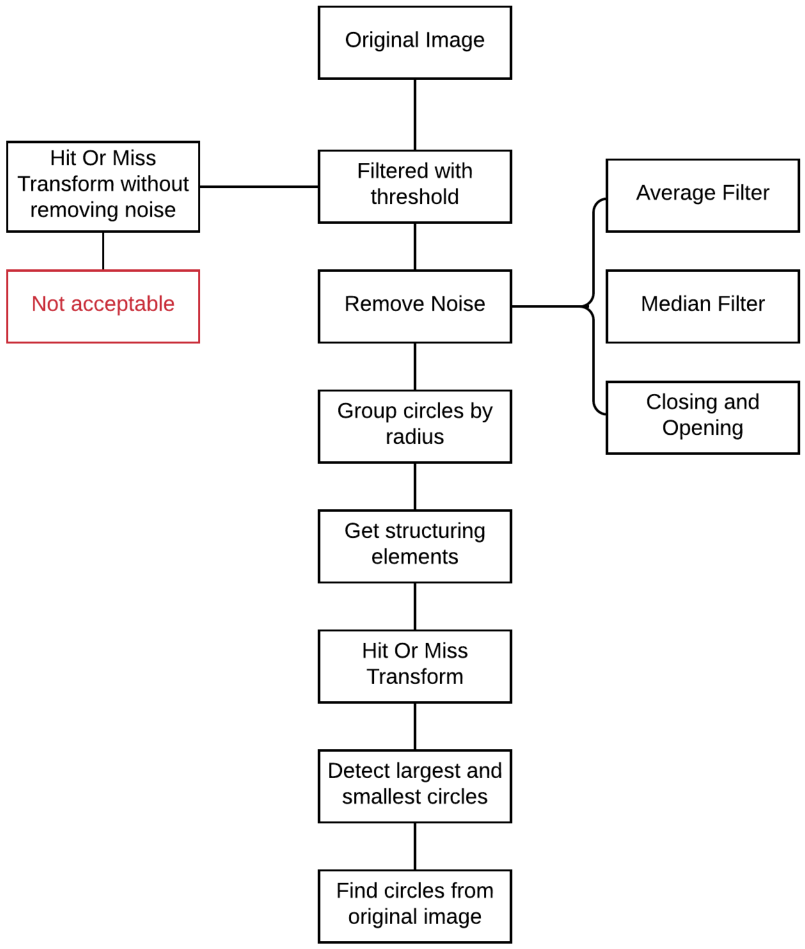


Figure 1: Flowchart of entire project

1. Remove noise: *removeNoise.m* and *closeAndOpen.m*

The binary image contains 10% level of salt-and-pepper noise. To remove those noise, we use a close/open method and verify the result by using mean and median filter. For close/open method, we create a 3x3 white structuring element and use that structuring element to move through the entire image just like a sliding window. For closing, we implement a dilation followed by an erosion. After closing, we enforce opening, which contains an erosion first and then a dilation. Notice that we implement opening twice to completely remove all noises.

Part of dilation and erosion codes are shown in Figure 2.

```
se = [1 1 1; 1 1 1; 1 1 1]; % white 3x3 structuring element

% Closing:
% Dilation
for i = 2: row - 1
    for j = 2: col - 1
        if (im(i-1, j-1) == se(1, 1) || im(i, j-1) == se(1, 2) || im(i+1, j-1) == se(1, 3) ||...
            im(i-1, j)   == se(2, 1) || im(i, j)   == se(2, 2) || im(i+1, j)   == se(2, 3) ||...
            im(i-1, j+1) == se(3, 1) || im(i, j+1) == se(3, 2) || im(i+1, j+1) == se(3, 3))
            filt_im_1(i, j) = 1;
        else
            filt_im_1(i, j) = 0;
        end
    end
end

%Erosion
for i = 2: row - 1
    for j = 2: col - 1
        if (im(i-1, j-1) == se(1, 1) && im(i, j-1) == se(1, 2) && im(i+1, j-1) == se(1, 3) &&...
            im(i-1, j)   == se(2, 1) && im(i, j)   == se(2, 2) && im(i+1, j)   == se(2, 3) &&...
            im(i-1, j+1) == se(3, 1) && im(i, j+1) == se(3, 2) && im(i+1, j+1) == se(3, 3))
            filt_im_1(i, j) = 1;
        else
            filt_im_1(i, j) = 0;
        end
    end
end
```

Figure 2: Code for Dilation and Erosion

To verify the correctness of our close/open method, we use two other filters, mean and median filters. For the mean filter, we also check the value for each pixel and calculate the mean of that pixel and its eight neighbors. If the average of filter numbers is greater than the threshold, we change the center pixel to white; otherwise, change it to black. For the median filter, the procedure is the same except that we use the median of the nine numbers in the filter to determine the color for the center pixel.

2. Get structuring elements: *strucutringElement.m*

Hit-or-miss transform is defined as

$$X \otimes (A,B) = (X \ominus A^S) \cap (X^c \ominus B^S)$$

where *X* is the input image and *A, B* are structuring elements. So, structuring elements are crucial to implement the hit-or-miss transform. The first step is to assign all circles into 5 different groups. The function *regionprops* can measure properties of image regions. It shows all radii of circles in the image. Then we put them into 5 different groups by *kmeans*.

The next step is to create structuring elements. Notice that in MATLAB it is hard to create a circle-shaped array, so we decide to use square-shaped array instead. In order to find smallest circles, **the side length of structuring element A for hit should be fit just inside the smallest circle, the formula is**

$$L_A \leq 2 * \frac{r_{min}}{\sqrt{2}}$$

and the side length of structuring element B for miss should be larger than the diameter of the smallest circle with padding, the formula is

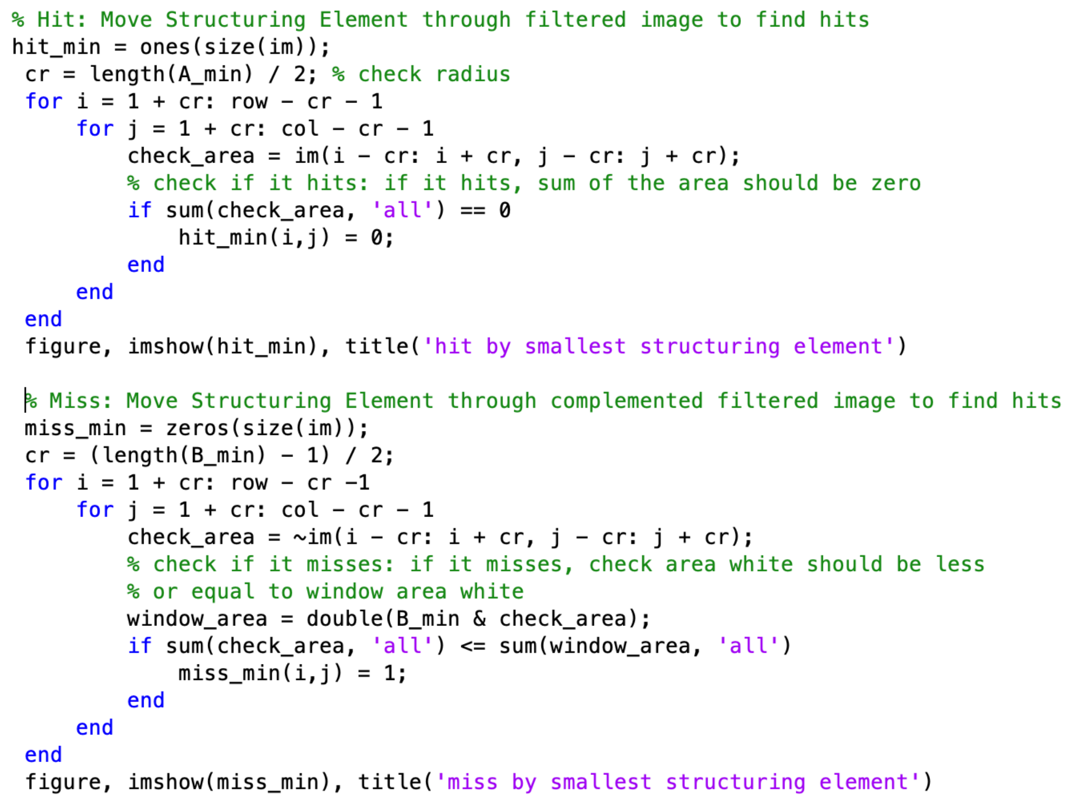
$$L_B > r_{min} + padding$$

Structuring elements for largest circles are in the same way.

3. Hit-or-miss transform: *HitOrMiss.m*

The flow for finding hit and miss is similar to sliding windows: move the structuring elements through the filter image and check the values within the region of the structuring elements. For hit, the sum of the detected area should be zero, which means they are all black. For miss, the sum of the detected area should be less or equal to the window area of the structuring elements.

Part of hit or miss codes are shown in Figure 3.



```
% Hit: Move Structuring Element through filtered image to find hits
hit_min = ones(size(im));
 cr = length(A_min) / 2; % check radius
 for i = 1 + cr: row - cr - 1
     for j = 1 + cr: col - cr - 1
         check_area = im(i - cr: i + cr, j - cr: j + cr);
         % check if it hits: if it hits, sum of the area should be zero
         if sum(check_area, 'all') == 0
             hit_min(i,j) = 0;
         end
     end
 end
 figure, imshow(hit_min), title('hit by smallest structuring element')

 % Miss: Move Structuring Element through complemented filtered image to find hits
 miss_min = zeros(size(im));
 cr = (length(B_min) - 1) / 2;
 for i = 1 + cr: row - cr -1
     for j = 1 + cr: col - cr - 1
         check_area = ~im(i - cr: i + cr, j - cr: j + cr);
         % check if it misses: if it misses, check area white should be less
         % or equal to window area white
         window_area = double(B_min & check_area);
         if sum(check_area, 'all') <= sum(window_area, 'all')
             miss_min(i,j) = 1;
         end
     end
 end
 figure, imshow(miss_min), title('miss by smallest structuring element')
```

Figure 3: Code for hit or miss

4. Highlight detected circles from original image

Now we have done our hit-or-miss transform and find smallest and largest circles. The last step is to highlight them on the original (filtered) image. Result shows that we have found 5 smallest circles and 7 largest circles.

C. Results

In summary, the results using our method meet our expectation and satisfy the requirement per operation. We did work on some difficulties during our meeting and discuss the possible ways of solving them. All the results are shown in the order as in Section B.

1. Remove noise: *removeNoise.m* and *closeAndOpen.m*

Results show that close/open method has the same output as mean and median filters. It is sufficient to claim that our close/open method is good enough to remove salt-

and-pepper noise from the binary image. The original image is shown in Figure 4. It is not completely black-and-white and contains 10% level of salt-and-pepper noise.

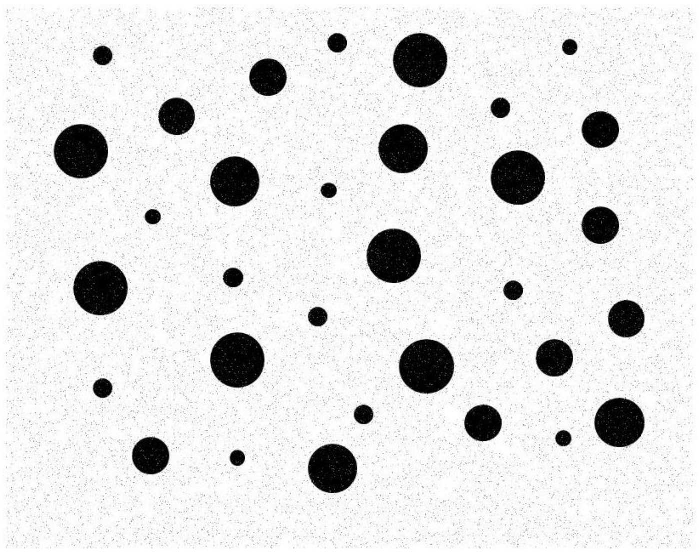


Figure 4: Original Image

After setting threshold, we can convert image into pure black-and-white image (Figure 5), but salt-and-pepper noise still exists.

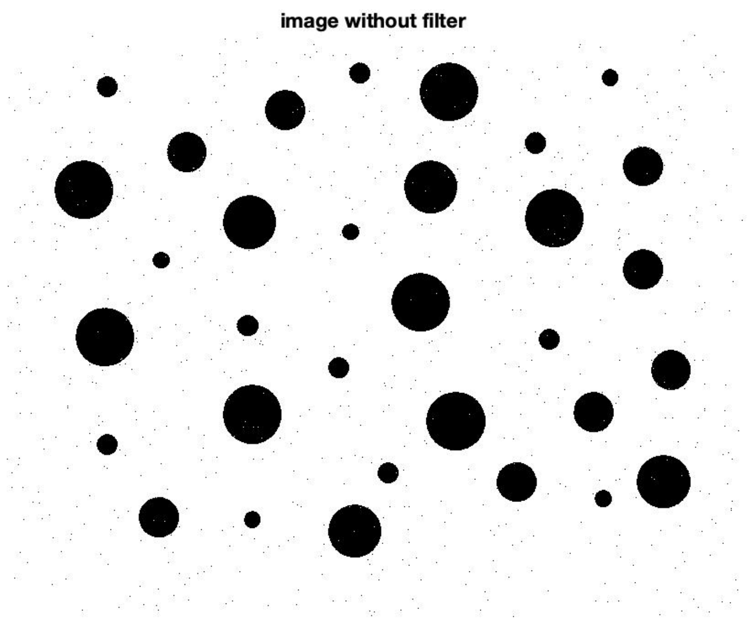


Figure 5: Binary image with noise

If we implement close/open method once, we can see there is no white spot in black circles, but black spots still exist (Figure 6). If we implement opening again, we can remove all noise (Figure 7).

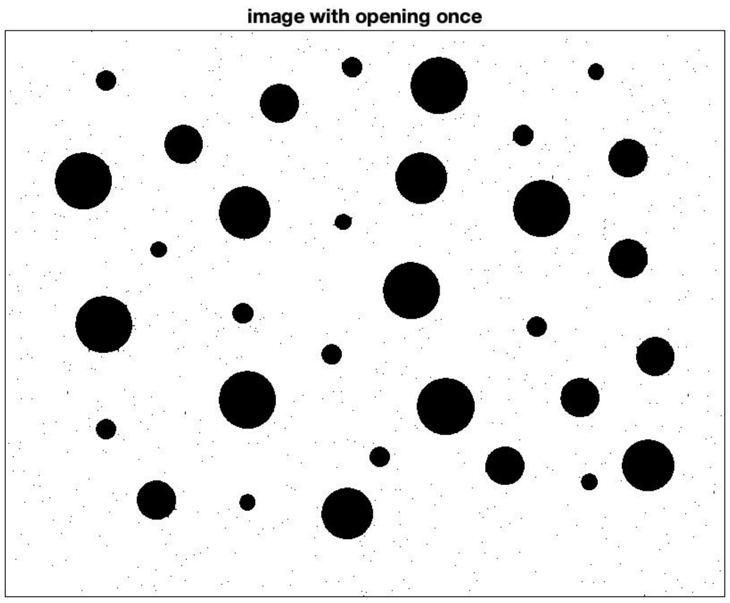


Figure 6: Close/Open once

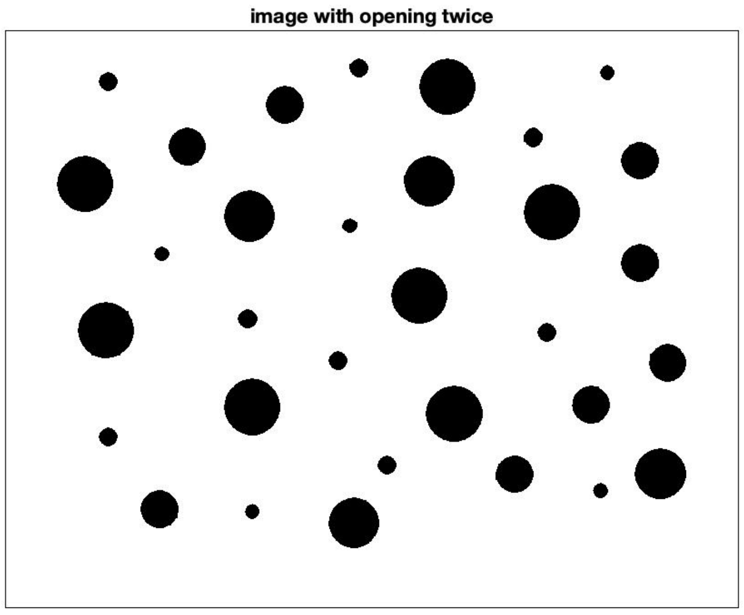


Figure 7: Opening twice

We can compare our result with mean and median filter (Figure 8, Figure 9). The results are the same. So, our close/open method is good enough to remove noise.

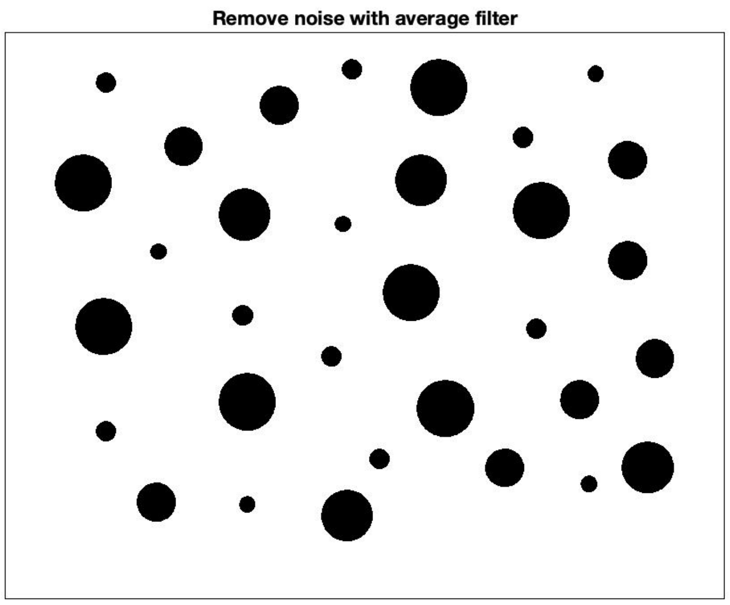


Figure 8: Remove noise by mean filter

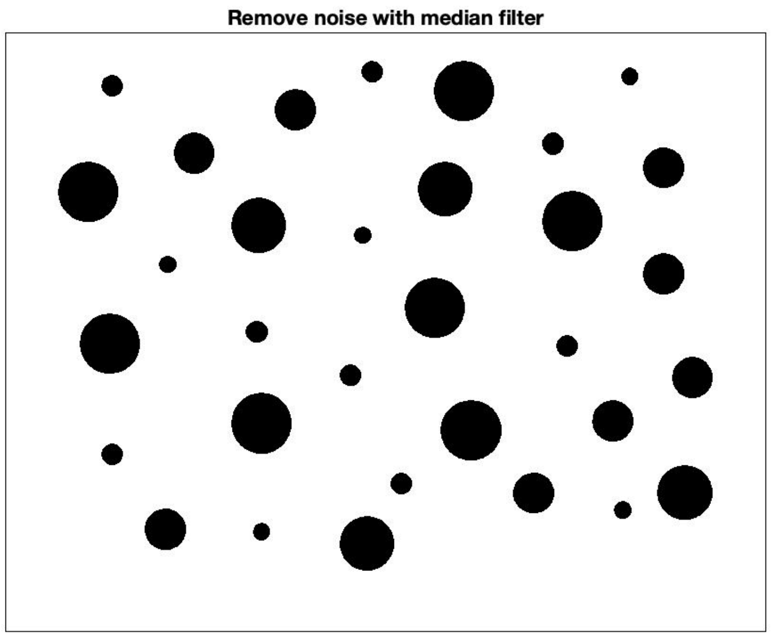


Figure 9: Remove noise by median filter

2. Get structuring elements: *strucutringElement.m*

The radii of five sets are around 8 pixels, 10 pixels, 20 pixels, 27 pixels and 30 pixels. So, for smallest circles, the size of structuring element A for hit is set as 10x10 pixels; the size of structuring element B for miss is set as 23x23 pixels. Similarly, for largest circles, the size of structuring element A for hit is set as 42x42 pixels; the size of structuring element B for miss is set as 69x69 pixels. Note that we add 3x3 padding to structuring element B. All four structuring elements are shown in Figure 10.

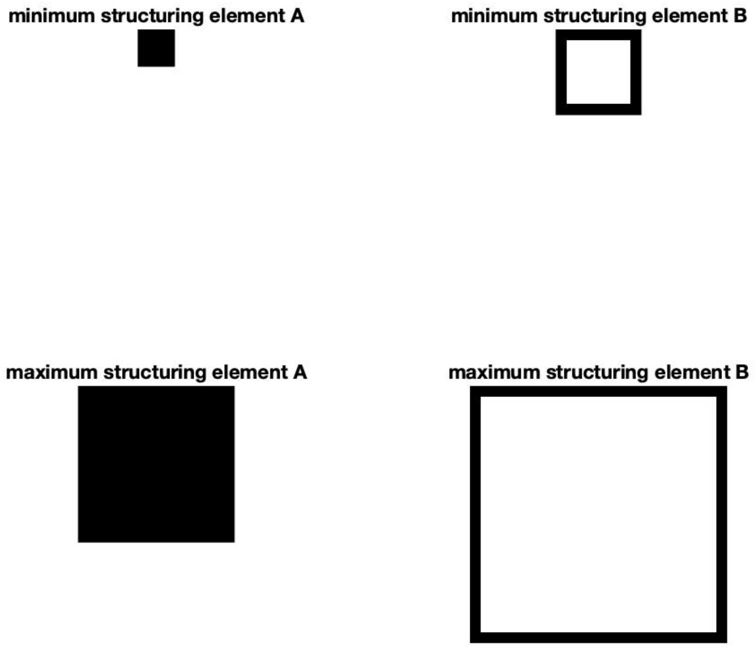

minimum structuring element A
minimum structuring element B
maximum structuring element A
maximum structuring element B

Figure 10: Structuring Elements

3. Hit-or-miss transform: *HitOrMiss.m*

After implementing hit-or-miss transform on the image, we can get results shown in Figure 9 ~ Figure 12. On Figure 9 and 11, black spots are hit circles. Smallest black spots, though very hard to see, are the target circles we need to find. On Figure 11 and 13, white spots are missed circles. Smallest white spots are the target circles we need to find. They should be in same locations where smallest black spots are on Figure 12 and 14.

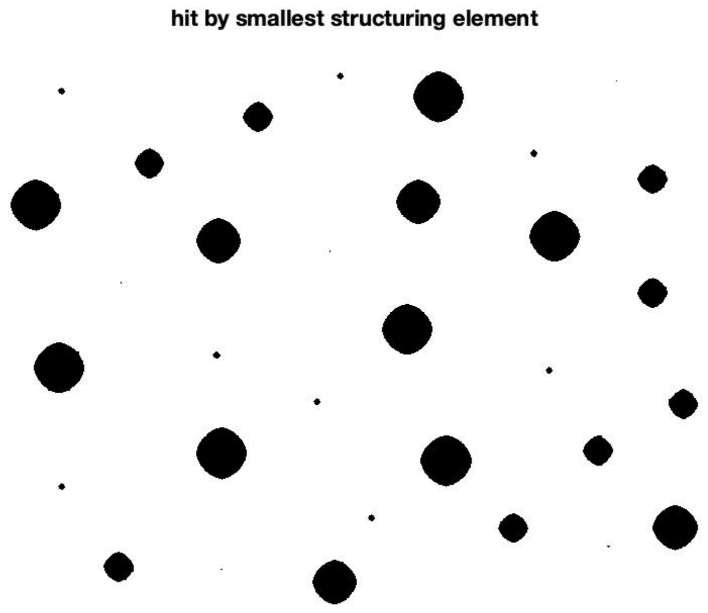


Figure 11: Smallest circles hit by structuring element A

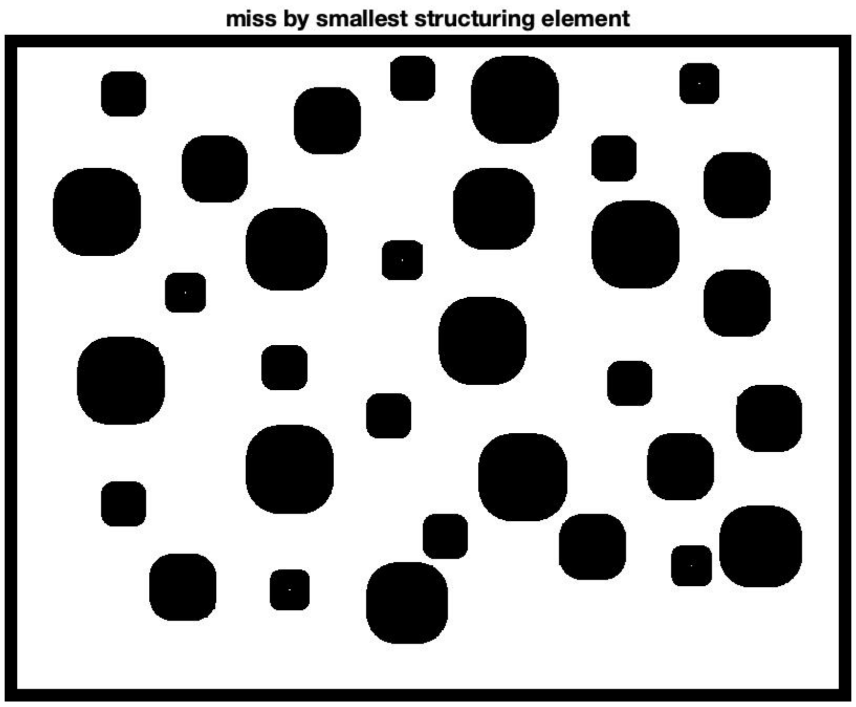


Figure 12: Smallest circles miss by structuring element B

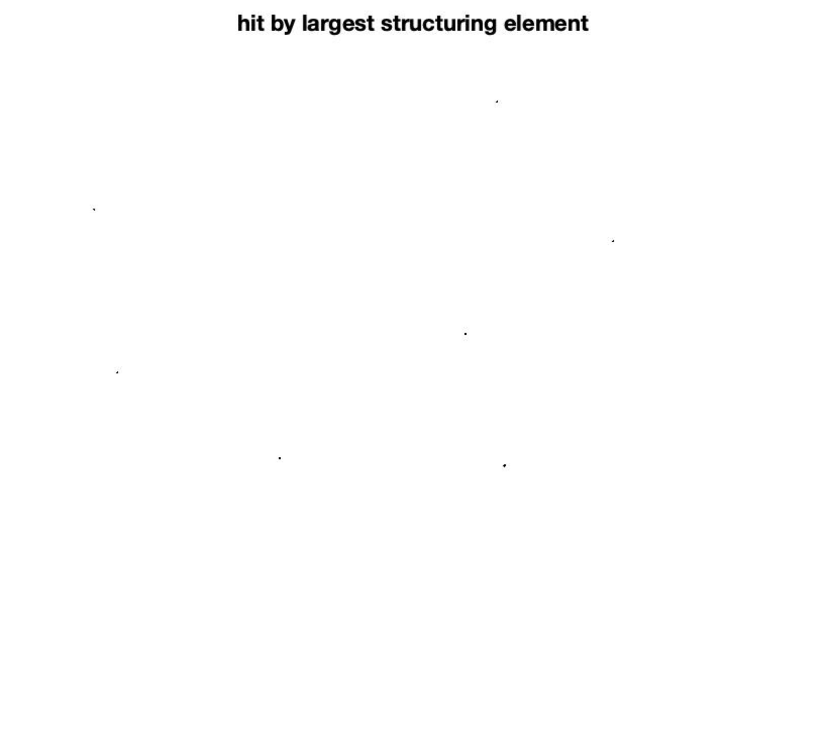


Figure 13: Largest circles hit by structuring element A

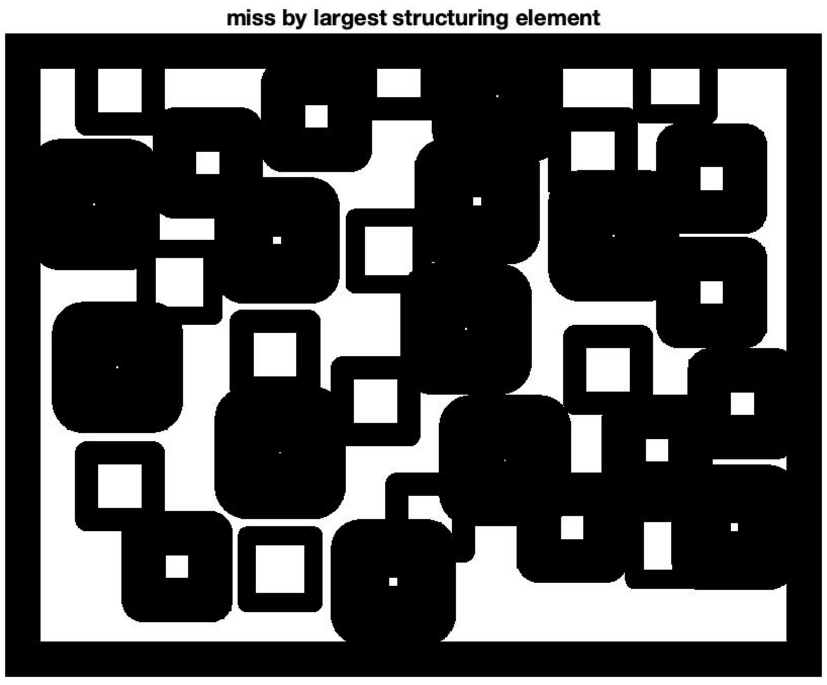


Figure 14: Largest circles miss by structuring element B

Now if we just implement hit-or-miss transform on image without noise reduction. We will get result on Figure 15 ~ Figure 18. As we can see, those noise greatly affect the results.

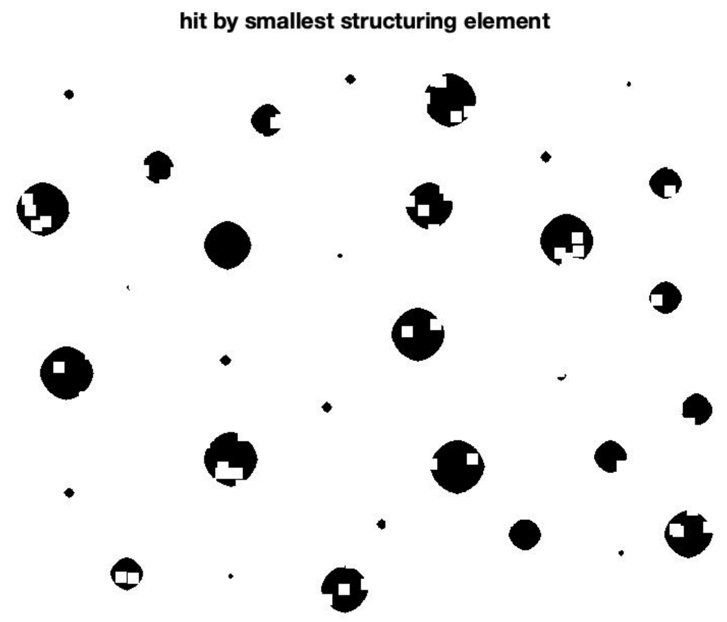


Figure 15: Smallest circles hit by structuring element A on un-denoised image

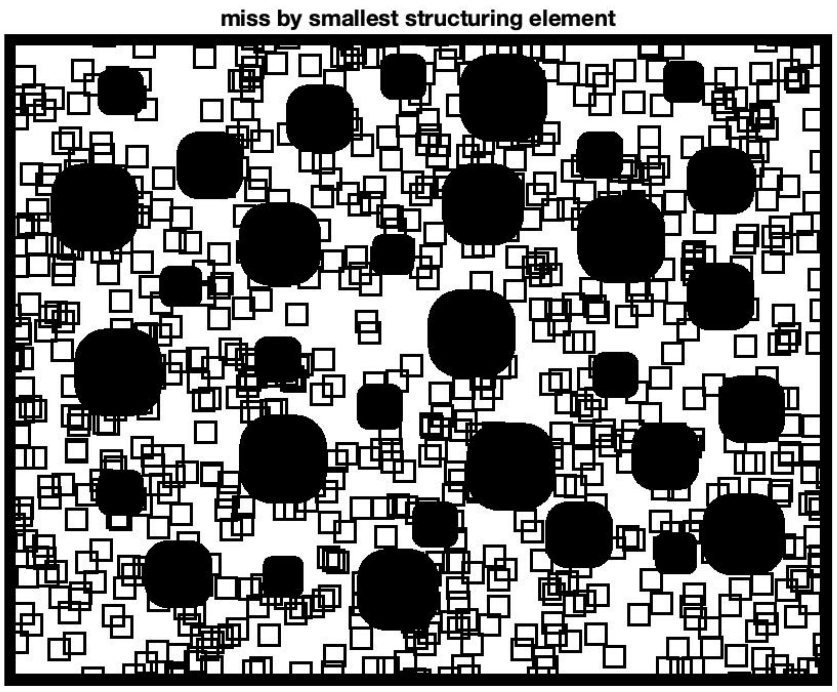


Figure 16: Smallest circles miss by structuring element B on un-denoised image

hit by largest structuring element

Figure 17: Largest circles hit by structuring element A on un-denoised image

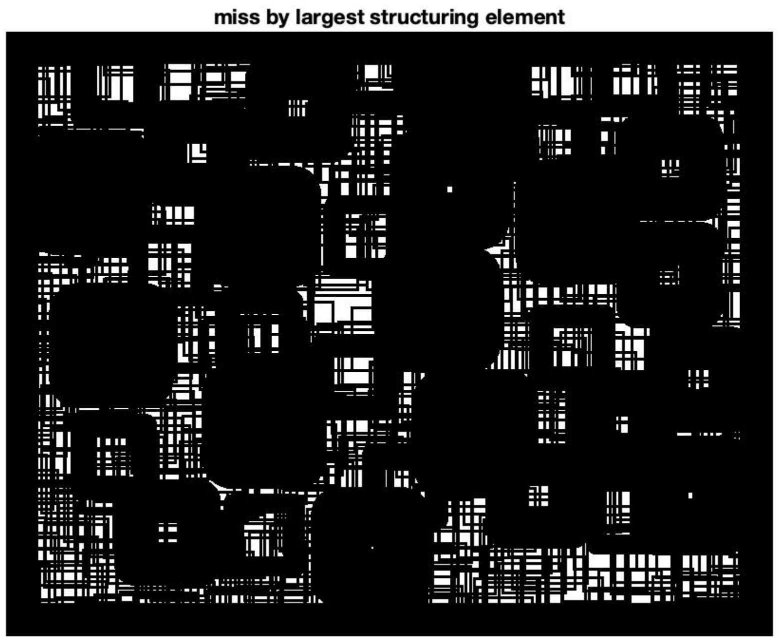


Figure 18: Largest circles miss by structuring element B on un-denoised image

4. Highlight detected circles from original image

Now we have locations of circles of detected circle, by using AND logic of hit and miss, we can find target circles. Red circles in Figure 19 are smallest and largest circles detected. This is the final result by implementing hit-or-miss transform.

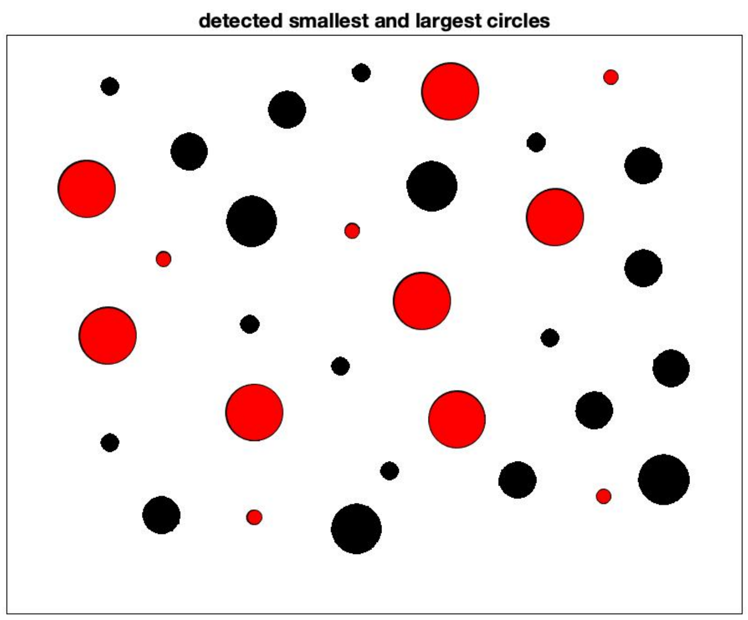


Figure 19: Final result

D. **Conclusions**

In this project, our method is proposed as a sequence of operations to process a random disk image. Before actually implementing hit-or-miss, there are steps that have to be taken so that the final result meets our expectation. Using hit-or-miss transform, we can determine the patterns in the binary image which using the morphological erosion and intersection of two erosions.