5. Image segmentation and region characterisation

Abstract
Segmenting an image into parts can be a step towards scene understanding. If we identify different regions, these can be couned, characterised, or classified. This lab covers how to perform these operations at an introductory level using simple images captured under relatively controlled conditions.

Keywords
Thresholding by histogram analysis • Otsu algorithm • Connected components labelling • Region characterisation

Contents

1. Binarisation

The possibly easiest segmentation approach to segment an image is binarisation, i.e. assigning each pixel one out of two possible values, typically objects (or foreground) and background. Otsu method is a classical algorithm of *thresholding*, i.e., image binarisation by using a threshold of the gray level. In Otsu's method, this threshold is found automatically. ▷ E_7 Let's see how to use and assess this method. Consider the image `monedas.pgm`. Display its histogram and check it is clearly bimodal. From that observation, choose manually one threshold (to that end, use the variable `myThreshold`) and compare the segmentation using this threshold with Otsu's one. If the goal is to extract the objects from the background, which of the two thresholds do you think is more adequate for *this* image?

Then, at programming level and using our code:

- Which function do we use to apply Otsu method?
- To which Python module it belongs to?
- How do we binarise an image given a threshold?
- Which values does the binarised image have?

2. Region labelling

▷ E_1 If we display a binarised image, we can perceive different "regions", and the traditional way to identify them automatically is by a connected-components labelling algorithm. Which function and Python module did we use for that purpose? What happens, and why, if in this function call we switch `0` to `1` as the argument for parameter `background`?

If you pay attention to the identified regions, you will notice the result is somehow undesirably noisy. You can see that the regions have lots of little holes, and there is a very small region (what size is it?).[1] Simple cases like this one can be fixed by operations that belong to an area known as mathematical morphology, but which is not covered in our course.

3. Region characterisation

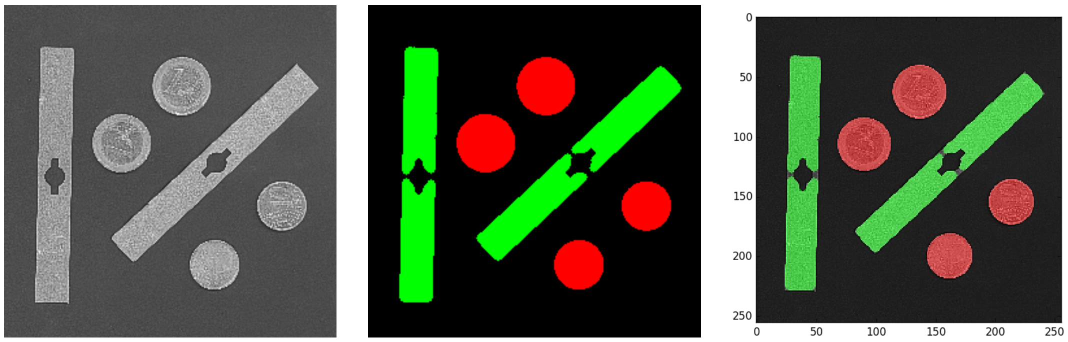
Finally, once we know the set of pixels corresponding to each region, we can compute values to characterise it: area, perimeter, *bounding box*, excentricity, etc. Those features can be the basis for other computations and/or taking some decisions. For instance, in our coin images we can guess which regions correspond to coins and which others are something else. ▷ E_2 In one further step, we can distinguish different coin types and then straighforwardly proceed to money counting. ▷ E_3 ▷ E_4

4. Exercises

1. Try to find a (good) segmentation of the coin images. For this and the following exercises, you can use `monedas`, but also `monedas1` y `monedas2`. However, the later two images are more challenging, mostly without using mathematical morphology. So, do not worry if you find hard to get good results; just use this experience to better appreciate how problems that are very simple for humans can be hard for computers, and how additional computer vision and image processing tools can help us in some situations. Notice that, at this point, we cannot yet distinguish coins from other objects ; we only know there are different regions and their set of pixels.

   Side note: Notice that in more realistic situations the problem we are considering is more challenging because our algorithms should work for many more images, not only one or a few, and the images can be even unknown or be unavailable at the

[1] Is this suboptimal result a consequence of the connected-component labelling algorithm?



(a) (b) (c)

Figura 1. Result of clasifying regions into the classes "coin" and "no-coin" (Exercises 2): (a) the original image (`monedas`); (b) coin class in red an no-coin class in green; and (c) overlay of the classification classification result on the original image

moment of algorithm development. Fortunately, in those cases we can include *machine learning* (ML) techniques to make our systems more robust and general. As you know, nowadays computer vision is intimately related with ML, so much so that it is less and less common to develop computer vision systems which do not rely on ML. If you are curious on research topics in computer vision, one good source are the proceedings of top international conferences such as CVPR, ICCV, ECCV, for instance at The Computer Vision Foundation – Open Access.

2. Use the perimeter and area of the found regions (or any other feature or feature combination that you consider useful or wan to experiment with) to identify those that have an approximate circular shape (which, in our case, corresponds to coins). Represent the result using one color for the circular region and another one for those which are not, similar to Fig. 1 (Note: we used mathematical morphology to get this result; it is perfectly fine if your result does not look like this one. Since `labelConnectedComponents()` returns both the unprocessed and processed binary images and the corresponding regions, you can experiment with both and see if you can distinguish which are the coin regions in both cases.) For each image, count programmatically and show (either on the standard output or drawn on the image) the number of coins.

3. Using the output to Exercise 2 as a starting point, find which regions correspond to each coin type (1 euro, 10 cents). The following information can be useful:

   - the coin images have been taken using 50 pixels per inch (1 inch = 2,54 cm).
   - the diameters of 1-euro and 10-cents coins are 23 mm 19,5 mm, respectively.

   Optionally, as in that previous exercise, represent visually the two types of coins. Using transparency, overlay the color image representing each coin and the original image (similarly to Fig. 1c), to facilitate the manual checking of whether the classification of each region is correct, and how much the actual and estimated regions match.

4. Lastly, compute automatically the total amount of money and, optionally, display it as text in the figure (as part of the title and/or as text drawn on the image). Compare the actual and estimated money counts. Identify the strong and weak points of the different parts of your solution. You can think of possible ways for improving the solution, but do not implement them.

5. (Optional, somehow advanced or costly) Develop your own implementation of (a) Otsu's method, and/or (b) the connected-components labelling algorithm.

6. (Optional) Try the multi-Otsu thresholding for the case of images with more than two object classes. You can test on coin images or your own set of images.

7. (Optional, open) The are other segmentation methods besides Otsu's that rely also on Thresholding. Choose and study a couple of them. Perform some tests and compare those algorithms and Otsu's on different types of images (p.e. coins, texts, traffic signs). Analyse the behaviour of the different algorithms under different conditions (image characteristics, noisy type, etc.)

8. (Optional, advanced) Instead of segmenting an image from individual pixels into *final* regions, one approach is to oversegment an image into the so-called "superpixels". Since superpixels are something between pixels and regions, they can be used as an intermediate output that can benefit subsequent problems. For instance, the efficiency and the performance of segmentation algorithms can be improved. One of such superpixel methods is SLIC (simple linear iterative clustering). Learn about it and then try it on our coin images or other images. You can use skimage's slic().