## CSE585/EE555:  Digital Image Processing II

## Computer Project # 1:

## Morphology: Hit-or-Miss Transform

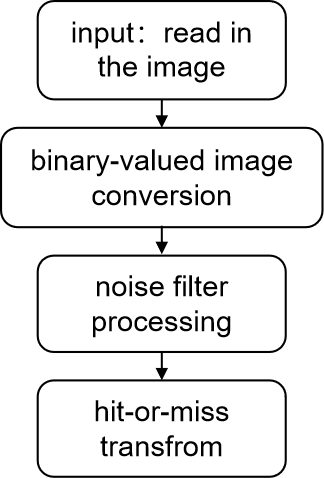
#### Hao Zhou, Fan Yang, Zerui Hua

#### Date: 01/25/2023

* + 1. **Objectives**

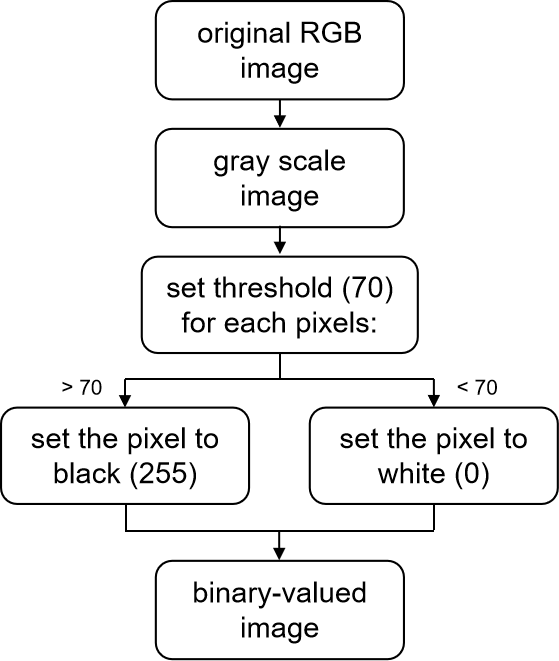
1. Learn how to use MATLAB for digital images processing;
2. Learn how to use math morphology and achieve morphological operations from simple (erosion / dilation) to complex (hit or miss);
3. Learn how the hit-or-miss operation can be used in the object detection in digital images.
   * 1. **Methods**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

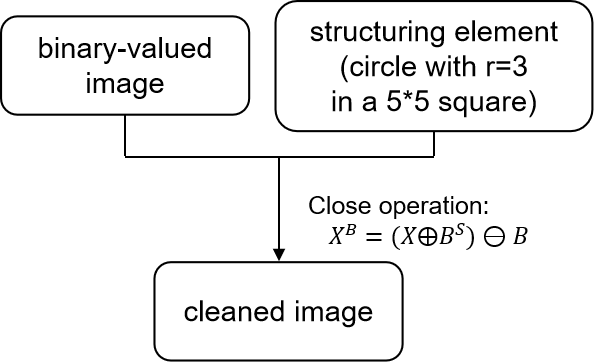


We will discuss these parts in the following report separately.

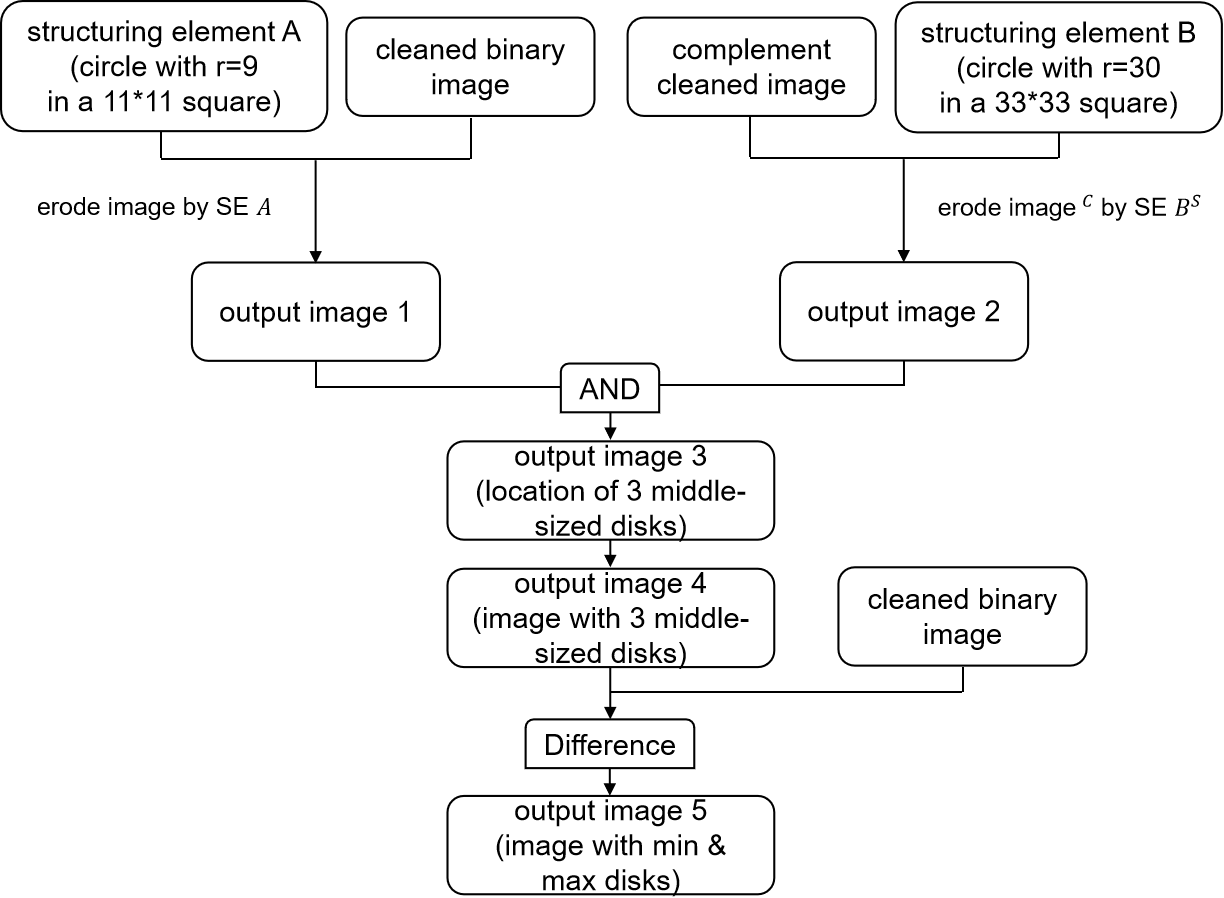
1. Image conversion: This part can first convert the RGB image into gray scale image. Next set a threshold value, classify the value of each pixels, and then generate the binary-valued image.



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.



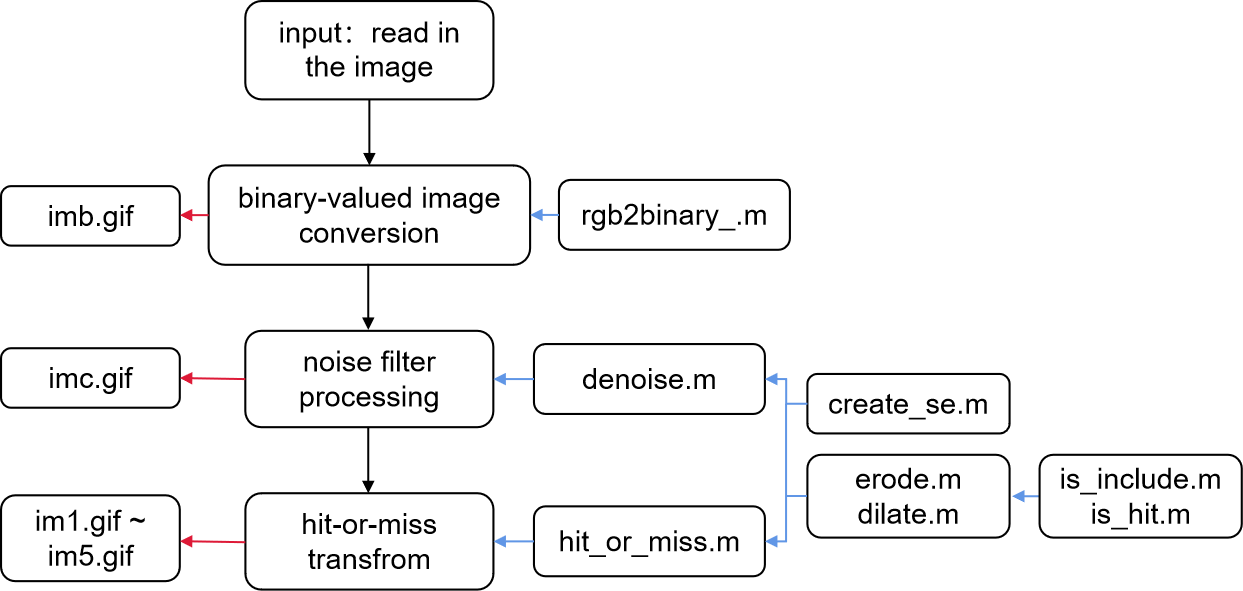
1. Hit-or-miss transform: The main purpose of this part is to implement hit-or-miss operations on image to find the largest and smallest disks in the image. First, introduce two SEs of a specific size and erode the image to get the location of the 3 middle-sized disks. Then extend the middle-sized disks 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.gif’;
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.gif’;
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.gif’ and ‘im2.gif’. The AND of these two images, which is the output of hit-or-miss transform, will be saved as ‘im3.gif’;
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 ‘imc.gif’ image 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.gif’;
12. Take the difference set between cleaned image ‘imc.gif’ and the image with middle-sized disks ‘im4.gif’, we can finally obtain the image with only the smallest and largest disks. This image will be saved as ‘im5.gif’.
    * 1. **Results** (Length: Could be many pages, including figures and tables)  
          Give all results here, be they figures, tables, or other numerical results.

Images can be in .gif format (or any other lossless format) – .jpg will affect quality because it is a **lossy** compression method.  The gray-scale images in the “*images.zip*” archive on CANVAS are mostly in .gif format.

For MATLAB, the *imread*, *imwrite*, and *imtool* commands do most of what you need to generate pictorial results.

Figures must have **complete captions (!)**, **giving image names, processing applied, and parameters used – see examples below**.

No more than **4 images** per page!

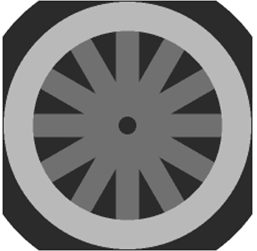


Figure 1. Original "wheel" image.

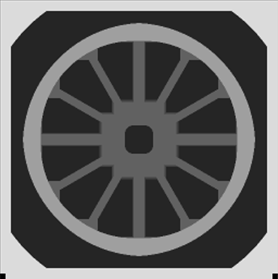


Figure 2. Result after applying an 11X11 erosion (minimum)

operator to the original "wheel" image.

Give detailed observations and/or analysis of all results.

Also, ask yourself the following general questions:

1. Are the results what you were expecting they would be? If not, why not?
2. Do you think it's because the algorithm is not the appropriate one, or because your implementation has a bug?

**Be sure to answer any questions the project write-up may pose**!

It is important that you realize when your results are wrong – this means that you understood the theory behind the project. **Partial credit is given for incorrect results when the error is pointed out and thorough explanations are given** for the incorrect results, along with possible corrections or solutions.

* + 1. **Conclusions** (Length: 1-2 paragraphs)  
       Write a clear short conclusion for your project.

Remember, conclusions **are not** observations.   You draw conclusions **from** what you observe in your results.

This section does not have to be long!