ET4283: Advanced Digital Image Processing
Final Project on "Image Segmentation"

Ali Alessio Salman, 4505775
A.A.Salman@student.tudelft.nl

María Silos Viu,4701888
M.SilosViu@student.tudelft.nl

January 7, 2016

1 Introduction

Image segmentation aims to partition an image into multiple connected segments with the purpose of extracting meaningful information. There are two main broad families of segmentation approaches, namely edge-based and region-based. Among the latter, several image segmentation techniques are widely known to employ approaches in which normally the main contours of the image are drawn first so that subsequent segmented regions are then subtracted from those contours.

This project aims at two main goals. Firstly, the most extensively applied region-based approaches are briefly described in order to get a general idea about the state-of-art image segmentation methods, among which watershed segmentation is explained. Secondly, two main basic implementations are presented. The first one is focused on the oriented gradients of histograms method that serves as an efficient contour detector. The second one employs the output of this contour detector to undergo a morphological watershed segmentation. At the end of this document, the influence of the main parameters and the obtained results are discussed.

2 Region-Based Approaches for Image Segmentation

Three main region-based approaches are most extensively applied to perform image segmentation.

*Graph cut theory***.** This approach is based on the idea of creating a graph from an image in which each pixel represents a single node of the graph. Graph cut theory takes into account the similarities between nodes and search for an optimal cut that divides the image into two regions ("background" and "foreground") while minimizing as much as possible the cut segment. However, the cuts of these graphs are not always in accordance to the user needs, as the optimal cut is usually the one dividing a very small region from the rest of the image. In this terms, *Normalized Cuts* [6] criterion is rather useful as it allows for the employment of a "cost function" that penalize too small cuts and creates subsets that are reasonably balanced, i.e. each region is large enough.

Mean Shift [3]. This mode-seeking algorithm consists of a clustering-based segmentation that looks for the local maxima of a density function of a set of pixels. The method firstly locates the center of a region of interest (ROI) in a random position, then the mean density of this ROI is computed so that the center is subsequently shifted towards this mean position. The process is iteratively performed until the ROI's mean density converges to a specific position. After the final mean is located, groups of pixels are clustered together so that all pixels within certain cluster lay down near the same peak value (referred to as a basin). The main disadvantages of this method are that the results highly depend on the ROI size and that it is computationally expensive.

Watershed Transform [7]. This method belongs to the broad family of the region growing approach and relies on viewing a 2D gray scale image as a topographic surface in which the grey level intensity is represented in the third dimension as the height of the relief. In this way, several sources of water, referred to as *seeds*, are located in the local minima of the image in order to flood the relief at a constant rate forming the so-called *catchment basins*. When the water level increases, the flooded regions grow in size until eventually, two of these regions merge. At this point, the algorithm creates a one-pixel contour that separates these flooded regions. The process is iteratively performed until all the regions of the image are flooded and separated by a dam between each other.

3 Oriented Gradient of Histograms Implementation

In this section, we present a contour detector that quantifies the presence of boundary from local image cues. This is performed by computing the oriented gradient of local histograms as implemented in [1]. Our implementation uses only the brightness channel. Given a pixel *A* placed in (x,y), G(A) refers to the gradient signal between the histograms of two half-discs with center in A at an angle θ. The magnitude of the gradient G(A) is defined as the distance χ^2 between the histograms g and h:

$$\chi^2(g,h) = \frac{1}{2}\sum_i \frac{\big(g(i) - h(i)\big)^2}{g(i) + h(i)} \tag{1}$$

Then, the orientation of the two half-discs is changed for different values of θ and the resulting value of the oriented gradient of pixel A will correspond to the maximum value among the different gradients. In our case, θ has values of 0º,45º,90º and 125º.

Initially, this algorithm is computationally expensive, as it reaches the complexity of $O(Nr^2 + NB)$, being N the number of pixels, r the radius of the circle around A, and B the number of histogram bins. There are three main key points that make this algorithm computationally faster without losing contour quality, as demonstrated in [1]. In the first place, when computing the oriented gradient at angles 45º and 125º, the image itself gets rotated once instead of the rectangles for each pixel. Secondly, the previously

mentioned half-discs are replaced by rectangles forming a square with center in A. Finally, the histograms are processed bin by bin. In this way, having I^b (x,y) as the image of pixels falling inside bin b, we compute the integral image J^b, which stores in each of its components the cumulative sum of pixels within an area, in our case, within either one of our two rectangles. Then, the resulting value of the histogram in b is equal to:

$$h(b) = J^b(P) + J^b(S) - J^b(Q) - J^b(R) \quad (2)$$

Being, P, S, Q, R the upper-left, upper-right, lower-left, and lower-right corners of a rectangle, respectively. Next, the rest of histogram bins are processed and, finally, equation (1) is applied to obtain the total value of the oriented gradient in pixel A.

At this point, the time required for the computation has been reduced to O(NB), meaning that the size of the rectangles has no effect on the computation cost, giving the same results as the original approximation of the algorithm, as shown in [1].

An example of a resulting image of this process is represented in Fig. 4.

4 Morphological Watershed Implementation

As previously mentioned, watershed segmentation focuses on constructing a topographic surface flooded from seeds located at the regional minima.

Our watershed implementation is based on the morphological watershed algorithm described in [5]. Following these steps, we set the seeds as the minimum value of the oriented gradient of histograms and flood it from there. The idea is to compare the points belonging to certain level *n*, label them by regions using the Matlab function `bwlabel` to be stored in matrix `Q`, and compare them to the already flooded points in levels < *n*, labeling them again by regions and storing these components in matrix `CB`. Next, the intersection between `Q` and `CB`, named `newq`, is analyzed in order to see the number of contained labels, i.e. number of flooded regions that lie in the intersection, which is stored in variable `nqint`.

In this way, the intersection can fall into either one of these three scenarios: (a) it has no flooded regions inside, which means a new minimum has been found and `q` is added to `CB` with a new label, to form `CB+1`; (b) there is one flooded region within the intersection, which means `q` lies within the catchment basin of a regional miminum, again we grow `CB` with only the new element in `Q`; or (c) the intersection has more than one region inside, which means a ridge has been encountered and two regions will merge in th next stage. In this last scenario, the regions are grown with a structural element until a dam is created in between them.

In order to avoid “noisy seeds”, we set a threshold of gray level above which the program will not add new seeds in scenario (a).

The core part of our watershed implementation is shown at page 4.

5 Discussion

Regarding the implementation of the gradient of oriented histograms, several parameters played a crucial role in the final result of the contour detector.

In the first place, the asset of leveraging on integral images becomes evident if we increase the *(i) size* of the neighborhood. With the former, summation of pixels can be done in constant time, regardless of the neighborhood size. The bigger this size is, the more pronounced is the difference in performance, up to 4x times faster. However, increasing the size of the neighborhood leads to inaccuracy of the computation of each pixel gradient as a result of the wider area that we are taking into account for the calculations. Thus, we found out that a neighborhood size of 5 is a very good trade-off.

Hence, the *(ii) scale* of the image is also an important factor since the pixel resolution must be high enough to use 5 pixels neighborhoods without blurring the contours too much, see Fig. 3.

The efficient algorithm perfectly takes advantage of the integral images processing each histogram bin separately. Consequently, a higher *(iii) number of bins* induces a slower computation. For our purposes and dataset, a number between 8 and 24 has worked very well in both performance and results. Another essential technique to successfully use the integral computation is to *(iv) pre-rotate* the images according to the gradient orientation we want to compute, in order for the rectangles to be axis-aligned.

Our implementation of the Watershed Transform works on the oriented gradient of histograms, thus the *(i) accuracy of the gradient* is the starting point to properly find the seeds. The noise in the gradient can lead to the well-known problem of over-segmentation. Therefore noise reduction has been performed through a stage of *(ii) median filtering* suitable of smoothing without blurring sharp edges, crucial for the transform. Noticeable is also the noise at higher intensity values, i.e. closer to the ridges. The latter can cause the algorithm to erroneously find new small catchment basins, driving again to an over-segmentation. Establishing and tuning a proper *(iii) threshold* after which no more catchment basins is added is thence an important step towards a clearer segmentation. Performance improvements are possible looping only on actual intensity values. Our implementation provided mostly good segmentations, not perfect though. This is mainly due to the fact that our gradient wasn't always able to provide closed contours.

Overall, we consider the achieved results to be satisfying, proving both the high efficiency of the gradient of oriented histograms and the reliability of the watershed transform. For future work, it could be interesting to improve the algorithm with what Arbelaez et al[1] call the *Oriented* watershed transform.

Furthermore, while this work has focused on data-driven segmentation, the watershed transform can be successfully used for knowledge-based techniques aiming to even better results.

Watershed algorithm, matlab

```
threshold = 140;
grad= get_gradient(original_image) ;
%   Initializes the catchment basins
CB = bwlabel(grad < imin+1);    % C[min+1] = T[min+1]
dam = zeros(size(CB)); % Initializing dam vector
for n=imin+1:imax+1;   %looping on the stages
    Q = bwlabel(grad <= n); %Labels the new basements
    NQ = max(max(Q));
    for q=1:NQ; %   For each element q in Q
        newq  = (Q==q) & (CB > 0);
        nqint = get_number_list(CB(newq));% list of regions
        if(length(nqint)==0), % scenario a
```

```
            if(min(min(grad(Q==q))) < threshold),
                CBnplus1 = CB + (max(max(CB))+1)*(Q==q);
            end;
        elseif(length(nqint)==1), % scenario b
            CBnplus1 = CB + nqint*((Q==q) & (CB==0));
        else % scenario c
            [CBnplus1, tdam] = grow_regions_inside_Q(CB, Q==q);
            dam = dam | tdam; % we add the new dam drawn above
        end;
        CB = CBnplus1;
    end;
end;
```

The resulting regions from the watershed implementation can be seen in Fig 1. and Fig. 2.

Figures

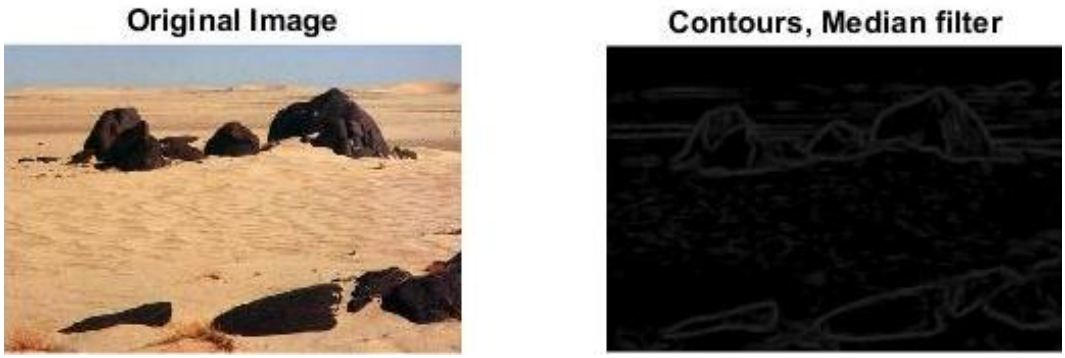

Fig. 1. ***Left:*** Original image. ***Right***: Gradient of the original image.

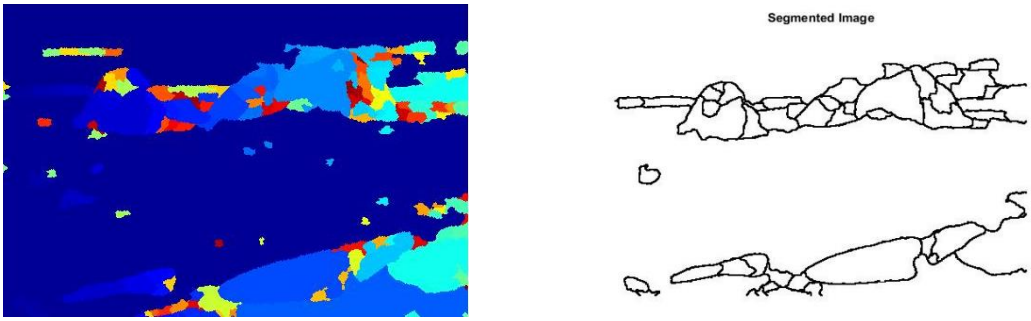

Fig. 2. ***Left:*** Colored regions from watershed algorithm. ***Right***: Segmented regions.

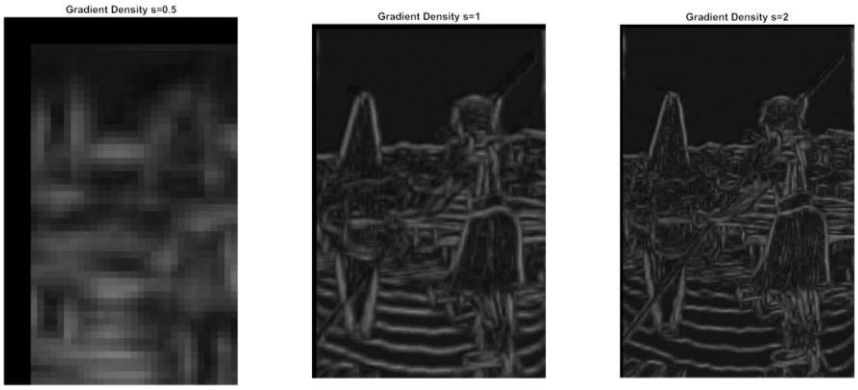

Fig. 3. Influence of scale in gradients' magnitudes. Big scales images give rise to fine contours in contrast with the coarse contours acquired in the smaller scale image.

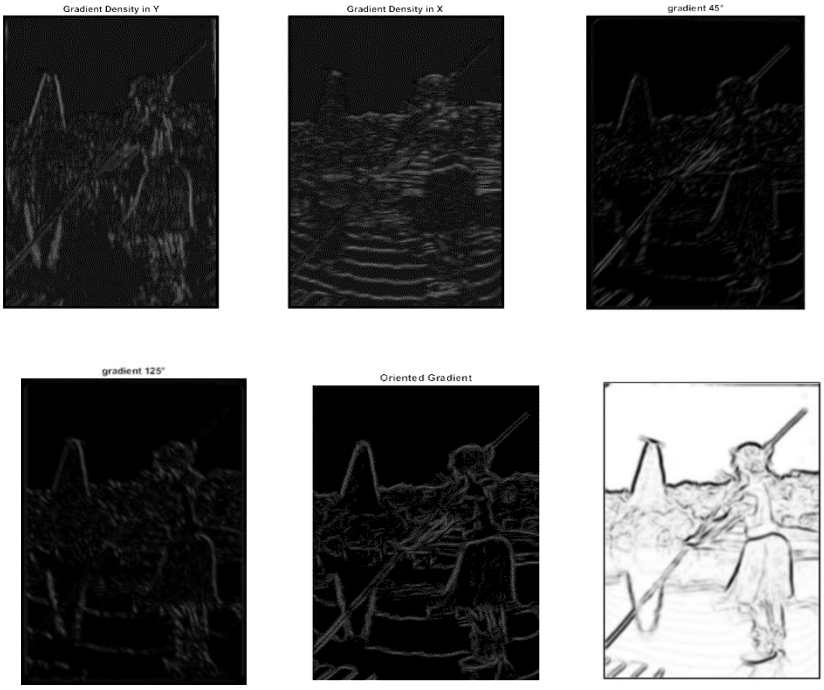

Fig. 4. Influence of θ in gradients' magnitudes. Using one single orientation (θ=0º,90°,45°,125°) is not enough to acquire contour quality, but the combination of all give rise to an efficient contour detector. The last image is the result of the contour detection by Arbelaez et al [1].

References

1. Arbelaez, P., Maire, M., Fowlkes, C., & Malik, J. (2011). Contour detection and hierarchical image segmentation. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 33(5), 898-916.
2. Felzenszwalb, P. F., & Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. International Journal of Computer Vision, 59(2), 167-181.
3. Comaniciu, D., & Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 24(5), 603-619.
4. Gallier, J. (2015). Spectral Theory of Unsigned and Signed Graphs Applications to Graph Clustering: a Survey.
5. R.C. Gonzalez, R.E. Woods (2002). Digital Image Processing, 2nd edition, Prentice Hall.
6. T. Cour, F. Benezit, and J. Shi, "Spectral segmentation with multiscale graph decomposition." CVPR, 200
7. S. Beucher and F. Meyer, Mathematical Morphology in Image Processing. Marcel Dekker, 1992, ch. 12