## CSE585/EE555:  Digital Image Processing II

## Computer Project # 1:

## Morphology: Hit-or-Miss Transform

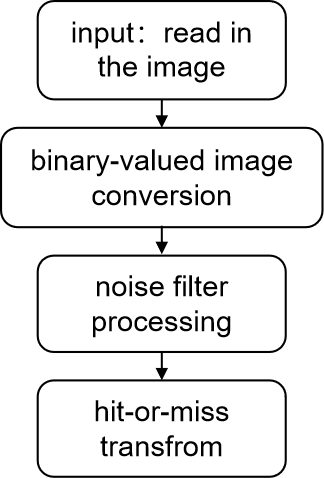
#### Hao Zhou, Fan Yang, Zerui Hua

#### Date: 01/25/2023

* + 1. **Objectives**

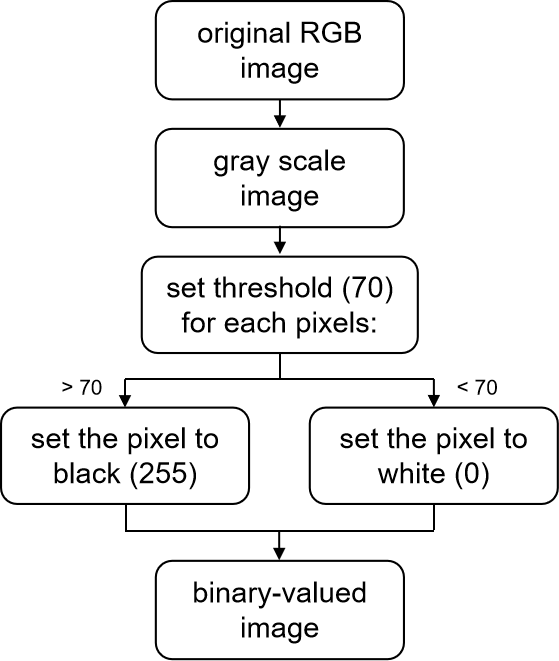
1. Learn how to use MATLAB for digital images processing;
2. Learn how to use math morphology and achieve morphological operations from simple (erosion / dilation) to complex (hit or miss);
3. Learn how the hit-or-miss operation can be used in the object detection in digital images.
   * 1. **Methods**
4. Algorithm and Theory

The main function of this project mainly consists of three parts, including the binary-valued image conversion, the noise filtering and the hit-or-miss transform operation, as shown in the flow chart.

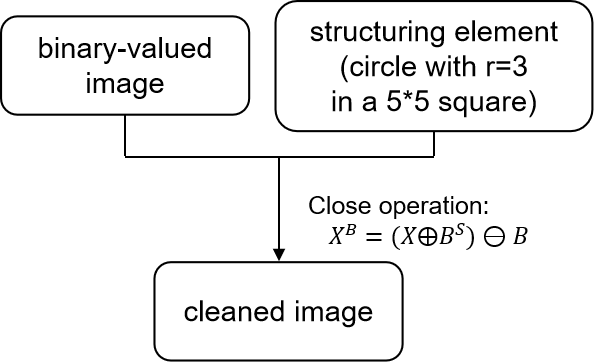


We will discuss these parts in the following report separately.

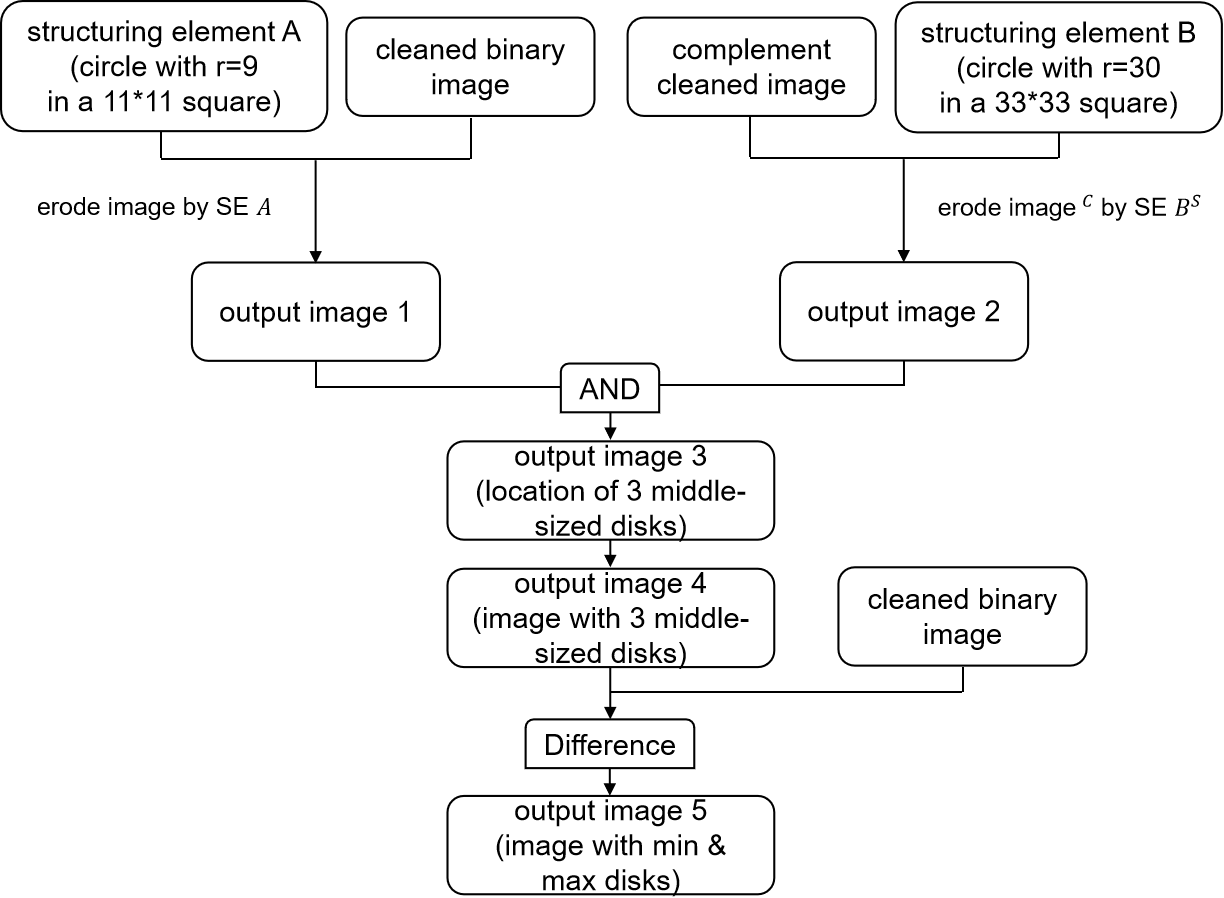
1. Image conversion: This part can first convert the RGB image into gray scale image. Next set a threshold value, classify the value of each pixels, and then generate the binary-valued image.



1. Filter processing: In this part, we set a new function ‘create\_se’ , which can create a symmetric structuring element (a circle inside a square). With it, we can implement a closing operation in order to remove the salt-and-pepper noise. The finally output will be a cleaned image without noise.



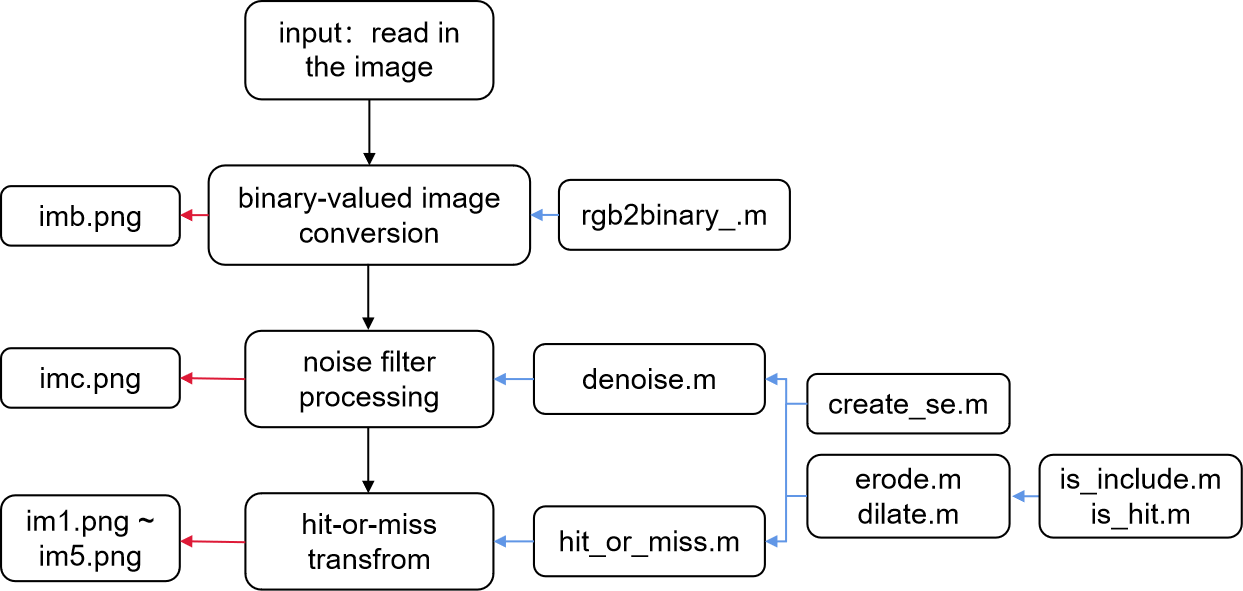
1. Hit-or-miss transform: The main purpose of this part is to implement hit-or-miss operations on image to find the largest and smallest disks in the image. First, introduce two SEs of a specific size and erode the image to get the location of the 3 middle-sized disks. Then extend the middle-sized disks into their original size. Take the difference with the original image and finally can obtain the image only containing the largest and smallest disks.



1. MATLAB Implementation

In this section, we will show the structure of the MATLAB code and how to run the code correctly.

The overall structure of the ‘main.m’ code is shown as the flow chart. The black arrow represents the process, the blue arrow represents which function file need to be called, and the red arrow represents the output of the current step.



There is an image ‘RandomDisks-P10.jpg’ and total 9 files in the ‘project’ folder, including the ‘main.m’, ‘rgb2binary\_.m’, ‘denoise.m’, ‘hit\_or\_miss.m’, ‘create\_se.m’, ‘erode.m’, ‘dilate.m’, ‘is\_include.m’ and ‘in\_hit.m’. To run the code correctly, please put all the files and image in one folder. Then just run the ‘main.m’ file, the code will automatically generate a folder named ‘results’ and put all outputs in it.

Next, we will introduce the MATLAB code in detail and analyze how to get the results step by step.

1. Read in the original image 'RandomDisks-P10.jpg';
2. Call the ‘rgb2binary\_’ function to convert the original image into binary-valued image. This function will first use the ‘rgb2gray’ command to convert the RGB image into gray scale image. Then set a threshold value equal to 70, classify the value of each pixels. If value of a pixel is greater than 70, then we’ll set it to 255 (black), otherwise we’ll set it to 0 (white);
3. Then we can generate the binary-valued image from the original RGB image. Finally, because in the following function, the calculation in erode and dilation operation of the foreground will be between 0 and 1 (0 for foreground and 1 for background), we need to transfer the data type of the image into bool type. The output of this function is saved as ‘imb.png’;
4. Call the ‘denoise’ function to remove the salt-and-pepper noise. In this function, we also call other three functions ‘create\_se’, ‘erode’ and ‘dilate’ to implement closing operation;
5. The ‘create\_se’ is the function to create a symmetric SE based on the value we entered. In the denoise part, we set a small SE with a circle with r = 3 inside a 5 \* 5 square. (Make sure that the SE is small enough (far less than the minimum disk) to filter the noise without affecting the disk);
6. Then we call the ‘erode’ and ‘dilate’ to generate the close filter. The closing formula is: • B. The ‘erode’ and ‘dilate’ will loop through all pixels in the image, and call the other two functions ‘is\_include’ and ‘is\_hit’ respectively to check the whether the SE is included in or hitting the image. If so, set the pixel to foreground, otherwise set it to background. With this, we can achieve the erosion and expansion operation;
7. After denoise the image, we can get the cleaned image, which is saved as ‘imc.png’;
8. Next, we call the ‘hit\_or\_miss’ function to implement hit-or-miss transform to detect the location of 3 middle-sized disks. During this function, we still need ‘create\_se’, ‘erode’ and ‘dilate’ function as well;
9. The formula for hit-or-miss transform is: . So first we need to create two SE A and B. Because we need to locate the middle-sized disks, so we set A as a circle with r = 9 inside a 11 \* 11 square, which is bigger than the smallest disk. B is a circle with r = 30 inside a 33 \* 33 square, which is smaller than the biggest disk;
10. Next, we implement the hit-or-miss transform and get the location of 3 middle-sized disks. The results of image and will be saved as ‘im1.png’ and ‘im2.png’. The AND of these two images, which is the output of hit-or-miss transform, will be saved as ‘im3.png’;
11. Then, in order to obtain the smallest and largest disks, we implement a ‘for’ loop, which is used to extend middle-sized disks to their original sizes. The loop will detect each pixels, if encounter a foreground pixel, we extend it based on the second largest disk’s radius. And compare it with the image ‘imc’ to zero the redundant part of the expansion. Thus, we can get an image that contain 3 middle-sized disks, which is saved as ‘im4.png’;
12. Take the difference set between cleaned image ‘imc’ and the image with middle-sized disks ‘im4’, we can finally obtain the image with only the smallest and largest disks. This image will be saved as ‘im5.png’.
    * 1. **Results**   
         1. System Output

In this section, we will show all the result images.

Figure 1 is the original RGB image with noise.

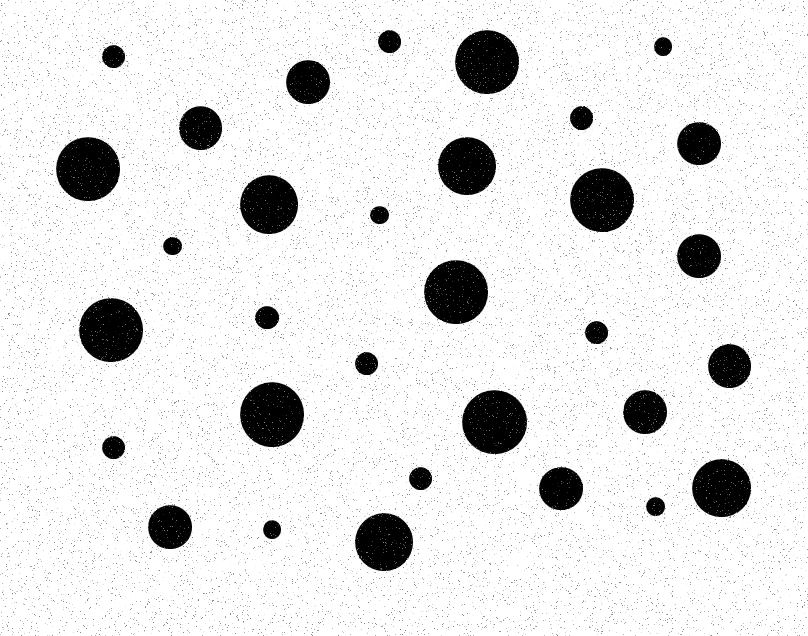


Figure 1. Original image “RandomDisks-P10”

Figure 2 “imb” is the result after convert the Figure 1 into binary image.

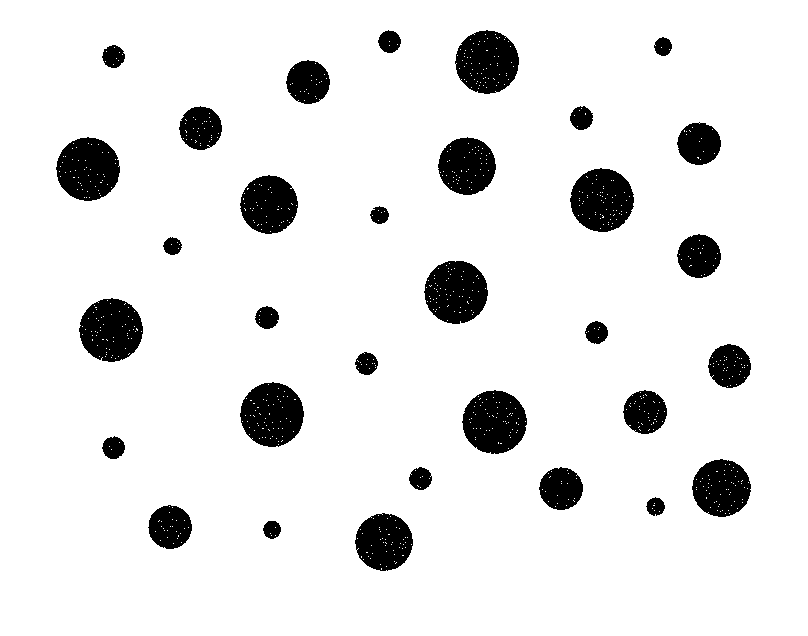


Figure 2. Binary-scale image “imb”

Figure 3 is the small structuring element generated during the denoising operation. The radius of the middle circle is 3 and the length of the square is 5.

se

Figure 3. SE for denoise operation

Figure 4 “imc” is the result after remove the salt-and-pepper noise in Figure 2.

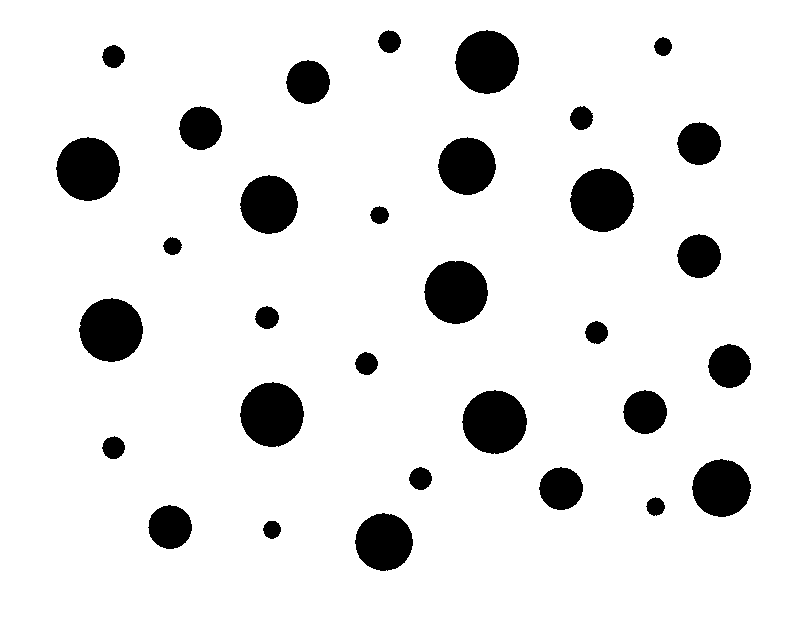


Figure 4. Clean image “imc”

Figure 5 and Figure 6 are two SEs generate for hit-or-miss transform operation. SE A is a circle with radius of 9 inside a square with width of 11. SE B is a circle with radius of 30 inside a square with width of 33.

seA

Figure 5. SE A (bigger than the smallest disk)

seB

Figure 6. SE B (smaller than the biggest disk)

Figure 7 “im1” is the result of the erosion of clean image “imc” X by SE A.

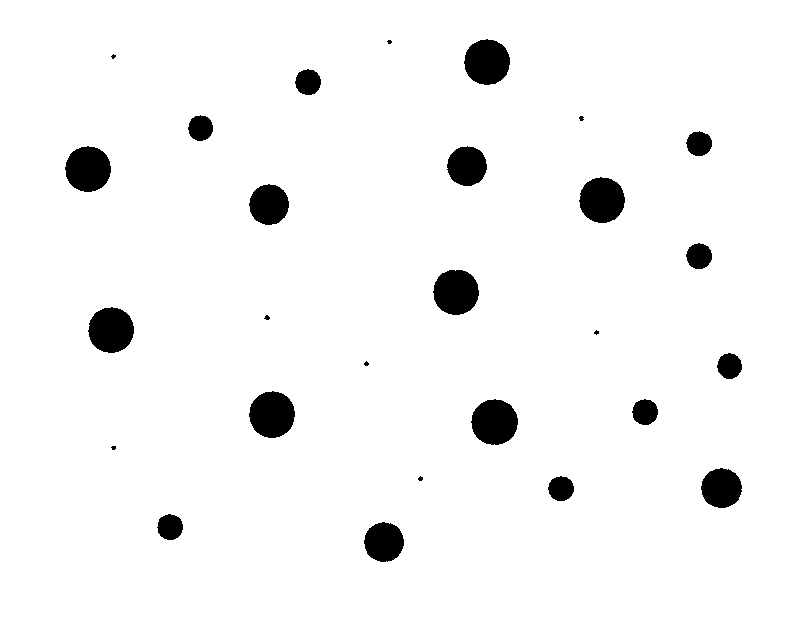


Figure 7. “im1”, the result of X erode A

Figure 8 “im2” is the result of the erosion of the complement of clean image “imc” X by the symmetry of SE B

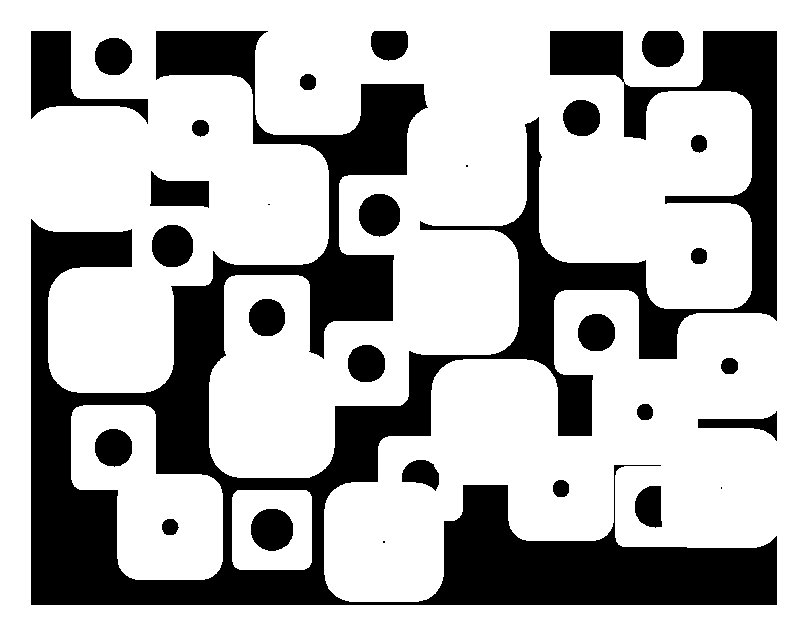
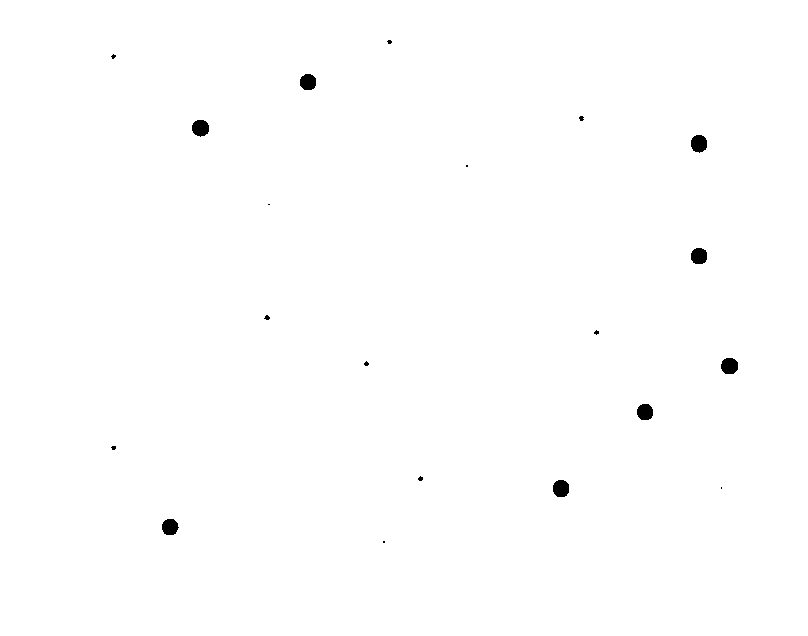


Figure 8. “im2”, the result of erode

Figure 9 “im3” is the result of the AND of “im1” and “im2”, which is the location of the 3 middle-sized disks after operate the hit-or-miss transform.



**Figure 9.**  “im3”, the location of the 3 middle-sized disks

Figure 10 “im4” is the result of the extension of the middle disks to their original sizes.

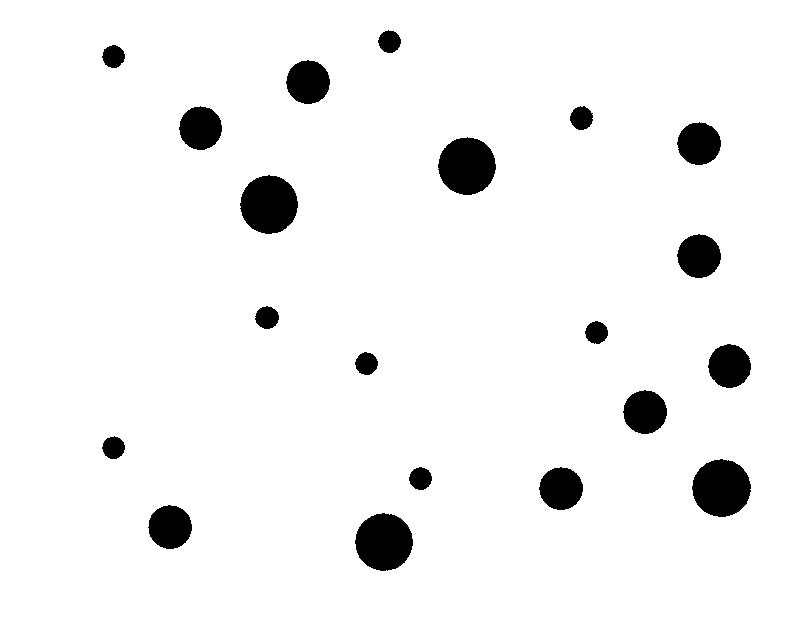


Figure 10. “im4”, the image with 3 middle-sized disks

Figure 11 “im5” is the result of the difference set between the clean image ‘imc’ and ‘im4’, which is the image with the smallest and largest disks.

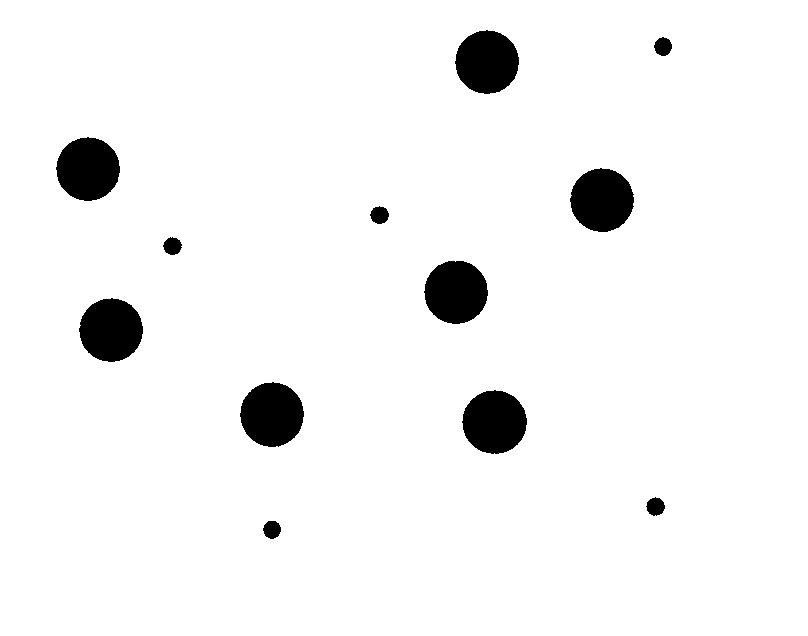


Figure 11. “im5”, the final result, image with the smallest and largest disks

2. Result Analysis

(a) About the threshold:

In the image conversion phase, we set a threshold to convert the RGB image into binary-scale image. In this part, we set the threshold value to 70. If value of a pixel is greater than 70, then we’ll set it to 255 (black), otherwise we’ll set it to 0 (white).

We used the ‘imtool’ command to observe the results and found that too large or too small threshold will affect the shape and size of the original disk. 70 is a suitable value, which can ensure that the shape of the disk does not change, and there are no pixels of other colors in the white background.

(b) About the filtering of salt-and-pepper noise:

For the denoise operation, we implement the closing operation as a filter. The basic formula of closing is • B. Thus, we created a SE which is small enough to filter the noise without affecting the disk

(c) The selecting of structuring elements A and B:

In order to select the middle-sized disks, we need to choose the SE depend on the sizes of the biggest and the smallest disks. So we set A to be bigger than the smallest disk and B to be smaller than the biggest disk. After the erosion operation, we eliminated the biggest and the smallest disks and got the image with the middle-sized disks. As for the value of A and B, we used ‘imtool’ command to measure the sizes of each disks, and selected the parameters according to this.

(d) If the closing operation is not applied:

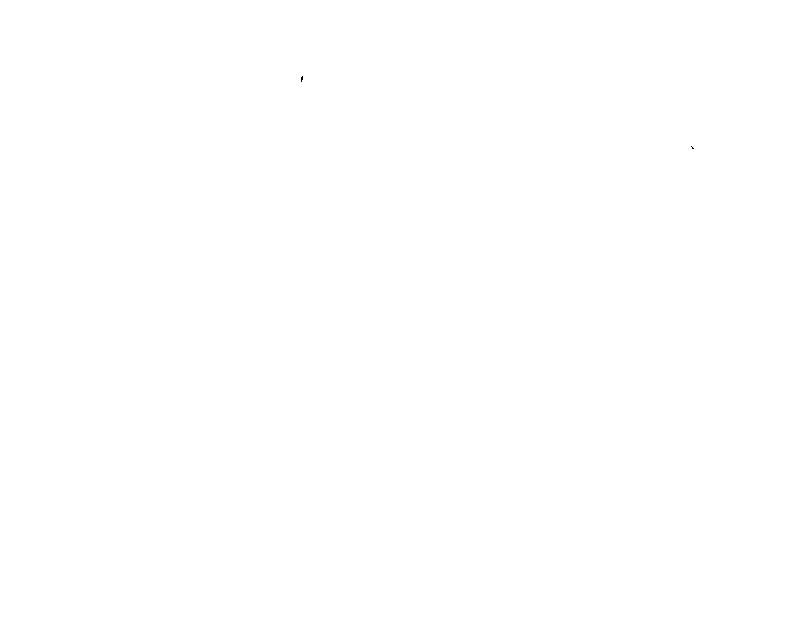


Figure 12. the hit-or-miss result when the closing operation is not applied

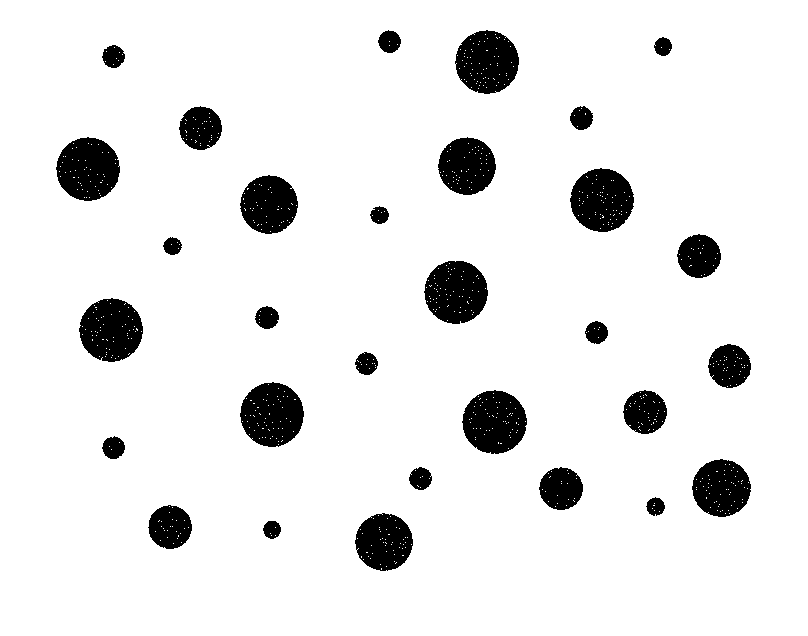


Figure 13. the final result when the closing operation is not applied

As shown on the previous figure, our hit-or-miss transform does not work when we do not apply the close filter. The close filter greatly affects the result of denoising, and ultimately affects the hit-or-miss and final results. The noise makes the image X unable to correctly eliminate the specific disks when erode by the SEs A and B. Thus the hit-or-miss dose not work at all.

* + 1. **Conclusions**   
       As the results shown above, we achieved the task of detecting and saving the positions of the biggest and smallest disks from the original image. The results satisfy the requirements of the project.

After completing this project step by step, we learned how to perform simple operations on images in MATLAB, such as the conversion of RGB images to grayscale images and binary-valued images. At the same time, we implemented the close filter through the basic erosion and dilution operations to remove the noise. And through set theory, we processed the hit-or-miss transform for easy implementation in MATLAB coding. During this project, we understand that morphological operations can be used to detect the position of the objects or elements. We can use morphological operation to detect and operate specific elements in the image under certain conditions.