Image Segmentation in Naval Ship Images

Oliver Rusch, Christian Ruwwe and Udo Zölzer
Department of Signal Processing and Communications
Helmut-Schmidt-University
University of the Federal Armed Forces Hamburg, Germany
or@inf-ing.com, {christian.ruwwe|udo.zoelzer}@hsu-hh.de

1 Introduction

Extracting features from naval ship images is a difficult task to solve, because the background contains rough-textured regions - water. Classical edge and corner operators tend to fail due to high contrast in these parts. First appropriate preprocessing steps are necessary to separate the object from the background. Then subsequent feature extraction techniques can be applied to the object itself.

Several segmentation algorithms have been proposed in the past [1], [2]. One of the latest is the *GrabCut* algorithm by Rother, Kolmogorov and Blake [3], [4], based on Boykov's *GraphCut* [5]. We present an overview of this approach and some results using it with naval ship images. A further enhancement justified for this type of images has been integrated into the algorithm, that leads to better segmentation results.

In the next section we start with an overview of naval ship images and their special properties. Section 3 and 4 explain the theoretical background of the two approaches that form the basis for our enhanced *MorphCut* version in section 5. The following section 6 gives results we discovered during implementing and extending the algorithms. Finally, we conclude and give a short outlook in the last section.

Figure 1: A naval ship image.

2 Naval Ship Images

In fact, we consider color images, even if they seem to be gray valued ones (Figure 1). Although all these images ought to consist of the same contents - dark-blue water, light-blue sky and gray-colored ships - it is hard to predict the precise color distributions in the forefront.

Depending on weather conditions and exposure properties the water tends to vary between shades of blue and green with a strong affection to gray. In addition, sunlight beams are reflected from the surface of the surrounding water, causing more or less resplendence to the viewer. Even the sky varies between different shades of blue and gray, sometimes even cloudy and nebulous. This results in a high contrast, that in turn baffles edge detection algorithm that rely on contrast differences only. For example, a classical Canny detector [6] will give no reasonable results (Figure 2).

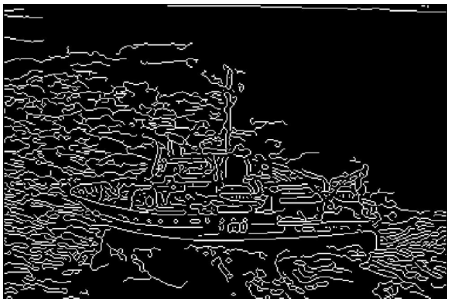

Figure 2: Result of the *Canny* edge detector applied to the naval ship image.

With this in mind, it is obvious that even the ship itself has slight modifications between different types of gray and blue. The color of the water and the sky reflects itself in the hull of the ship, accounting for the blue channel, for example. Sunlight beams adding more resplendence making it even harder to distinguish from the water.

3 The basis - GraphCut

The *GraphCut* algorithm is generally based on energy minimization. The energy function consists of two parts: a probability term P and a contrast term C. The influence of each part can be adjusted with the weighting factor γ.

$$E = P + \gamma \cdot C \tag{1}$$

Gray-valued histograms are used to compute two probabilities for each pixel from the image: one for the background and one for the object part, i.e. how well the pixel color fits into the background/object histogram. A cost measure $P_{z,\alpha}$ for assigning a pixel z to the wrong class α - background or object - has to be defined, for example the log-likelihood function of their probability to fit into each histogram. The total sum over all costs is the first objective to be minimized:

$$P = \sum_{z \in I} P_{z,\alpha} \ . \tag{2}$$

The second (optional) smoothing term describes an arbitrary measure of contrast in the image. Usually this is calculated from the image gradient between all neighboring pixels N, but other edge detection techniques are also possible. At least a kind of an edge map, where edges have higher values than non-edges has to be provided to the algorithm. It is used to make the segmentation boundary fall on

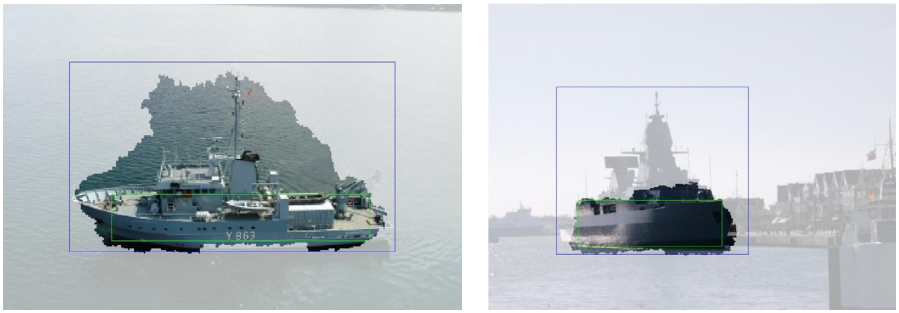

Figure 3: Segmentation results with *GraphCut*: initial segmentation is given with two rectangles (outside the blue one is background, inside the green one is the object).

edges more likely.

$$C = \sum_{(p,q)\in N} C_{p,q} \cdot \delta(\alpha_p, \alpha_q) \tag{3}$$

where

$$\delta(\alpha_p, \alpha_q) = \begin{cases} 1, & \alpha_p = \alpha_q \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

All contrast terms $C_{p,q}$ of pixels lying on the segmentation border - i.e. where the pixel p and q belong to different classes α, expressed by the indicator function $\delta(\cdot)$ - are summed up. So the minimization criterion is to find the shortest possible segmentation border that gives the smallest sum over its contrast terms. The contrast between neighboring pixels p and q can be expressed as

$$C_{p,q} = \exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2 \cdot ||p-q||}\right) , \tag{5}$$

where the norm $||p-q||$ indicates the (spatial) Euclidean distance between two neighboring pixels p and q. In a 4-way connectivity this is, of course, always one. The variance σ^2 over all differences in intensity can be seen as the noise floor present in the picture.

From these two properties of every pixel - one belonging to the object or the background, the other beeing an edge or not - a graph is built. More precisely a so called S/T-graph is build, where the two terminals S and T represent the object respectively the background. Edges from and to these terminals are weighted with the corresponding object/background probabilities. Neighboring pixel are connected with edges in 4-way neighborhood, weighted with the corresponding contrast.

For the initial background/object distribution, two regions containing background parts and object parts of the image have to be specified by the user first. The remaining unknown parts of the image will be segmented based on these histograms

with a standard minimum-cut/maximum-flow algorithm. This resulting segmentation is proved to be the global minimum that can be found with the two given energy terms. Therefore an optimum solution for the segmentation task has been found and the algorithm terminates. Pixels that are connected to the S terminal belong to the object, pixels connected to the T terminal represent the background.

The results (Figure 3) are binary - hard - segmentations between object and background. In case of failures, the initial specifications for object and background have to be refined and the *GraphCut* algorithm applied again. There is no user interaction after the algorithm has been started.

4 Adding color and iterations - GrabCut

GrabCut extends this algorithm in a natural way to color images. The energy minimization (1) stays the same, but the two terms are calculated in a slightly different manner.

The data distributions are now retrieved from Gaussian Mixture Models (GMMs) taking into account the statistical dependencies between the three color channels. This is the essence of the *GrabCut* algorithm to work, so - as a drawback - it won't work for single-layered, gray scale images. Two GMMs with five components k for the distribution of background pixels and vice versa five components for the object pixels are used. The probability of a pixel z belonging to the component k is:

$$M_{z,k} = \frac{1}{2\pi\sqrt{\Sigma_k}} \exp\left(-\frac{1}{2}(z-\mu_k)^T \Sigma_k^{-1}(z-\mu_k)\right) . \tag{6}$$

Mean μ_k and covariance matrix Σ_k are calculated separately for each component k with respect to the three (color) channels. Actually this is a realization of the Expectation Maximization algorithm (EM) widely used for GMMs [7]. As cost measure for assigning a pixel to the wrong component, the negative log-likelihood is used as before.

Edge-ness is again contrast-based, but now calculated as the vector norm in the three (color) dimensions. For example the Euclidean distance between two pixel p and q. The contrast terms $C_{p,q}$ in (3) now are:

$$C_{p,q} = \exp\left(-\frac{||z_p - z_q|\,|^2}{2\sigma^2 \cdot ||p-q||}\right) . \tag{7}$$

In addition *GrabCut* is an iterative algorithm where the calculated segmentation border is refined automatically (or by the user) in every step. Only the background parts are changing (and increasing) over time. Image pixels classified as object during one iteration step are regarded as unknown again in the following one. Since the global energy, which has to be minimized, is decreasing monotonically and convergent, a suitable stopping criterion is straight forward. User interaction for the

Figure 4: Segmentation results with *GrabCut* after 25 iterations: the initial background selection is given with the green rectangle.

initial estimate can be less than in *GraphCut*: only parts of the background have to be selected first, everything else is regarded as unknown. Object parts can be added on an optional basis, whereas in *GraphCut* these are also mandatory initial values.

The results (Figure 4) are again binary segmentations between object and background. Due to the iterative nature an intermediate refinement of all object/background parts can be done by the user in each iteration step. Each iteration step is almost the same as for the *GraphCut* algorithm: a S/T graph is build from the two data distributions and the contrast values. The resulting segmentation is calculated by the minimum-cut/maximum-flow algorithm in each iteration step. It is again the optimal solution for the input data of this iteration.

5 Inserting morphological operations - MorphCut

Observing the iterative calculation in *GrabCut*, we found out that the border around the object tightens too fast. Once a pixel of the image is treated as *background* there is no way back to become *unknown* or even the *object* again. So it is necessary to relax this classification scheme to get softer.

In addition, the color distributions do not have enough time to adapt themselves to the fast changing object content at the first few iterations. This is fortified by the low color information contained in the ship images. It might be useful to build suitable GMMs first, before applying the algorithm or to adapt them faster during the first iterations.

Therefore we introduced an intermediate operation between two consecutive iteration steps, that enhances the segmentation border again. So pixels near the border which have already been classified as *background* are able to become *unknown* or *object* in the following iteration step again. Since this is more useful in the beginning

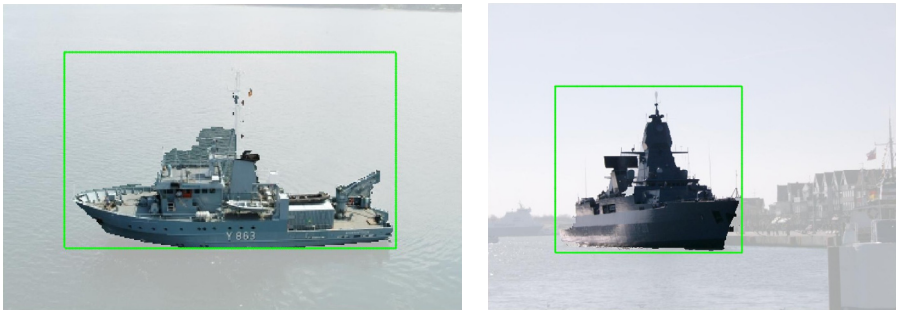

Figure 5: Segmentation results with our new *MorphCut* after 25 iterations: the green rectangle is again the initial background selection.

of the algorithm, the amount of the extension is decreasing slightly over the time.

Enhancing or growing a binary object H is done with the so called *dilation*, a morphological operation. The dilation is only defined in terms of mathematical set operations and only valid for binary images [8]:

$$G_{j,k} = F \oplus H = \bigcup_m \bigcup_n (F_{m,n} \cap H_{j-m+1,k-n+1}) \ , \tag{8}$$

A second (structuring) element H is used. Every position of this structuring element *touching* the original segmentation is marked as the new segmentation. Naturally, the bigger this structuring element is the more the original segmentation is grown. We used the MATLAB built-in functions `imstrel` and `imdilate` to produce a rounded structuring element and to calculate the dilation operation.

The benefit is, that the dilation operation can fill up small holes that exist after a *GrabCut* iteration. These holes are often not wanted, especially when dealing with compact objects like ships. Clustered segmentations will be joined together again, if their distance is closer than twice the current amount of growing.

The results in Figure 5 are obviously better than the preceding ones (Figure 4). Of course, *GrabCut* is also capable of producing such results, but more user interaction between the iteration steps are needed. Our new approach achieves this automatically from the initial user input of the background part. 25 iterations were used again, the involved dilation operation enhanced the intermediate segmentation results by 10% after the first step, decreasing to 0.4% before the last one.

Convergence to an almost stable segmentation is slower now, because more iteration steps than in *GrabCut* are needed. In addition, the global property of the preceding *GrabCut* algorithm - the monotonically decreasing of the energy function that is to be minimized - is violated. However, this does not seem to be a big

drawback: the energy minimization is still working in general, sometimes slightly disturbed by the dilation operation. Nevertheless the decreasing rate of the dilation ensures a stable segmentation result in the end as well. Actually, it seems to be even more stable than the *GrabCut* result, since the GMMs are fitting better to the color distribution in the image now.

6 Results

We used Kolmogorov's *min-cut/max-flow* implementation [9] for the core minimization part. Around this C-implementation we implemented the algorithms using a mixture of C-code and MATLAB scripts. This is certainly not the fastest, but a very flexible implementation. As a direct result we did not achieved the proposed *less than* 1 *sec* processing time for images sizes of 450x300 pixels [3]. In fact, our implementation runs in the order of several seconds for *each* iteration step.

The segmentation using the basic *GraphCut* in Figure 3 is clearly not sufficient for naval ship images. No color information is used at all. The heavy texture information from the background is misleading the algorithm into a scattered segmentation border. A sharp object with a blurry background would be needed for a more reasonable result. Actually there is no *perfect* segmentation available yet. So we only consider a subjective human measure of quality here.

Figure 4 shows the two ships images after 25 iterations of *GrabCut*. We did not use any manual refinement of the background during the iteration steps, nor did we use an optional object definition as used for *GraphCut*. The segmentation results are obviously better but with some parts of the objects missing. Even holes are present inside the objects. However: the segmentation results are quite as good as in [3] for their sample images. So the algorithm itself is working fine.

The final results of our new *MorphCut* variant (Figure 5) are currently the best segmentation results we found so far. Relaxing the previously found segmentation for the following iteration step gives the GMMs more time to fit their distributions according to the color data. The contrast term never changes over time. But the relative importance is increasing during execution by the decreasing influence of the dilation operation. A previously found segmentation border is more likely to stay the same the longer the algorithm is running, leading to a stable and pleasant segmentation result without any user interaction.

Adjusting the γ parameter relocates the importance between the color and the contrast term. Since our ship images contain few color information, relying too heavy on the color term won't work at all. In the contrary to much contrast is present in the surrounding water and a high γ value will scatter the segmentation border around. A reasonable trade off is often hard to predict, but values between 5 and 50 (as proposed in [5]) tend to work well.

7 Conclusion & Outlook

Using naval ship images the segmentation results are worse than with usual images. This arises from the special properties of these images: having low color information and many textured regions containing high contrast. Standard segmentation algorithms based only on color or contrast information are hardly able to compete with that.

But nonetheless, *GrabCut* and our new *MorphCut* variant still outperform other segmentation algorithms. In sense of needing less user interaction, *MorphCut* outperforms *GrabCut* as well, without degrading the overall segmentation result. On the contrary - at the cost of a few extra computations for the dilation operation - the segmentation results are even better using no user interaction at all. Additional operations, for example deleting tiny clusters which are not connected to the main object, can be included easily.

The computation has been slightly increased compared to *GrabCut*. Additional operations have been introduced and more iterations are needed in general. Because no more user interaction is needed, a faster realization can be expected. The total running time is strongly related to the size of the input image. A downsampling or multi-scale approach might be useful.

References

[1] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *Proc. IEEE International Conference on Computer Vision*, 1987, pp. 259-268.

[2] E. Mortensen and W Barrett, "Intelligent scissors for image composition," *Proc. ACM Siggraph*, 1995, pp. 191-198.

[3] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut - interactive foreground extraction using iterated graph cuts," *Proc. ACM Siggraph*, 2004.

[4] A. Blake, C. Rother, M. Brown, P. Perez, and P. Torr, "Interactive image segmentation using an adaptive gmmrf model," in *Proc. European Conference on Computer Vision*. 2004.

[5] Y. Boykov and M.-P. Jolly, "Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images," 2001, Vol. I, pp. 105-112.

[6] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986, Vol. 8(6), pp. 679-698.

[7] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society*, 1977, Series B, 39(1), pp. 138.

[8] W. K. Pratt, *Digital Image Processing*, John Wiley and Sons. Inc., 2001.

[9] V. Kolmogorov, "Maxflow implementation," http://www.cs.cornell.edu/People/vnk/software.html.