Color segmentation for scene exploration

Gyuri Dorkó, Dietrich Paulus, Ulrike Ahlrichs

Color segmentation for scene exploration

Gy. Dorkó[1] D. Paulus and U. Ahlrichs

Universität Erlangen–Nürnberg
Institut für Informatik
Lehrstuhl für Mustererkennung
Martensstr. 3, 91058 Erlangen

dorkogy@freemail.hu
[paulus/ahlrichs]@cs.fau.de
http://www.mustererkennung.de

Abstract

In this contribution we describe the results of the color structure code algorithm and a modified split and merge algorithm, which both segment a color image into regions. We integrate these algorithms into our system for knowledge–based image analysis and compare the two distinct methods concerning their applicability in our system for knowledge-based exploration of a scene.

1 Introduction

Color images can be segmented into regions, lines or points based on the distribution of colors, similarly to the algorithms in gray–level images. For color images, homogeneity or gradients have to be defined on vector valued functions, rather than the scalar functions used for gray–level images. An example is the well–known region segmentation using the split–and–merge strategy which was extended to color in [DJ93]; in [HP97] we made experiments on different color distances using this segmentation algorithm. Another region segmentation that was proposed for color images is the so–called color stucture code, CSC [RB95]. There exist algorithms for color image segmentation which have no equivalent in gray–level segmentation. These algorithms work locally and classify individual pixels into color classes without considering the spatial neigborhood, e.g. in [ABS90]. In a second step, pixels of one class which are spatially close are grouped to regions. In the following we will not consider such segmentation algorithms.

The paper is organized as follows: we give a description of the color structure code algorithm, in Sect. 2, which we apply in Sect. 3 to natural images taken from the sample set used in our application domain, which is the recognition of office tools in a natural environment. Problems during CSC segmentation and our solutions are described in Sect. 2.3. Several filters for color images are applied in Sect. 3.1. In Sect. 3.2 we outline our system and describe how region segmentation is integrated into it and list results. We conclude in Sect. 4 and give an outlook on further work.

2 Color Structure Code Segmentation

The CSC color segmentation which we use here was first introduced in [RB95]. Although it was described previously in this series of workshops, we briefly repeat the description using some other formalism, so this contribution is self-contained. The base of this color image segmentation method is a special data structure described in Sect. 2.1. The code element which represents the regions is described in Sect. 2.2 besides of its creation procedure which is the most relevant part of the algorithm. Sect. 2.3 points out some problems of the segmentation and its implementation and it also gives some methods how to solve them.

[1]Gy. Dorkó is funded by SOKRATES. He is a student in Kálmán-Kandó Polytechnic, Budapest.

2.1 Hexagonal structure

The CSC segmentation is based on a special data structure so-called hexagonal island structure, Fig. 1. Let the smallest circles be pixels and seven pixels in a specified structure (Fig. 2, left) compose an island. These small islands are so-called “islands on level 0”. Each seven islands compose another island on level 1 in the same structure and seven of these new islands construct another one on level 2. Continue this procedure until one island covers the whole image. Notice that an island on level N contains seven islands at most from level $N - 1$ or when $N = 0$ it contains seven pixels instead.

Notice that two neighboring islands on level N are overlapped and they share one and only one sub-island; on level 0 this means they share a single pixel. To apply this structure to an orthogonal bitmap can bring up some difficulties. One simple method for this transformation is to shift every other row theoretically by half of a pixel either left or right. Fig. 2 (center) illustrate an island on level 0 and level 1 over an orthogonal bitmap. By the help of Fig. 2 (rightmost), an island is shown on level 2 but let us point out that the overlapping part of its sub-islands are not precise. It is confined to one shared sub-island and not other additional pixels.

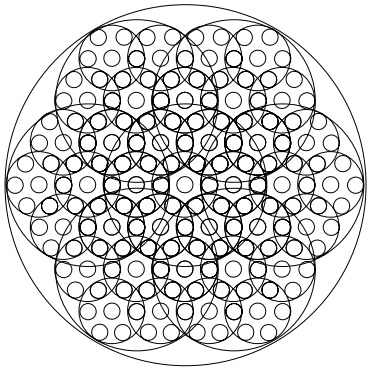


Fig. 1: The hexagonal topology

This hierarchical structure would seem to be complicated but it worth while to build up because this will make the algorithm so fast and accurate later.

2.2 Creating code elements

On the described hexagonal hierarchical structure each island has one or more so-called code elements. On level 0 a code element simply means linked pixels which are similar in color. Each island on level 0 has up to seven code elements depending on the image and the applied color similarity. It is also important that the whole island is “covered” by code elements. On an island on level N where $N > 0$ a code element is defined as liked code elements of its sub-islands which are connected. A code element can be implemented as an array of pointers to other code elements and on level 0 as an array of image coordinates. The code elements are stored on the appropriate islands together with other attributes like area (number of pixels) and mean color.

In our implementation this creation procedure is a simple request to create all code elements of the “top level island” and it does everything else due to the following recursive algorithm: First of all an island on level $N > 0$ calls the procedure to create code elements on all of its sub-islands if necessary (recursion). After all the code elements of its sub-island have been computed we can create some new ones on the island itself and link the appropri-

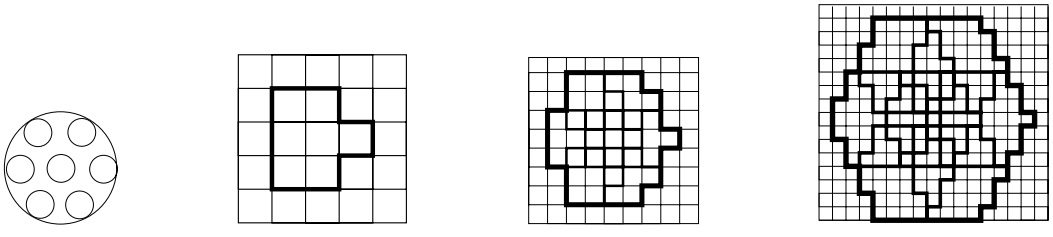


Fig. 2: From left to right: islands on level 0 on hexagonal, and orthogonal strcture. island on level 1, island on level 2 over an orthogonal bitmap

ate sub-code elements to their lists. Two sub-code elements must be attached to the same code element if they are connected and must not if they are not. Whether two elements are connected or not, can be detected easily by comparing their list of sub-code elements. If they have an identical entry they are connected. This part of the algorithm is also called the linking phase.

Trivially, the code element creation is different on level 0 from that on any other level. Here we create code elements for the pixels itself on each island. Two pixels are connected to the same code element if they are similar in color as mentioned earlier in this section.

Furthermore during the linking process it is unnecessary to create code elements with only one sub-code element because if the algorithm did not find any connected element on the same level means that this would be a so-called “root code element” and it can be collected globally in an array.

This “hierarchical region growing” part of the algorithm would not have been as powerful and accurate as it is without the following splitting phase [RB95] which extend the segmentation with a global view. That means, before two connected sub-code elements are to be linked we check the color similarity between their mean colors. If they are not similar they will not be liked; moreover they must be separated by this method because they share a region on a previous level. The subregions of this common region will be associate to the one which is closer in color. Therefore some additional splitting could be required on the lower level and in these cases we should act in the same way which means an elegant recursion in the implementation.

2.3 Final steps

After the previous phases the results (the regions) are represented hierarchically. Each region has a root element which is stored either as an “immediate” root[2] or as a simple code element on the “top level island”[3]. Usually this hierarchical information cannot be used directly. The final phase is responsible for converting them into an appropriate format. This can be a label image[4]or a set of chain codes. The other important task of this phase is to clear up some problems and difficulties of the original segmentation algorithm.

Problems and difficulties

During the implementation of the described algorithm we may meet some difficulties or problems. In rare cases the splitting phase can cause unconnected regions: To split one code element from another always cut a region out. In absence of this sub-region, the original area represented by the “maimed” code element is not one region any more but two. Fig. 3 shows a small example: The pixels marked “A” “O” and “X” were connected to the same region but because of some reasons they had to be separated into two parts (“A” and “X”). If “O” belongs to the same region as “X” it would not cause any problem but in the other case cutting the pixel “O” out makes the region “X” unconnected. To avoid this and similar confusions we integrated an algorithm after the splitting phase to detect these problems.

“Empty” code elements are another effect of the splitting method but instead of the previous problem it can be detected and solved easily. Another region connectivity problem is due to the special hexagonal structure, directly to the special neighborhood relation. It is

[1] If a code element did not find any partner on a level it becomes a kind of root. These special elements can be collected separately during the execution of the algorithm.

[2] The one and only island on the topmost level. It covers the whole image.

[3] A matrix with the same size as the original image. Each element corresponds to a number which specifies the associated region (so-called region number).

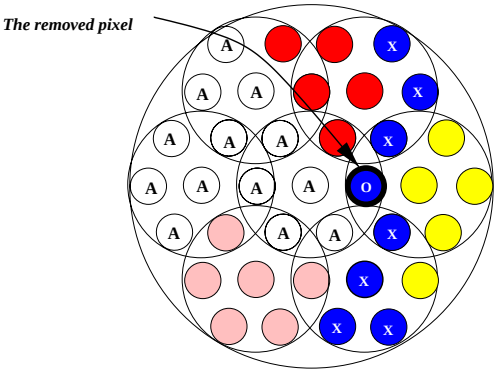


Fig. 3: A region becomes unconnected

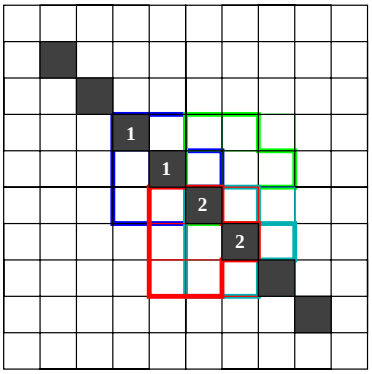


Fig. 4: Homogenous diagonal line

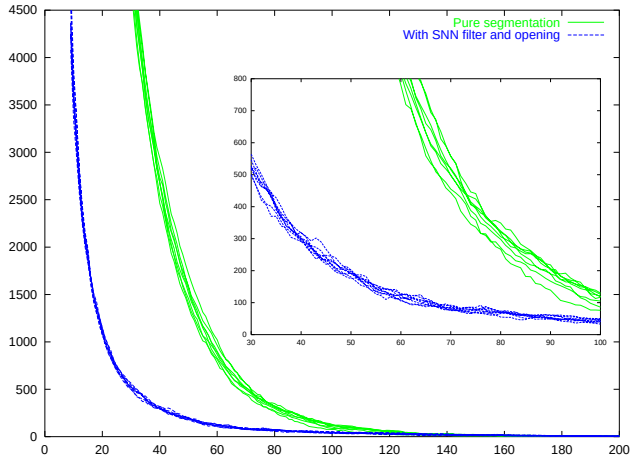
neither 4-connected nor 8-connected. An example is shown on Fig. 4 where a one pixel wide diagonal line has been segmented as multiple regions but not as single pixels. The pixels marked with the same numbers are segmented into one region. Due to the special neighbor relation this line is segmented to small parts and each part has 2 pixels.

An easy solution of the previously described problems is an extention of the final phase. The detection and the correction of these problems are easier on the output (converted) structure than on the hierarchical representation. If the output is a label image, this simply requires relabeling the whole structure at the very end of the algorithm. In the implementation it means applying a flood-fill algorithm to each region. If it uses 4-connectivity the last problem will be solved. The empty code elements will disappear as well as the unconnected regions because their parts will be “reordered” with different numbers. It is also important to mention that this method is not the perfect solution but it solves the problem very quickly.

3 Experiments

The primary goal of our knowledge–based system for scene exploration is to combine active vision, object recognition, and knowledge–based pattern analysis. One application of the architecture for this system which is described in [AFPN99] is to find objects in an office room.

In Fig. 12 we see left the input image of an image taken from the sample set of office tools, in the center the result computed with the method described in [DHP95] is shown, right we have our new results computed with the CSC algorithm. The CSC algorithm has one major parameter, which can be tuned in the implementation, namely the threshold which determines the similarity of two color vectors depending on a chosen color distance. The results vary considerably with the selection of this parameter; the results in Fig. 5 show the mean of the number of computed regions depending on the threshold. In one hand the following experiment shows that after applying the segmentation routine with the same paramters the number of regions are about the same on different images (with same dimensions and conditions). In another hand we could see that it depends on the pre-processing and the final phase as well as the threshold level. We conclude it worth while to apply a filter (in our example an SNN filter in a special hexagonal structure) and an opening by the final phase in order to avoid small regions.



The similarity of the curves resulting from different pictures shows, that the choice of the threshold can be deterimined for an experiment globally, without the need of adjusting it to an individual picture.

Fig. 5: Number of regions with and without filtering and opening for various thresholds

Number of regions	
Un-filtered (Fig. 7)	512
Vector impulse noise (Fig. 8)	428
SNN (Fig. 9)	364
Vector median (Fig. 10)	292
Mean (Fig. 11)	529

Time for computation	
Filtering:	0.21 sec
Structure creation:	0.14 sec
Segmentation (linking+splitting):	1.68 sec
Finalizing (labeling+opening):	0.42 sec

Table 1: Results of experiments: Number of regions (left) and computation times (right)

3.1 Color Preprocessing

Before a segmentation algorithm is applied to an image, we usually do some preprocessing. Filtering is one of the most important and usual part of that. We applied a symmetric nearest neigbor filter (SNN), a color vector median, as proposed in [SSW99], and a color mean filter to the input images which we also overlayed with noise. The original image is shown in Fig. 6; Fig. 7 to Fig. 11 are the segmentation results represented by contours. The number of regions found are listed in Table 1 left. The computation times for our programs which were compiled with GCC and computed on a Pentium III computer (600 MHz) under Linux are listed in in Table 1 right; the image size was 768 × 576.

3.2 System

In order to localize objects in an office room, we apply various steps which require color image processing.

First, objects are individually presented to the system in order to compute color histograms. The objects are then put into the scene, i.e. into the office room. An estimate of the object's position is calculated using color histogram backprojection. In most cases, the intensity-normalized *rg* space performed best here, as described in [CPAH98]. An active camera is then moved on a linear sledge and points are tracked during the movement. Color again improved the results when compared to tracking of points in gray–level images [HP97]. The result of tracking is combined with calibration information on the camera and the results of histogram backprojection to compute 3D estimates for the positions of objects.

Close-up views are now captured showing details of the scene at the estimated object positions. These images are subject to color region segmentation. The images used in this paper are taken from such experiments. Features of the color regions, such as size and mean color are further used for knowledge-based verification of the object positions. A semantic network is used for knowledge representation [AFPN99].

The computation consists of a (possibly iterated) sequence of individual processing steps. During object verification, close–up views of objects captured by the active camera are subject to segmentation. We provided a common interface to the two segmentation algorithms used; this way they could be dynamically exchanged in the program. We also varied the color space and distance measure.

4 Conclusion and Future Work

The CSC algorithm was integrated into the system for office exploration described in Sect. 3.2. It now provides an alternative to the segmentation [DHP95]; with respect to computation times, the CSC algorithm is superior, i.e. faster. Performance evaluation of image segmentation is a non–trivial subject. In principle, the parameters of each algorithm used in the sequence of processing steps has to be varied and the results have to be judged according to some criterion; the number of regions, as shown in Fig. 5 cannot serve as an ultimate criterion. In our case, the evaluation is based on the final recognition rate and the rate of successfully found objects in the office room. This work is carried out currently. First results showed recognition rates in the range of 70% which is similar or slightly superior to our previous results; the computation times were lower with the CSC approach.

References

[ABS90] I. Andreadis, M.A. Browne, and J.A. Swift. Image pixel classification by chromaticity analysis. *Pattern Recognition Letters*, 11:51–58, 1990.

[AFPN99] U. Ahlrichs, J. Fischer, D. Paulus, and H. Niemann. Approach to Active Knowledge-Based Scene Exploration. In M. Bramer, A. Macintosh, and F. Coenen, editors, *Research and Development in Intelligent Systems XVI – Proc. of the* 19^{th} *SGES International Conference on Knowledge-Based Systems and Applied Artificial Intelligence (ES99)*, BCS Conference Series, pages 289–301, Cambridge, 1999. Springer.

[CPAH98] L. Csink, D. Paulus, U. Ahlrichs, and B. Heigl. Color Normalization and Object Localization. In V. Rehrmann, editor, *Vierter Workshop Farbbildverarbeitung*, pages 49–55, Koblenz, 1998. Föhringer.

[DHP95] J. Denzler, B. Heigl, and D. Paulus. Farbsegmentierung für aktives Sehen. In Rehrmann [Reh95], pages 9–12.

[DJ93] M. Dubuisson and A.K. Jain. Object contour extracting using color and motion. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 471–476. New York City, 1993.

[HP97] B. Heigl and D. Paulus. Punktverfolgung in Farbbildsequenzen In D. Paulus and Th. Wagner, editors, *Dritter Workshop Farbbildverarbeitung*, pages 87–92 & 105, Stuttgart, 1997. IRB-Verlag.

[RB95] V. Rehrmann and M. Birkhoff. Echtzeitfähige Objektverfolgung in Farbbildern. In Rehrmann [Reh95], pages 13–16.

[Reh95] V. Rehrmann, editor. *Erster Workshop Farbbildverarbeitung*, volume 15 of *Fachberichte Informatik*, Universität Koblenz–Landau, 1995.

[SSW99] B. Smolka, M. Szezepansik, and K. Wojciechowski. Random walk approach to the problem of impulse noise reduction. In K.-H. Franke, editor, *5. Workshop Farbbildverarbeitung*, pages 43–50, Ilmenau, 1999. Schriftenreihe des Zentrums für Bild- und Signalverarbeitung e.V. Ilmenau.

5 Color Images

Fig. 6: Noisy original

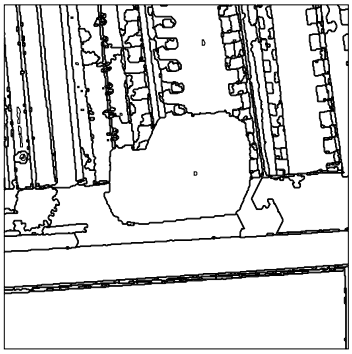


Fig. 7: Unfiltered

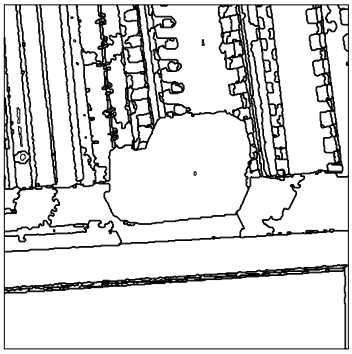


Fig. 8: Impulse-Noise

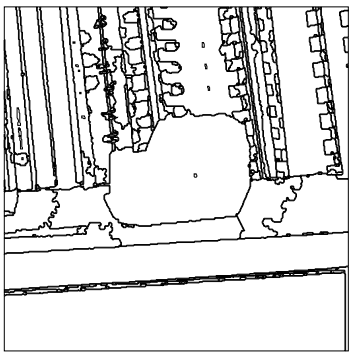


Fig. 9: SNN

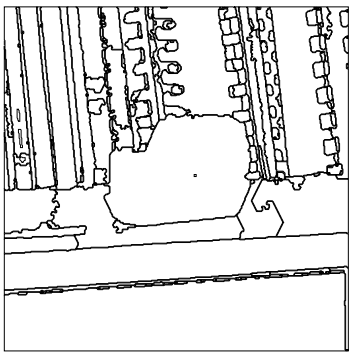


Fig. 10: Median

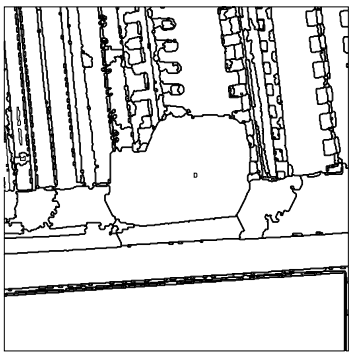


Fig. 11: Mean

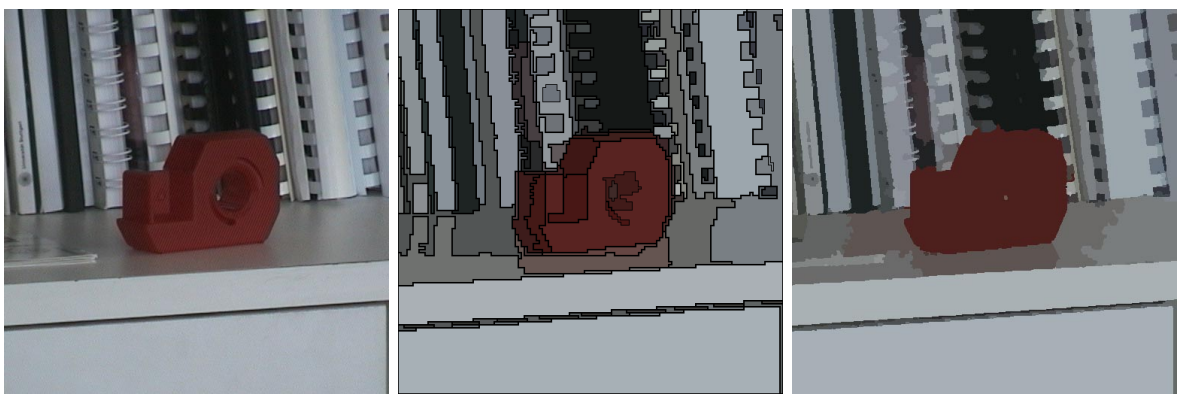


Fig. 12: Input image (left), split and merge segmentation (center), CSC segmentation (right)

The test image is shown in Fig. 12. Impulse noise was added to that picture as shown in Fig. 6.[4]

[4]We used programs provided by B. Smolka for that purpose, which were also used in [SSW99]. The authors acnowledge that fruitful cooperation.