TDT4195: Visual Computing Fundamentals

Image Processing - Assignment 3

Håkon Hukkelås
hakon.hukkelas@ntnu.no
Department of Computer Science
Norwegian University of Science and Technology

November 8, 2019

- **Delivery deadline: Friday, November 22, 2019, by 23:59.**
- **This project count towards 5% of your final grade.**
- You can work on your own or in groups of up to 2 people.
- Upload your code as a single ZIP file.
- Upload your report as a single PDF file to blackboard.
- You are required to use python3 to finish the programming assignments.
- The delivered code is taken into account with the evaluation. Ensure your code is well documented and as readable as possible.

Introduction

In this assignment, we will explore how we can segment an image into foreground and background by using basic segmentation algorithms, such as thresholding and region region growing. Then, you will explore how we can use binary morphological operations to manipulate the contents of a binary image.

With this assignment, we provide you starter code for the programming tasks. You can download this from:
https://github.com/hukkelas/TDT4195-StarterCode.

To set up your environment, follow the guide in the Github repo:
https://github.com/hukkelas/TDT4195-StarterCode/blob/master/python_setup_instructions.md

Recommended readings

1. Chapter 9 of Digital Image Processing [1]: Specific sections: 9.1, 9.2, 9.3, 9.5
2. Chapter 10 of Digital Image Processing [1]: Specific sections: 10.1, 10.2, 10.3, 10.4

Delivery

Upload your answers as a PDF file to blackboard. For the source code, upload your code as a zip file. In the assignment starter code, we have included a script (`create_submission_zip.py`) to create your delivery zip. **Please use this**, as this will structure the zipfile as we expect. (Run this from the same folder as all the python files). Note, do not include the PDF in the zip for the code!

To use the script, simply run: `python3 create_submission_zip.py`

Task 1: Theory [1 point]

(a) **[0.2 points]** Define *opening* and *closing* in terms of *erosion* and *dilation*. What happens when open and closing are applied multiple times on the same image?

(b) **[0.1 points]** Smoothing of an image is often done before performing edge detection. What is the purpose of smoothing the image before edge detection?

(c) **[0.2 points]** The Canny edge detector uses a method called *hysteresis thresholding*. Shortly explain how hysteresis thresholding work.

(d) **[0.2 points]** Why do we use hysteresis thresholding instead of a single treshold?

(e) **[0.3 points]** Determine the dilation $A \oplus B$ of the binary image in Figure 1a. Use the structuring element in Figure 1b.

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

(a) A 6×6 binary image.

1	•	1

(b) A 1×3 structuring element

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.

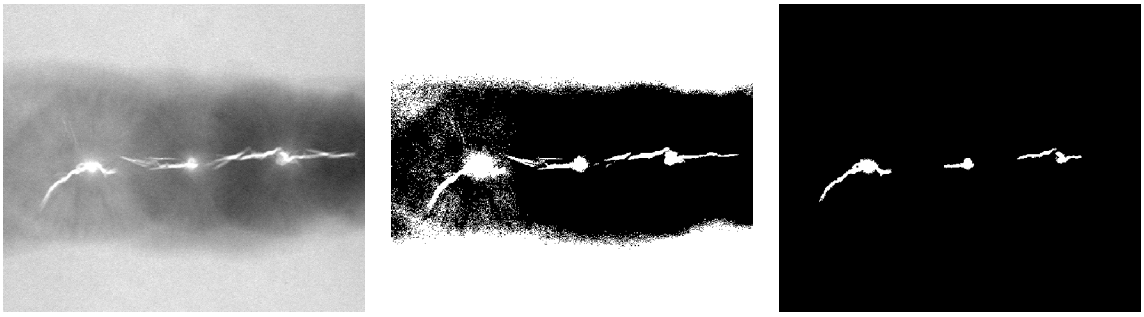
Programming

Task 2: Segmentation [2 points]

Segmentation by thresholding on pixel intensity is a intuitive, computational efficient, and simple way of implementing image segmentation. Selecting a good value for thresholding is a cumbersome process that is best left up to a threshold detection algorithm. One automatic algorithm for selecting a good threshold in a grayscale image is Otsu's method. The algorithm finds a single threshold value that separates pixels into two classes, foreground and background. The basic idea is that a threshold giving the best separation between classes in terms of their intensity values would be the optimum threshold. In the assignment files we have included the file `otsu-thresholding.pdf`, scanned from the curriculum book [1], which describes the algorithm in detail.

(a) **[1.0 points]** Implement a function in `task2a.py` that implements Otsu's algorithm for thresholding, and returns a single threshold value.

Segment the images `thumbprint.png` and `polymercell.png`, and include the results in your report.



(a) X-ray image of a defective weld

(b) Segmented using *Otsu's thresholding algorithm.*

(c) Segmented using *region growing* with four manually selected seed points.

Figure 2: Two different ways to segment the image in (a). Image source: [1]

Region growing is a region-based segmentation algorithm that uses a set of seed points and a homogeneity criteria $H(R_i)$ to perform segmentation. For each seed point, a region is grown by inspecting neighboring pixels and evaluating whether or not to include them in region R_i using $H(R_i)$. The neighboring pixels that are currently being evaluated are typically called *candidate pixels*. The growing of a region stops when there are no more candidate pixels to inspect. The simplest homogeneity criteria is a threshold, where the threshold defines the maximum absolute difference in intensity between the seed point and the current candidate pixel.

(b) **[1.0 points]** Implement a function in `task2b.py` 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.

Task 3: Morphology [2 points]

For the following tasks, you are allowed to use already implemented functions for opening, closing, erosion, and dilation. Skimage has several useful functions for this:

- `skimage.morphology.binary_dilation`
- `skimage.morphology.binary_erosion`
- `skimage.morphology.binary_closing`
- `skimage.morphology.binary_opening`

(a) A noisy binary image

(b) An almost noise free version of (a) after morphological operations.

(c) A distance transform of the binary image in (b) using chessboard distance.

Figure 3: Morphology operations on a binary image (a). Image source: [1]

(a) **[0.6 points]** Use what you know about erosion, dilation, opening, and closing to remove the noisy elements from the image in Figure 3. Implement this in the file `task3a.py`. Your result should look something like the one in Figure 3b.

Include the resulting image in your report and shortly explain how you removed the noise.

The distance transform is an operation typically applied on a binary image and creates a grayscale image where each foreground pixel shows the distance to the closest boundary pixel. An example of this using chessboard distance can be seen in Figure 3c. The chessboard distance is the minimum number of moves a king has to perform to move from one square (or pixel in our case) to another square in the game of chess.

One inefficient way of calculating the distance transform is to use erosion. Intuitively, by using erosion the distance transform for a pixel is simply the number of erosion operations it took to remove it from the foreground of the original image.

(b) **[0.6 points]** Implement the distance transform using the erosion method explained above, in the file `task3b.py`. You can use a 3×3 structuring element of all ones to get chessboard distance.

Test the function on the noise free binary image you got from task (a) and show the result in your report.

Mathematical operations can be used for extracting boundary information from images. The operation for extracting the inner boundary extraction can be seen in Equation 1, where $\ominus$ is erosion.

$$A_{boundary} = A - (A \ominus B) \tag{1}$$

(c) **[0.3 points]** Implement a function that extracts the boundary from a binary image, as defined in Equation 1, in the file `task3c.py`. You can use a 3×3 structuring element of all ones.

Show the boundary on the image `lincoln.png` and include the result in your report. What operation could we use to extract the outer boundary of the binary image?

Hole filling is a method to fill in holes in a segmentation, and is very useful for post-processing imperfect segmentations. Algorithm 1 outlines a basic algorithm that takes a binary image and fills all known holes indicated by a set of starting points.

Algorithm 1 A algorithm to fill holes in a binary image I. $\oplus$ is the dilation operation, I^c is the complement of I, and $\cup$ is the union.

Input: Image I, number of iterations K, starting_points S, and structuring element B.
Form an array, X_0, of 0's (the same size as the Image I)
for row, column in S **do**
    $X_0[row, column] \leftarrow 1$
for $k \leftarrow 1$ to K **do**
    $X_k \leftarrow (X_{k-1} \oplus B) \cap I^c$
return $X_k \cup I$

(d) **[0.5 points]** 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 outlined in Algorithm 1. Implement this in the file `task3d.py`.

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.

References

[1] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing.* Pearson, 2018.