**CSE585/EE555:  Digital Image Processing I**

**Computer Project # 1:**

**Mathematical Morphology: Hit-or-Miss Transform**

**Yanxi Yang, Jiuchao Yin, Hongjie Liu**

**Date: 01/30/2020**

A.    **Objectives**

The objective of this project is to realize a “hit-or-miss” morphological operation on a binary image with MATLAB as experiment tool. The goal is to detect disk shapes with the biggest and smallest radius using “hit or miss” operation. The basic knowledge in this project includes: converting RGB image to binary image, closing and opening operations to eliminate “salt and pepper” noise, combination of using erosion and dilation operations, structuring element, set theory to realize “hit-or-miss” transform. All the MATLAB code in this project is source code oriented without using image processing toolbox functions.

B.    **Methods**   (Length: 3-15 pages – be complete!)

**Variables and functions in the code:**

 Input Variables:

    f            input 2D 8-bit image

 Returned Results:

    B            A binary image of f

    new\_B        Image B after noise removal

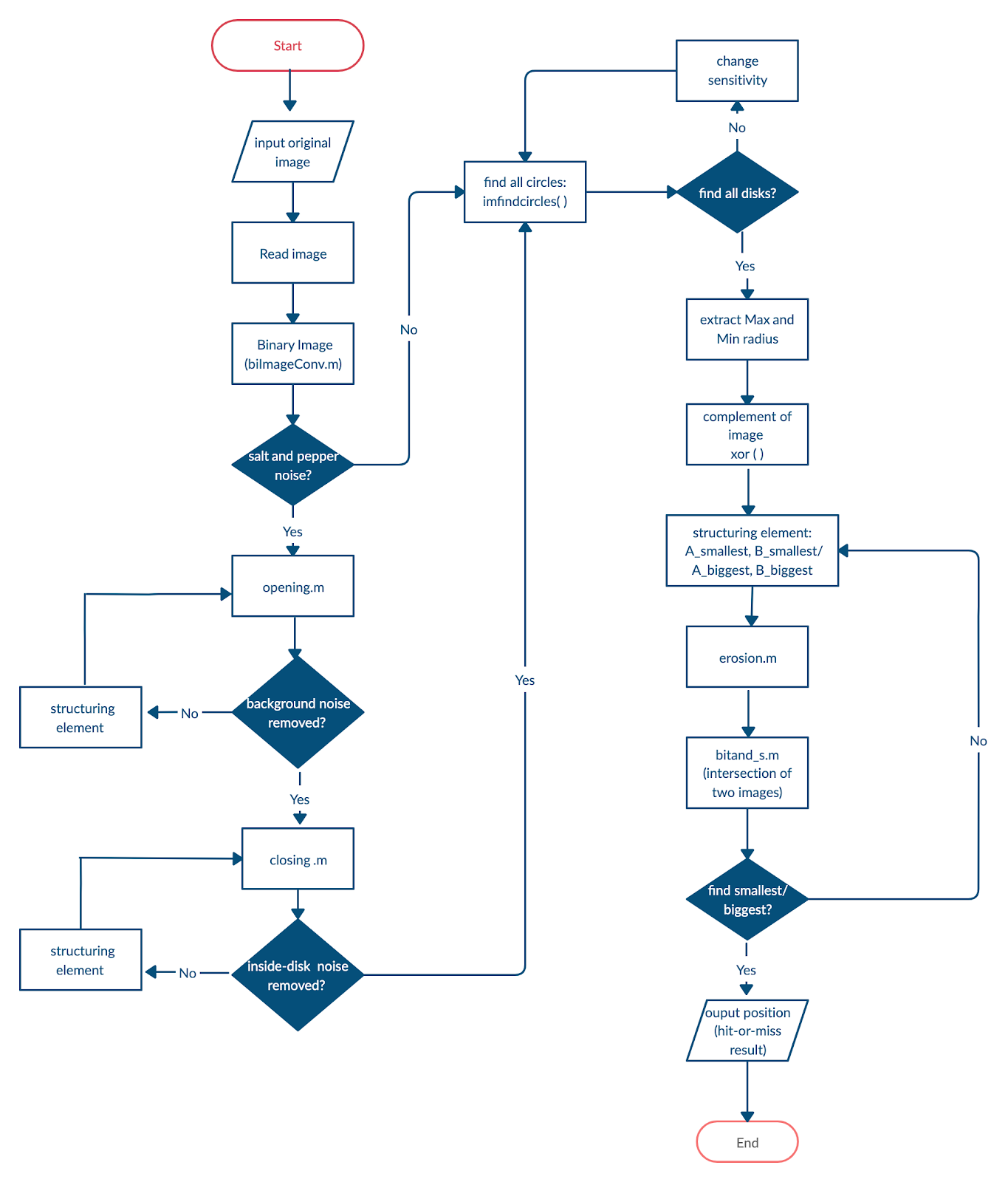
    min\_r        Radius of the smallest disks

    max\_r        Radius of the biggest disks

    smallest     An image indicating locations of smallest disks in B

    biggest      An image indicating locations of biggest disks in B

**Processing Flow chart:**

****

**Process flow architecture:**

1. Load, extract the gray layer, and display the input image

 2. Convert the input image into binary image

 3. use opening and closing to remove noise at the background and  inside disks

 4. Find all disks in the image and get their centers and radius, especially the radius of the smallest and biggest disks: min\_r and max\_r.

 5. Find the locations of smallest disks:

    - Erode the binary image with a structure element: a disk with  radius (min\_r) - 1.

   - Erode the complement of the image with a structure element: a square having a round cavity with radius (min\_r) + 1.

   - Perform bitand of the two resulted matrices

   - The disks with radius in range [(min\_r) - 1, (min\_r) + 1] will be located.

 6. Find the locations of biggest disks:

  - Erode the binary image with a structure element: a disk with radius (max\_r) - 1.

  - Erode the complement of the image with a structure element: a square having a round cavity with radius (max\_r) + 1.

  - Perform bitand of the two resulted matrices

  - The disks with radius in range [(max\_r) - 1, (max\_r) + 1] will be located.

**Methods of functions**

**1.**     **biImageConv.m**

What this function dose is to convert a grey image into a binary image. The input of this function should be a 2D grey image and the output is a 2D binary image with the same size as the input.

First, a threshold should be set to divide each pixel into 0 or 1. If the value of a pixel in the input image is less than the threshold, then in the output image the value of that pixel is 1, which represents the foreground (i.e. the disks in our project); otherwise, the new value of that pixel is 0 which represents the background. As noticed, the original black disks are turned into white since disks are our interest areas, which is better to be set to 1.

Pseudocode:

|  |
| --- |
| Function B = biImageConv (A)  % set the threshold of grey level  threshold = level \* 255;    for each pixel (x, y) in A:         if A(x, y) < threshold  B(x, y) = 1;         else               B(x, y) = 0;        return |

**2.**     **bitand\_s.m**

The purpose of this function is to compute the bit-and operation of two input matrices. The input of this function is two binary image with the same size, and the output is a matrix including the bit-and result of two input images.

This function will traverse all pixels in two input images. The value of a pixel in the output image is the bit-and result of the elements at the same position in two input images. For example, A, B are two same-sized binary images, C is the output. For the pixel at position (i, j), C(i, j) = A(i, j) & B(i, j).

Pseudocode:

|  |
| --- |
| Function C = bitand\_s (A, B)  for each position (x, y) in A and B:        if A(x, y) == B(x, y)  C(x, y) = A(x, y);        else  C(x, y) = 0;         return |

**3.**     **erosion.m**

The purpose of this function is to compute the erosion of an input X with respect to a structuring element A. The input of erosion is a 2D binary image X and a structuring element A. The output is the result image X’ after X eroded with respect to A. One restriction in this function is that the number of rows and columns in the structuring element A must be odd since we suppose A = As can be true.

First, the input image needs to be padded with 0s so that the border of the image can be processed properly. Then, for each pixel (i, j) in X, check whether the structuring element A can be included in X after translated by (i, j). We applied equation (6.2.11) in *Nonlinear Digital Filters, I. Pitas and A.N. Venetsanopoulos* to compute the erosion of X with respect to A.

Pseudocode:

|  |
| --- |
| Function B = erosion (X, A)  pad X with 0s extending half-size-of-A pixels out;  find the border of X before padding;  for each pixel in X:         region = sub-region of X with the same size of A;         if all value in region is 1  B(i, j) = 1;         else  B(i, j) = 0;          return |

**4.**     **dilation\_s.m**

The purpose of this function is to compute the dilation of an input image X with respect to a structuring element A. The output is the result after X dilated by A. There is also a restriction in this function that the number of rows and columns in the structuring element A must be odd.

The process of dilation is very similar to that of erosion. The only difference is that for each pixel in X, check whether the structuring element A can hit X after translation. We applied equation (6.2.8) in *Nonlinear Digital Filters, I. Pitas and A.N. Venetsanopoulos* to compute the dilation of X with respect to A.

Pseudocode:

|  |
| --- |
| Function B = dilation\_s (X, A)  pad X with 0s extending half-size-of-A pixels out;  find the border of X before padding;  for each pixel in X:         region = sub-region of X with the same size of A;         if all value in region is 0  B(i, j) = 0;         else  B(i, j) = 1;          return |

**5.**     **opening.m**

The function of opening is to compute the opening of a binary image X with respect to the predefined structuring element B = {(-1, -1), (-1, 0)}. The process of this function is eroding X with B and then dilating the eroded X with Bs, which conforms the equation (6.3.1) in *Nonlinear Digital Filters, I. Pitas and A.N. Venetsanopoulos.*

Pseudocode:

|  |
| --- |
| Function A = opening (X)  % get the size of X  [m, n] = size(X);  % erode X by B  for each pixel in X:          if X(i-1, j-1) == 1 && X(i-1, j) ==1                  X\_ero(i, j) = 1;          else                  X\_ero(i, j) = 0;    % dilate X\_ero by Bs  for each pixel in X\_ero:          if X\_ero (i+1, j+1) == 1 && X\_ero (i+1, j) ==1                  A(i, j) = 1;          else                  A(i, j) = 0;          return |

**6.**     **closing.m**

The function of closing is to compute the closing of a binary image X with respect to the predefined 2x2 structuring element B = {(-1, -1), (-1, 0), (0, 0), (0, -1)}. The process of this function is dilating X with B and then eroding the dilated X with Bs, which conforms the equation (6.3.2) in *Nonlinear Digital Filters, I. Pitas and A.N. Venetsanopoulos.*

Pseudocode:

|  |
| --- |
| Function A = closing (X)  % get the size of X  [m, n] = size(X);  % dilate X by B  for each pixel in X:          if X(i-1,j-1)==1 || X(i-1,j)==1 || X(i,j) == 1 || X(i, j-1) == 1                  X\_dil(i, j) = 1;          else                  X\_dil(i, j) = 0;    % erode X\_dil by Bs  for each pixel in X\_dil:         if X\_dil(i+1,j+1)==1 && X\_dil(i+1,j)==1 && X\_dil(i,j)==1 && X\_dil(i, j+1)==1                  A(i, j) = 1;         else                  A(i, j) = 0;          return |

**C. Result**

**1. Input image loading and binary image conversion**

We first loaded the 8-bit original image and extracted the grayscale image out (Figure 1). For further processing, the image was converted to binary image with 0.5 as the threshold level (Figure 2). To be specific, pixels with value bigger than 255 \* 0.5 will be changed to 0, while those whose values are smaller than 255 \* 0.5 will be set to 1. Note that in the binary image, we inverted the color of the original images, so that the color of disks is white (represented by 1) and the background is black (represented by 0), because disks are our interest areas, which had better be set to 1.

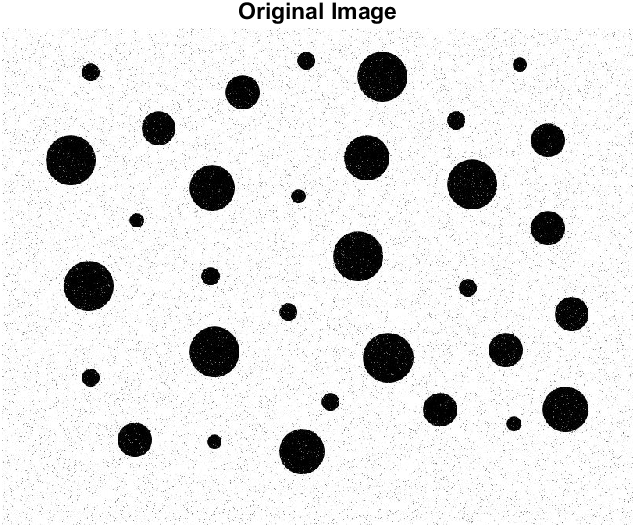


Figure 1 Grayscale image of the original image

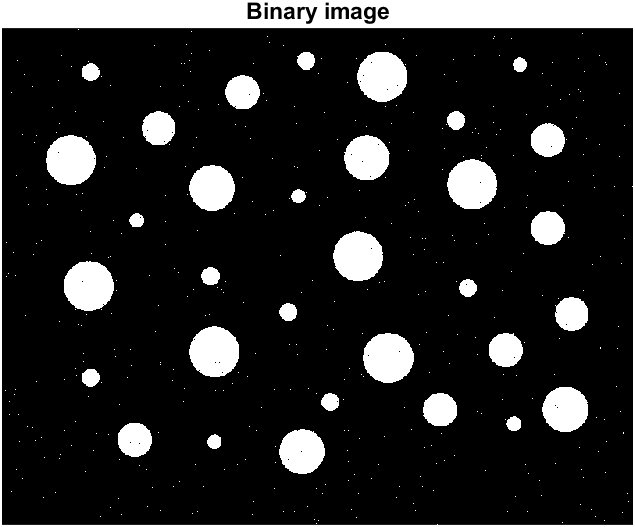
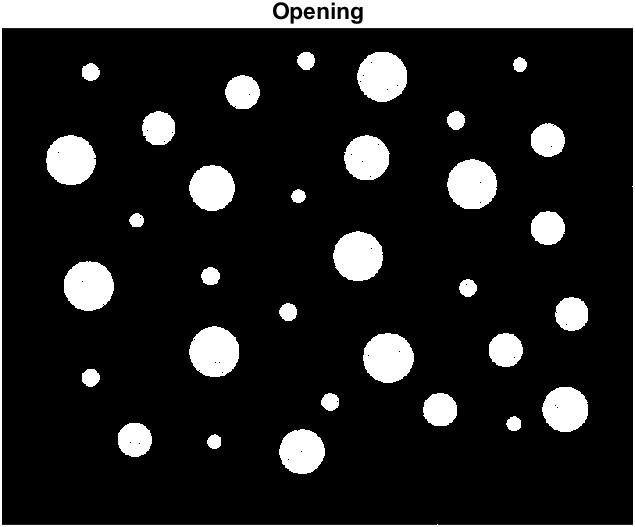


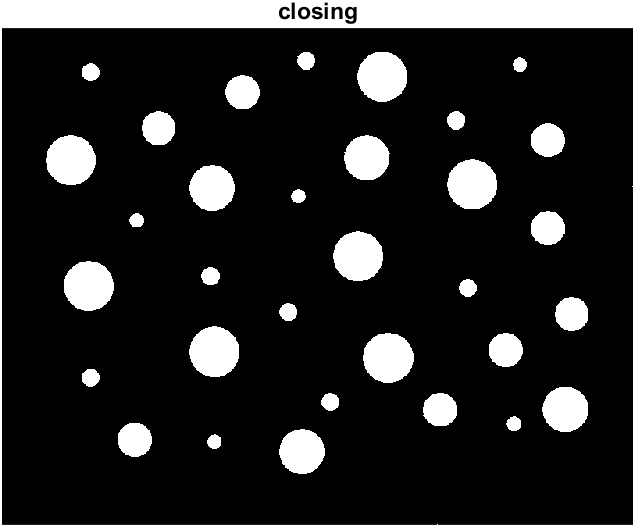
Figure 2 Binary image converted from the original image with threshold 255\*0.5

**2. Remove the salt-and-pepper noise**

As shown by F*igure 1* and *Figure 2,* there were salt-and-pepper noise at the background and inside the disks. The noise will severely affect the identification of disks with hit-or-miss transform. Therefore, we filtered such noise out with small open and close filters. We first applied the {(-1, -1), (-1,0)} open filter to the binary image (*Figure 3*). Opening is involved erosion followed by dilation, which is capable of removing object smaller than the structure element. *Figure 3* demonstrated the small white noise at the background were successfully gone after open filter. In addition to the white noise, there are also small cavities inside disks to be filled. We achieved this by further applying close filter {(-1, -1), (-1, 0), (0, 0), (0, -1)} to the image after opening. Any holes smaller than the close filter will be filled. From *Figure 4*, we can see, all the cavities were filled and all disks became solid.



*Figure 3 Binary image after {(-1, -1), (-1,0)} open filter*

**

*Figure 4 Binary image after further closing with structural element {(-1, -1), (-1, 0), (0, 0), (0, -1)}*

**3. Identify radius of the smallest and the biggest disks**

Having preprocessed the image, we next identified the radius of all the disks. Especially, we need to figure out radius of the smallest and biggest disks. As described in the method and shown in Figure 5, imfindcircles function enabled us to find both centers and radii of all circles within given radius range. As shown in Table 1, the radii of all disks fell into five groups (r1 ≈ 9.0, r2 ≈ 11.5, r3 ≈ 22, r4 ≈ 29, r5 ≈ 32). Therefore, the smallest radius is around 9, and the biggest radius is around 32.

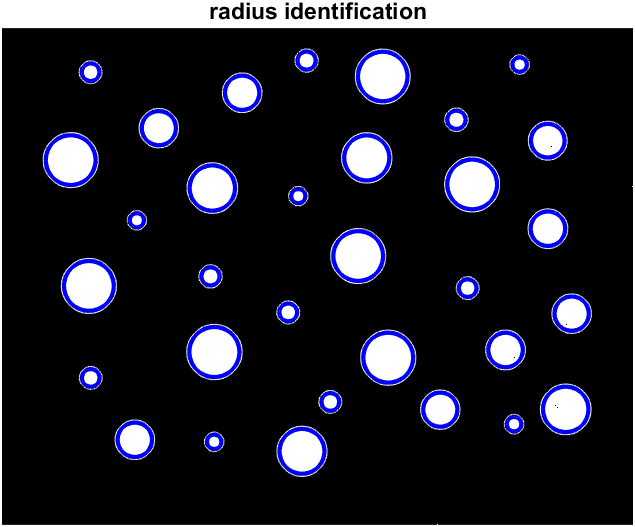


Figure 5 Circle capture with imfindcircles

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r1 | r2 | r3 | r4 | r5 |
| 9.021061 | 11.26251 | 21.48988 | 28.70378 | 31.68008 |
| 9.043704 | 11.26987 | 21.81422 | 28.83285 | 31.81814 |
| 9.155122 | 11.28027 | 21.81422 | 29.00218 | 31.81995 |
| 9.18997 | 11.29771 | 21.90512 | 29.05537 | 31.82826 |
| 9.229171 | 11.32621 | 21.96168 |  | 31.83494 |
|  | 11.53776 | 21.96443 |  | 31.9037 |
|  | 11.54774 | 22.04442 |  | 32.02316 |
|  | 11.66112 | 22.05914 |  |  |

Table 1 All radius identified with imfindcircles

**4. Detect the smallest and largest disks with hit-or-miss transform**

With the information of the smallest radius, which is around 9, we constructed two structural elements (*Figure 6*). A-smallest is a disk with radius 8, while B-smallest is a square with a hole, of which the radius is 10. To detect the smallest disks, we first computed erosion of the preprocessed image with respect to A-smallest. Disks smaller than A-smallest were removed at this step. Then with the complimentary image of the preprocessed image, we computed erosion with respect to B-smallest. This step will eliminate all disks having larger radii than 10. Lastly, the two resulted images were intersected with each other. In summary, the three steps detect disks with radii ranging from 8 to 10. Notably, even though the radius of smallest disks is around 9, they are not the same as shown in Table 1. This could be resulted from information loss during binary image conversion. If the disk in A-smallest and the cavity in B-smallest are of the same size, we can only detect disks that are exactly the same with the disk in A-smallest. In that way, we might not be able to find all small disks in this project. Therefore, we used a range [8,10] for the smallest disk detection. *Figure 7* below demonstrated that we successfully detected 5 disks. By checking corresponding positions in the original image (*Figure 8*), we further confirmed that the 5 identified positions were indeed the where the 5 smallest disks located.

Similarly, we constructed two structural elements A-biggest and B-biggest to detect largest disks (*Figure 9*). The radius of the disk in A-biggest is 31, while the radius of the hole in B-biggest is 33. With the same method discussed above, we are able to locate any disks with radius ranging from 31 to 33. This range was chosen based on the fact that the largest disks have radius ≈ 32. As shown in *Figure 10* and *Figure 11*, we successfully identified the 7 largest disks.

Together, we have identified 5 smallest disks and 7 largest disks from the original image.



Figure 6 Structural element for finding the smallest circles

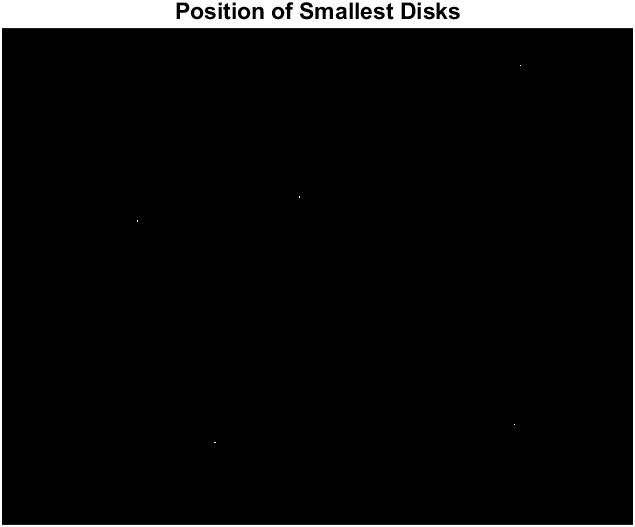


Figure 7 Positions of smallest disks detected (highlighted with red box)

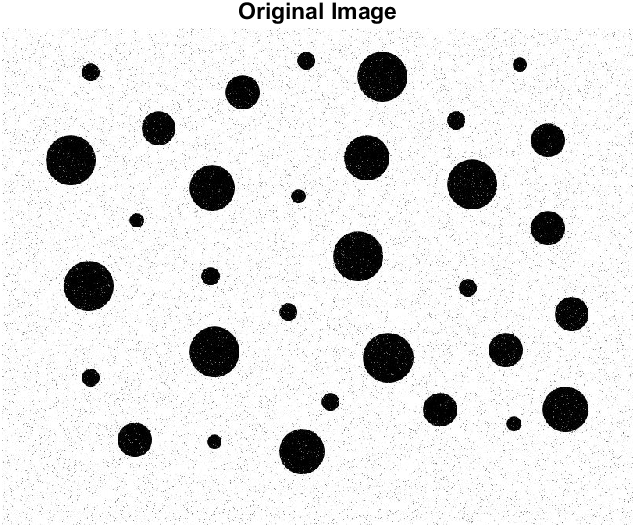


Figure 8 Corresponding smallest disks in the original image



Figure 9 Structural element for finding the largest disks

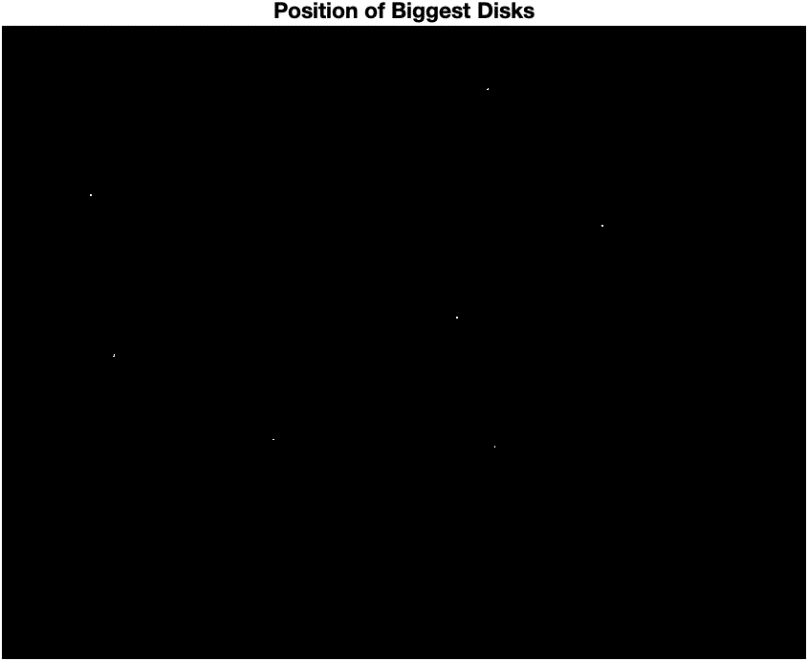


Figure 10 Positions of biggest disks identified

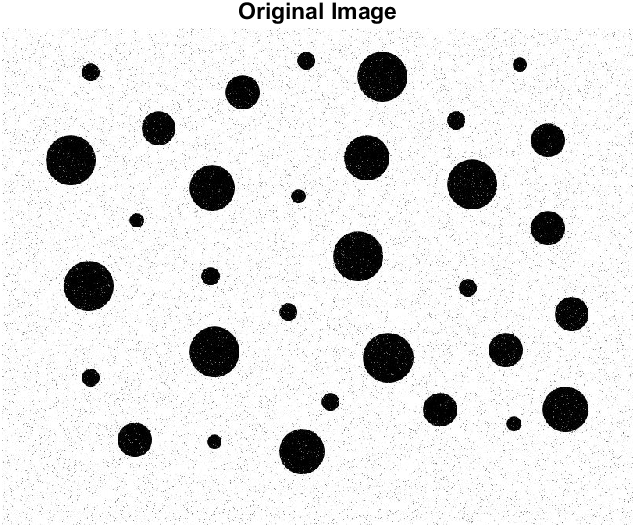


Figure 11 Corresponding biggest disks in the original image

**5. Try to detect the smallest and largest disks without close/open filer**

As mentioned above, we removed slat-and-pepper noise, because their presence could severely affect disk detection. The algorithm we used in this project is based on hit-or-miss transform, which involves two main steps.

During the first step, we erode the image with structural element A. The result is a collection of points. When we translate A with any of these points, the translated result will be included in the image. In this way, disks smaller than A would be filtered, because the translated A cannot be included in a smaller disk. However, if there are small cavities in the disk, even if disks are equal to or larger than A, the translated A will not be considered to be included in the disk, because A itself is a solid disk. As a result, we will filter out our targets as well due to the presence of cavities. Therefore, we have to fill in cavities inside disks.

During the second step, we performed erosion of the complementary image with structural element B. This result is also a collection of points, with which if we translated B, the translated B will be included in the complementary image. Similar to step one, if there are noise at the background, there will be cavities in the complementary image. Again, the translated B would not be considered to be included in the complementary image, because B is solid. Consequently, we must remove noise at the background to correctly detect our targeted disks.

We tried to run the same procedure with the binary image that kept all the noise. *Figure 12* demonstrated that only three smallest disks were detected. When I took a closer look at the three disks in the binary image, they did not contain any noise inside, nor did the background close to them have noise. Note that some noise was gone during binary image conversion. *Figure 13* showed that no largest disks were detected at all. This is reasonable, because all the largest disks have cavities inside them.

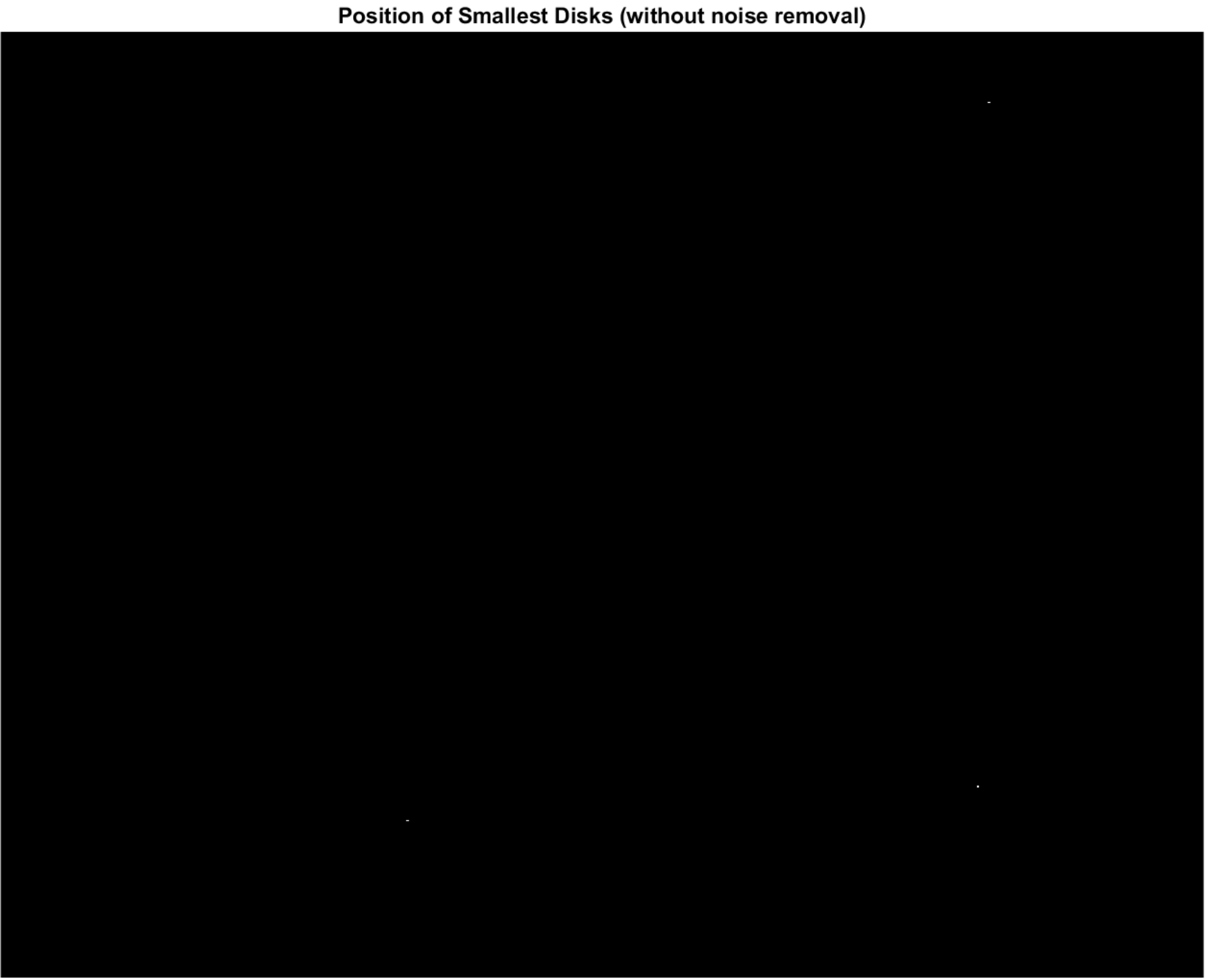


Figure 12 Positions of smallest disks without noise removal

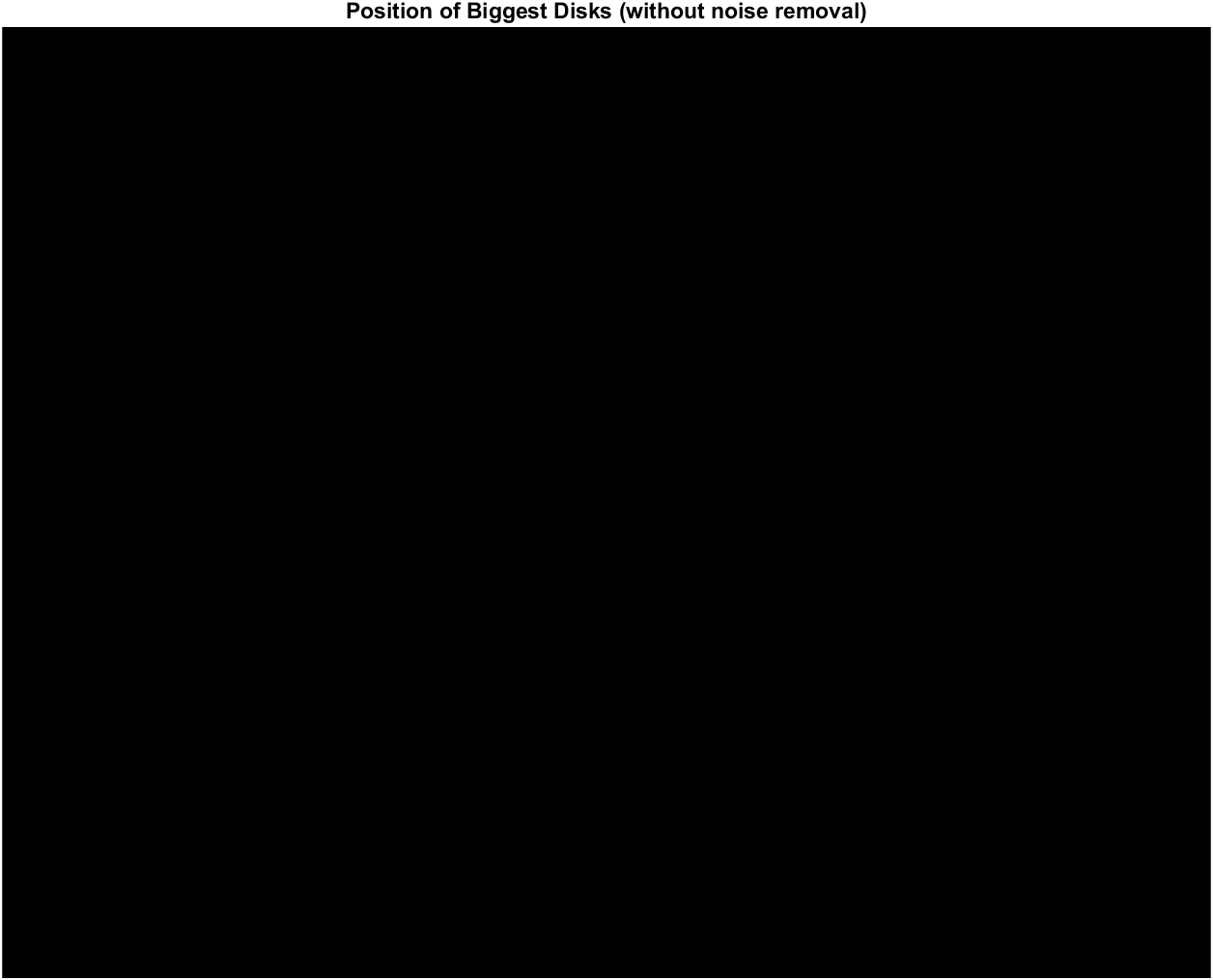


Figure 13 Positions of largest disks without noise removal