PRACTICAL IMAGE AND VIDEO PROCESSING USING MATLAB®

OGE MARQUES
Florida Atlantic University

IEEE PRESS

A JOHN WILEY & SONS, INC., PUBLICATION

Note: the image to be used in the part of the adaptive threshold will be the figure you find in Moodle called "gradient_without_text.png". Once imported into Matlab, you need to either convert to a grayscale representation, or use only one of the RGB channels to perform the experimentations.

This is, instead of

i_adaptTh=blkproc(i, [10,10], @adapt_thresh);

it should be

i_adaptTh=blkproc(i(:,:,1), [10,10], @adapt_thresh);

Note2: the values of the std of this picture are different from the ones obtained in the book, so you will need to adjust the value that controls the use or not of adaptive thresholding

15.5 TUTORIAL 15.1: IMAGE THRESHOLDING

Goal

The goal of this tutorial is to learn to perform image thresholding using MATLAB and the IPT.

Objectives

- Learn how to visually select a threshold value using a heuristic approach.
- Explore the `graythresh` function for automatic threshold value selection.
- Learn how to implement adaptive thresholding.

What You Will Need

- `gradient_with_text.tif`

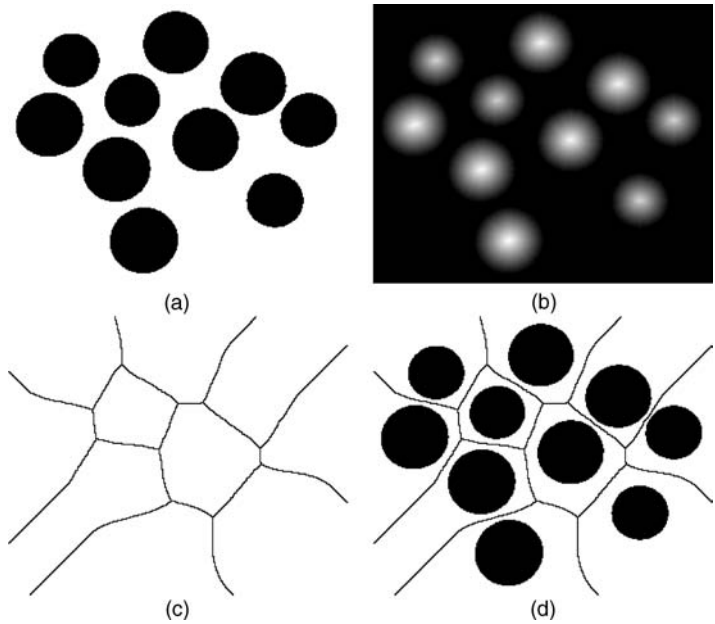

FIGURE 15.10 Segmentation using the morphological watershed transform: (a) complement of the image shown in Figure 15.3; (b) distance transform; (c) watershed ridge lines; (d) result of segmentation.

Procedure

Global Thresholding

The first method of thresholding that we will explore involves visually analyzing the histogram of an image to determine the appropriate value of `T` (the threshold value).

1. Load and display the test image.

```
I = imread('coins.png');
figure, imshow(I), title('Original Image');
```

2. Display a histogram plot of the `coins` image to determine what threshold level to use.

```
figure, imhist(I), title('Histogram of Image');
```

Question 1 Which peak of the histogram represents the background pixels and which peak represents the pixels associated with the coins?

The histogram of the image suggests a bimodal distribution of grayscale values. This means that the objects in the image are clearly separated from the background.

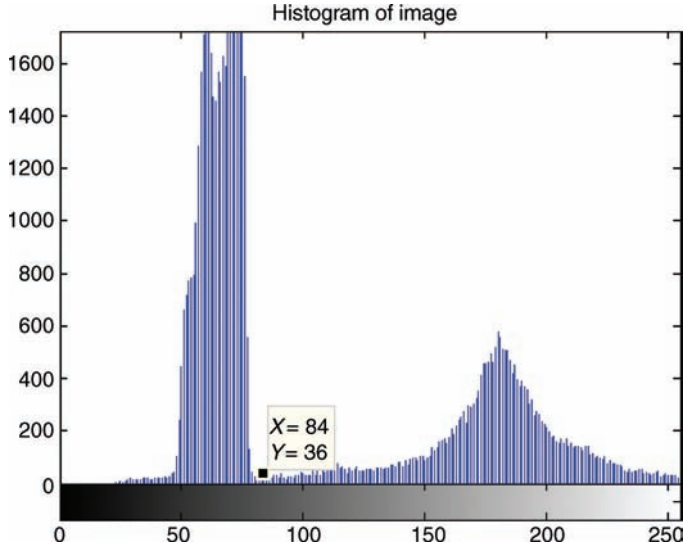

FIGURE 15.11 Histogram plot with data cursor selection.

We can inspect the `X` and `Y` values of the histogram plot by clicking the "inspect" icon and then selecting a particular bar on the graph.

3. Inspect the histogram near the right of the background pixels by activating the data cursor. To do so, click on the "inspect" icon.

Figure 15.11 illustrates what the selection should look like. The data cursor tool suggests that values between 80 and 85 could possibly be used as a threshold, since they fall immediately to the right of the leftmost peak in the histogram. Let us see what happens if we use a threshold value of 85.

4. Set the threshold value to 85 and generate the new image.

```
T = 85; I_thresh = im2bw(I,( T / 255));
figure, imshow(I_thresh), title('Threshold Image (heuristic)');
```

Question 2 What is the purpose of the `im2bw` function?

Question 3 Why do we divide the threshold value by 255 in the `im2bw` function call?

You may have noticed that several pixels—some white pixels in the background and a few black pixels where coins are located—do not belong in the resulting image. This small amount of noise can be cleaned up using the noise removal techniques discussed in chapter 12.

Question 4 Write one or more lines of MATLAB code to remove the noise pixels in the thresholded image.

The thresholding process we just explored is known as the *heuristic* approach. Although it did work, it cannot be extended to automated processes. Imagine taking on the job of thresholding a thousand images using the heuristic approach! MATLAB's IPT function `graythresh` uses Otsu's method [Ots79] for automatically finding the best threshold value.

5. Use the `graythresh` function to generate the threshold value automatically.

```
T2 = graythresh(I);
I_thresh2 = im2bw(I,T2);
figure, imshow(I_thresh2), title('Threshold Image (graythresh)');
```

Question 5 How did the `graythresh` function compare with the heuristic approach?

Adaptive Thresholding

Bimodal images are fairly easy to separate using basic thresholding techniques discussed thus far. Some images, however, are not as well behaved and require a more advanced thresholding technique such as *adaptive thresholding*. Take, for example, one of the images we used back in Tutorial 6.1: a scanned text document with a nonuniform gradient background.

6. Close all open figures and clear all workspace variables.
7. Load the `gradient_with_text` image and prepare a subplot.

```
I = imread('gradient_with_text.tif');
figure, imshow(I), title('Original Image');
```

Let us see what happens when we attempt to threshold this image using the techniques we have learned so far.

8. Globally threshold the image.

```
I_gthresh = im2bw(I,graythresh(I));
figure, imshow(I_gthresh), title('Global Thresholding');
figure, imhist(I), title('Histogram of Original');
```

As you may have noticed, we cannot pick one particular value to set as the threshold value because the image is clearly not bimodal. Adaptive thresholding may help us in this instance. To properly implement adaptive thresholding, we must use the `blkproc` function to perform an operation on small blocks of pixels one at a time.

In order to use the function, we must specify what is to be done on each block of pixels. This can be specified within a function that we will create manually. Let us first set up this function.

9. Close all open figures.
10. Start a new M-File in the MATLAB Editor.
11. Define the function as well as its input and output parameters in the first line.

```
function y = adapt_thresh(x)
```

This function will be used to define each new block of pixels in our image. Basically all we want to do is perform thresholding on each block individually, so the code to do so will be similar to the code we previously used for thresholding.

12. Add this line of code under the function definition.

```
y = im2bw(x,graythresh(x));
```

When the function is called, it will be passed a small portion of the image, and will be stored in the variable `x`. We define our output variable `y` as a black and white image calculated by thresholding the input.

13. Save your function as `adapt_thresh.m` in the current directory.

We can now perform the operation using the `blkproc` function. We will adaptively threshold the image, 10×10 pixel blocks at a time.

14. Perform adaptive thresholding by entering the following command in the command window. Note that it may take a moment to perform the calculation, so be patient.

```
I_thresh = blkproc(I,[10 10],@adapt_thresh);
```

15. Display the original and new image.

```
figure
subplot(1,2,1), imshow(I), title('Original Image');
subplot(1,2,2), imshow(I_thresh), title('Adaptive Thresholding');
```

The output is not quite what we expected. If you look closely, however, the operation was successful near the text, but everywhere else it was a disaster. This suggests that we need to add an extra step to our function to compensate for this unwanted effect. Over the next few steps, let us examine the standard deviation of the original image where there *is* text, and where there is *not*.

16. Calculate the standard deviation of two 10×10 blocks of pixels; one where there is text and another where there is not.

```
std_without_text = std2(I(1:10, 1:10))
std_with_text = std2(I(100:110, 100:110))
```

Question 6 What is the difference between the standard deviation of the two blocks of pixels? Explain.

Since there is such a difference between a block with and without text, we can use this information to improve our function. Let us replace the one line of code we previously wrote with the new code that will include an `if` statement: if the standard deviation of the block of pixels is low, then simply label it as background; otherwise, perform thresholding on it. This change should cause the function to only perform thresholding where text exists. Everything else will be labeled as background.

17. Replace the last line of our function with the following code. Save the function after the alteration.

```
if std2(x) < 1
    y = ones(size(x,1),size(x,2));
else
    y = im2bw(x,graythresh(x));
end
```

Question 7 How does our function label a block of pixels as background?

18. Now rerun the block process (in the command window) to see the result.

```
I_thresh2 = blkproc(I,[10 10],@adapt_thresh);
figure, subplot(1,2,1), imshow(I), title('Original Image');
subplot(1,2,2), imshow(I_thresh2), title('Adaptive Thresholding');
```

Question 8 How does the output of the new function compare with the old?

Question 9 What is the main limitation of the adaptive thresholding function developed in this tutorial?

WHAT HAVE WE LEARNED?

- Image segmentation is the process of grouping image pixels into meaningful, usually connected, regions. It is an important (and often required) step in many image processing solutions. It establishes the transition between treating the image as a whole to processing individual relevant regions.

- Image segmentation is a hard image processing problem: the quality of the results will depend on the algorithm, careful selection of algorithm's parameters, and the input image.
- Thresholding is an image processing technique by which an input (grayscale) image is requantized to two gray levels, that is, converted to a binary image. Each pixel in the original image is compared with a threshold; the result of such comparison will determine whether the pixel will be converted to black or white. The simplest thresholding algorithm (*global thresholding*, `im2bw` in MATLAB) employs one value for the entire image.
- Image segmentation techniques can be classified in three main groups: *intensity-based methods* (e.g., thresholding), *region-based methods* (e.g., region growing and split and merge), and *other methods* (e.g., segmentation based on texture, edges, and motion).

LEARN MORE ABOUT IT

- Entire books have been written on the topic of image segmentation, for example, [Wan08] and [Zha06a].
- Many surveys on image segmentation have been published during the past 30 years, among them (in reverse chronological order): [Zha06b], [FMR+02], [CJSW01], [PP93], [HS85], and [FM81].
- Chapters 6 and 7 of [SHB08] provide an extensive and very readable discussion of image segmentation algorithms. Several of these algorithms have been implemented in MATLAB [SKH08].
- Chapter 4 of [Dav04] is entirely devoted to thresholding techniques.
- Section 6.1.2 of [SHB08] discusses optimal thresholding techniques.
- The concept of thresholding can be extended to cases where the original image is to be partitioned in more than two regions, in what is known as *multiple thresholding*. Refer to Section 10.3.6 of [GW08] for a discussion on this topic.
- Chapters 4.7–4.9 of [Bov00a] discuss statistical, texture-based, and motion-based segmentation strategies, respectively.
- The discussion on watershed segmentation is expanded to include the use of gradients and markers in Section 10.5 of [GWE04].
- Comparing and evaluating different segmentation approaches can be a challenging task, for which no universally accepted benchmarks exist. This issue is discussed in [Zha96], [Zha01], and, more recently, [UPH05], expanded in [UPH07].

ON THE WEB

- Color-based segmentation using K-means clustering (IPT image segmentation demo)
  http://tinyurl.com/matlab-k-means