Assignment 3

Visual Computing Fundamentals: Image Processing

Pablo Díaz Viñambres : *pablodi@ntnu.no*

November 24, 2022

Contents

1 Task 1: Theory

(a) Define *opening* and *closing* in terms of *erosion* and *dilation*. What happens when open and closing are applied multiple times on the same image?

The opening of an image A (or, more generally, a set) by a structuring element B is defined as:

$$A \circ B = (A \ominus B) \oplus B$$

that is, the opening is the dilation by B of the erosion of A by B.

Conversely, the closing of A by B is defined as:

$$A \bullet B = (A \oplus B) \ominus B$$

that is, the closing is the erosion by B of the dilation of A by B.

When opening and closing are applied multiple times on the same image, a lot of noise can be eliminated. In particular, the erosion stages would remove background noise and the dilation stages help with foreground noise (holes or gaps).

(b) Smoothing of an image is often done before performing edge detection. What is the purpose of smoothing the image before edge detection?

Smoothing can reduce noise, which helps in edge detection since the derivatives of the intensity can become extremely irregular when a lot of noise is present.

(c) The Canny edge detector uses a method called *hysteresis thresholding*. Shortly explain how hysteresis thresholding work.

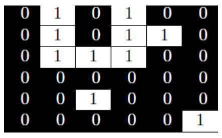
This method works by creating two images $g_{NH}(x, y)$ and $g_{NL}(x, y)$ that contain all of the original image pixels above a high threeshold T_H and a low one T_L, respectively. Since $T_H > T_L$, we have $g_N L(x, y) \geq g_N H(x, y)$, and we perform $g_N L(x, y) = g_N L(x, y) - g_N H(x, y)$. We know have a collection of marked pixels above a threeshold T_H on the original image and another collection of pixels between T_H and T_L.

We now take every pixel marked in g_{NH} as a valid edge pixel, add it to the final result and check if in its neighbourhood there exists a marked pixel in g_{NL}. Neighbourhood here can mean 8-connectivity or following the directions perpendicular to the gradient normal. If it does, we also add it to the final result.

(d) Why do we use hysteresis thresholding instead of a single threshold?

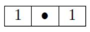
Because with a single threshold, it is very likely to get *false positives*, in the case the chosen threeshold is too low, or *false negatives*, if it is too high. With hysteresis thresholding, we take into account the continuous nature of edges and the previous gradient calculation in the Canny edge detector, making it more precise. It also returns linked edges, which are very useful in object recognition.

(e) Determine the dilation $A \oplus B$ of the binary image in Figure 1a. Use the structuring element in Figure 1b.



0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	1	0	0
0	0	0	0	0	0
0	0	1	0	0	0
0	0	0	0	0	1

(a) A 6 × 6 binary image.



1	●	1

(b) A 1×3 structuring element.

Figure 1: Figure 1: A binary image and a structuring element. The foreground is colored white and given the symbol 1. The background has the symbol 0 and is colored black. The reference pixel in the structuring element (b) is indicated by a black circle.

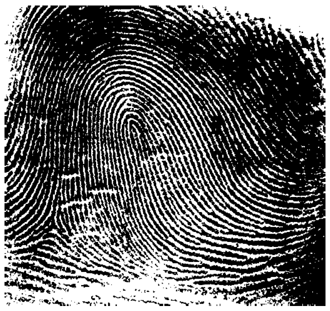
Dilation is defined as $A \oplus B = \{(\hat{B})^z \cap A \neq \emptyset\}$. Since the kernel is symmetrical then $\hat{B} = B$, and we get the following result in matrix form:

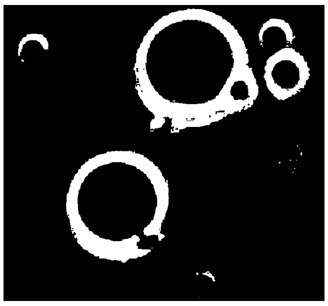
$$A \oplus B = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

2 Programming - Segmentation

(a) Implement a function for Otsu's thresholding algorithm, that returns a single threshold value. Segment the images `thumbprint.png` and `polymercell.png`, and include the results in your report.

After implementing the algorithm we get an optimal threshold of 153 for `thumbprint.png` and of 181 for `polymercell.png`. The final images are:


(a) `thumbprint.png` thresholded with the optimal threshold value


(b) `polymercell.png` thresholded with the optimal threshold value

(b) Implement a function in that segments a grayscale image using the region growing method outlined above. Use a Moore neighborhood (8 connectedness) to expand your set of candidate pixels around each seed point. Apply it on the image `defective-weld.png` and show the result in your report. Use the 4 seed points given in the starter code and use a pixel intensity threshold of 50.

We implemented the region growing function and got the following image for the given seed points and threeshold:

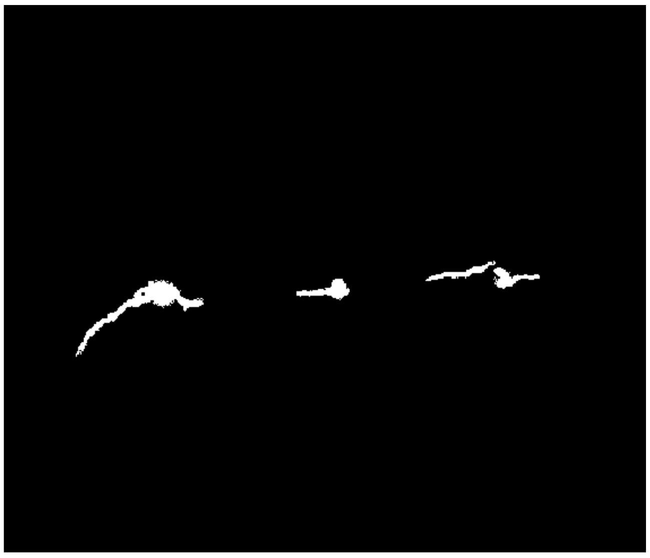


Figure 3: Region growing on`defective-weld.png` with a threeshold of 50

3 Programming - Morphology

(a) Use what you know about erosion, dilation, opening, and closing to remove the noisy elements from the given image.

In order to remove the noise on this image, we will first apply an opening and then a closing to the image, following what we learned on task 1a). Since the specks of noise in this image are quite large, we use a circular structuring element of size $SS \times SS$ and radius $\frac{SS}{2}$.

The advantage of using a circle instead of a big square, is that the edges of the triangle don't appear so jagged after the morphological operations.

Using $SS = 15$ we got a good result, removing all the noise and filling all the holes inside the triangle.

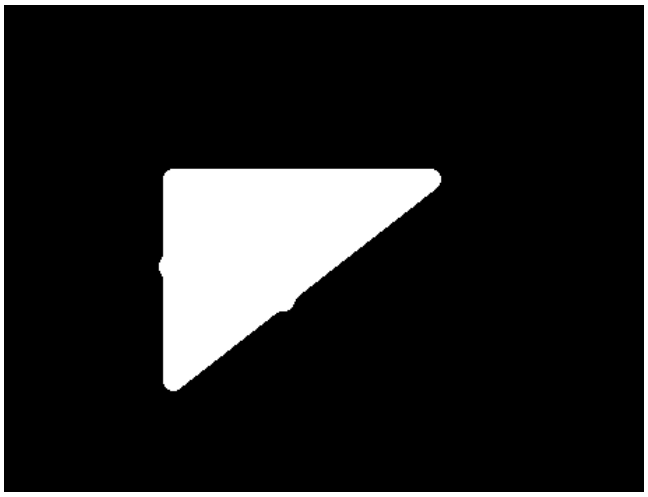


Figure 4: Noise removal on the `noisy.png` image using morphological operations

(b) Implement the distance transform using the erosion method explained above. You can use a 3×3 structuring element of all ones to get chessboard distance.

We implemented the distance transform using multiple passes of erosion on the image and got the following result:

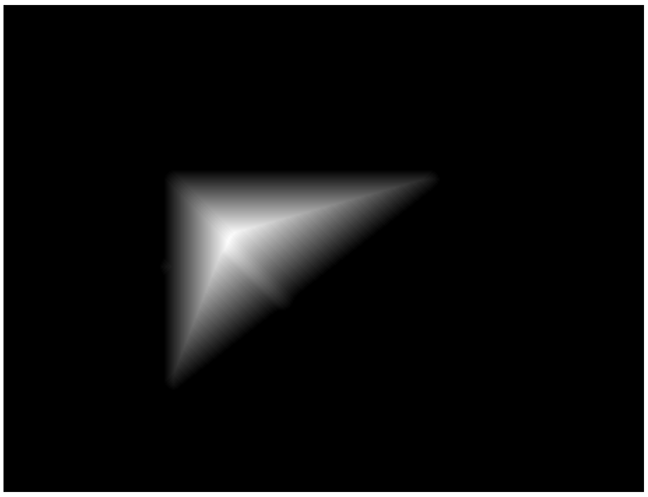


Figure 5: Chessboard distance transform using noise removal and erosion on `noisy.png`

(c) Implement a function that extracts the boundary from a binary image, using the formula $A_{boundary} = A - (A \ominus B)$. You can use a 3×3 structuring element of all ones.

After implementing the above formula (using $A - B = A \cap B^c$) we got the following result:

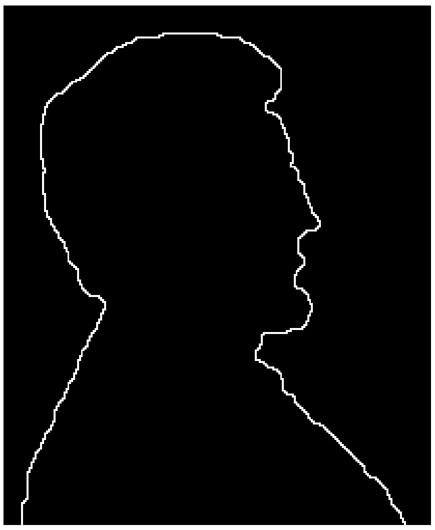


Figure 6: Inner boundary of `lincoln.png` using erosion and set substraction

(d) Implement a function that takes in a binary image and a set of starting points indicating position of holes, and applies the hole filling algorithm. Apply the function on the image `balls-with-reflection.png` and include the resulting image in your report. The position of the holes are given in the starter code. Use 30 iterations ($K = 30$), and a 3×3 structuring element (B) of all 1's.

After implementing the algorithm, we got the following result. All holes were successfully removed from the original image.

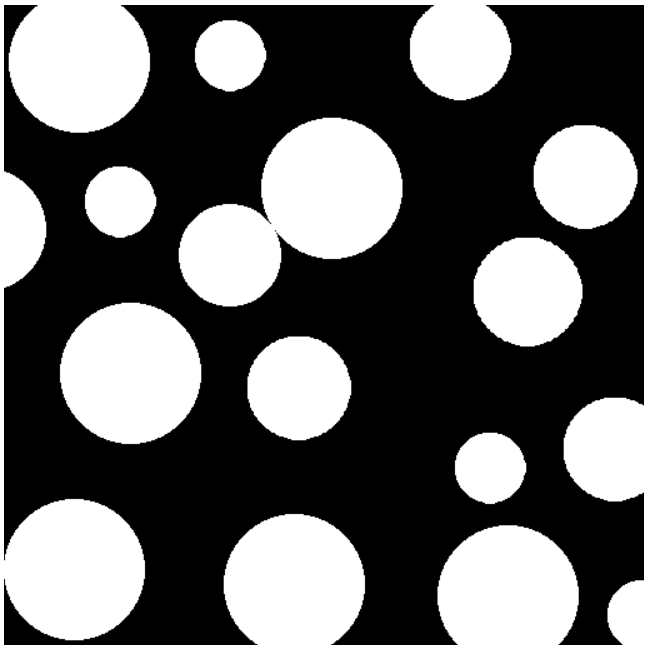


Figure 7: `balls_with_reflections.png` with filled holes using iterated dilations