Tracking Regions

Felix von Hundelshausen and Raúl Rojas

Free University of Berlin, Institute of Computer Science
Takustr. 9, 14195 Berlin, Germany
{rojas,hundelsh}@inf.fu-berlin.de

Abstract. In this paper we present a simple and new algorithm that tracks the contour of several homogenous regions in a sequence of images. The method exploits the fact that, when i.e. observing a moving object (exposing a homogenous region), the regions in two consecutive frames often overlap. We show that the method is valuable for the RoboCup domain: It allows to track the green playing field and the goals very efficiently, to detect the white marking lines precisely, enabling us to recognize features in them (the center circle, the quatre circles, corners, the rectangle of the penalty area,...). It is also useful to find the ball and the obstacles. Furthermore, it provides data for path planning based on potential field methods without further computation. We compared the algorithm with the fastest existing method and measured a speed enhancement of 30 percent. In contrast to other methods, our algorithm not only tracks the center of blobs but yields the precise boundary shape of the objects as a set of point sequences. First tests with real world data have confirmed the applicability for other domains than RoboCup.

1 Introduction

The algorithm presented in this paper emerged from our aim to localize our middle size robot by detecting the marking lines on the playing field. Our robot is equipped with an omnidirectional catadioptric vision setup, with the camera looking upwards into a convex mirror. In order to recognize features like the center circle, the quatre circles, corners, rectangles, T-junctions, etc., it is important to detected the lines precisely in the images, without false positives and last but not least connected. The latter issue is important for fast feature detection. Only if the order of line points is known, we can efficiently calculate curvature measures and tangent directions, which is important for both, relative matching as described in [14], and feature recognition.

To detect the marking lines the straight forward way is to use an edge detector. Possible solutions could be based on the Roberts Cross Edge Detector[19], the Sobel Edge Detector [18], the Canny edge detector[6], the compass operator [20], edge detectors using the Laplacian of Gaussian, Gabor filters[16] or wavelet based solutions [15]. However, applying such a scheme to the whole image is time consuming. In particular, when processing 15 up to 30 frames per second for real-time vision, it is important to restrict the area within the image to which the detector has to be applied. Furthermore, the problem of connecting the detected edge points remains [8]. Another approach, which can be found in [11], uses the Hough transform [10] for grouping, but this is not efficient.

To overcome the problem of linking model based prediction of edges has been used extensively over the last years [3]. However, the problem is that the model has to be known, and that the initial pose must be given. Although the determination of the initial pose is possible by propagation of a probability distribution [2], it is time consuming. Moreover, problems with matching under larger translations are known [7], since the models converge to local minima.

We approached the line detection problem differently: The idea is to detect the regions between the lines (see figure 1a). Since their boundaries neighbor the lines, they can easily be recognized: Just step around the boundaries, calculate the normal of the boundary curve at the actual position and apply a direction specific detector. For instance, search for a green-white-green transition.

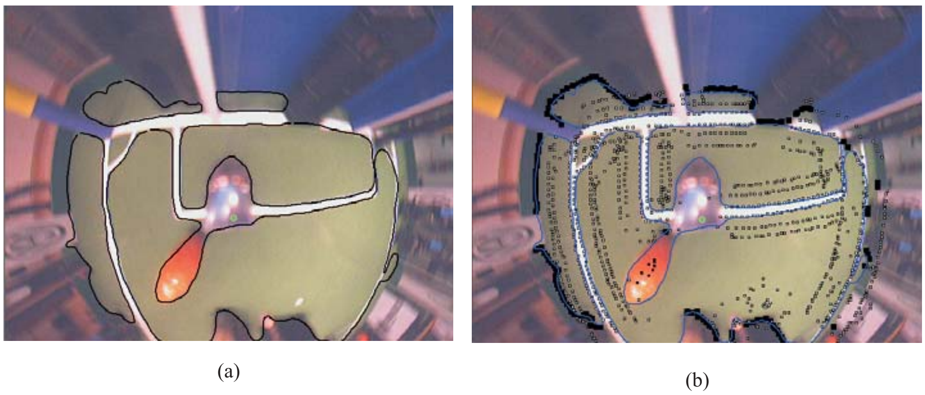

Fig. 1. (a) The green regions are tracked. The boundaries of the regions are visualized by black lines. They are represented as sequences of points. (b) Objects are searched along the boundaries. Black points mark detected obstacles, for the marking lines three points are investigated along the normal. One on the line, and two at each side of the line to verify a green-white-green transition. Detected ball points are painted black for the sake of visibilitiy.

The method is also useful for detecting the obstacles and the ball: Just look for something orange or black next to the boundaries of the green regions (figure 1b). Doing so, false balls outside the playing field are not detected since they are not next to a green region in the image.

In this paper, we will show how the regions and their boundaries can be tracked very efficiently. We use the results of the last image to calculate the solution for the next, and on average only 10 percent of the pixels have to be accessed per frame. Our tracking algorithm is based on the region growing paradigm [4][22] [21][1][9][17]. However, we extend the paradigm to track regions over time.

The remainder of this paper is organized as follows: Section 2 reviews the basic region growing algorithm and describes the key observation that leads to the extension of the algorithm. In section 3 we describe the boundary extraction. Section 4 illustrates the generality of the algorithm. Section 5 shows results of the algorithm applying it within the RoboCup domain and comparing it with the currently most used tracking algorithm by robotic soccer vision systems[5]. Finally, section 6 concludes the paper.

2 Extending the Region Growing Paradigm

We first give a short review of a specific region growing algorithm which will be extended afterwards.

2.1 Region Growing by Pixel Aggregation

In *Region Growing by Pixel Aggregation* one starts with a small region (i.e. a single pixel) and consecutively adjoins pixels of the region's neighborhood as long as some homogeneity criterion is fulfilled. This method can be implemented in the following way: We reserve a two-dimensional array V which has the same size as the image and which has a boolean entry for each pixel that indicates whether the respective pixel has been visited yet. Further, we maintain a queue Q that stores the spreading boundary pixels (their coordinates) of the region during the execution of the algorithm. We refer to the stored elements in Q as *drops*, following the idea that we pour out a glass of water at the seed pixel and that the drops of this water spread over the region. Q just stores the boundary drops at each time during the execution of the algorithm. Initially, if we start with a single pixel, Q stores a single drop corresponding to the pixel which is also marked as visited in V. The algorithm continues with a loop which is performed as long as any drops are stored in Q. In each pass of the loop one drop is extracted from the queue and the neighboring pixels are investigated. In the case of a 4-neighborhood we inspect the top, right, bottom and left pixels. After assuring that the pixels have not yet been visited, respectively, we determine whether they hold for a specific homogeneity criterion, color similarity for instance. For each pixel that conforms with this condition a new drop is instantiated and stored in Q. After the loop terminates Q is empty and the region is marked in V. Figure 2 shows an example of a growing process that finds the region of the yellow goal.

2.2 The Key Observation

Our goal is to efficiently track regions over time. To give a specific example, we want to track the yellow goal, while the robot moves. Assume, that the robot starts at pose P_A and takes an image I_A. Then the robot moves a little to pose P_B and takes another image I_B. Assume further that we have determined the region A of the yellow goal in I_A and the corresponding region B in I_B by the above

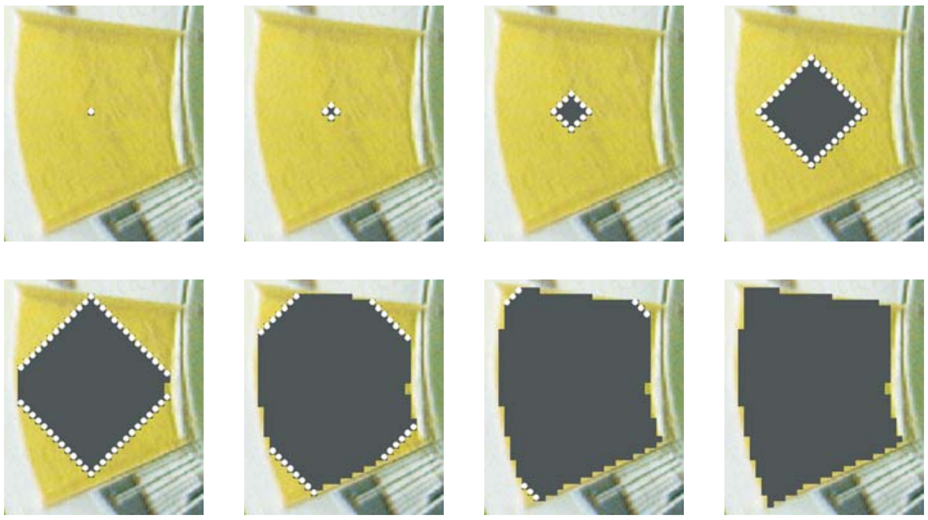

Fig. 2. The array V is visualized by the dark gray surface (visited elements). The white circles represent the drops. Here, the algorithm does not work on single pixels, but on blocks of 4×4 pixels.

region growing algorithm. If the video frequency is high enough with respect to the movement of the robot, then the two regions will overlap as depicted in figure 3. If we apply the region growing algorithm separately to image A and B, then the running time for each is linear in the number of pixels (or blocks of pixels) within the respective regions. We want to develop a more efficient method. Assume that we have extracted region A, then roughly speaking, we want to use the drops of the region growing of A to somehow flow to the region of B by making use of the overlap. The drops of region A should first shrink to the intersection S of A and B and then grow to the boundary of B. Thus, in order to find region B we don't start with a seed pixel, but with a whole seed region, the intersection of A and B. To realize the algorithm a method of how to shrink region A to the intersection of A and B has to be developed. This will be done in the following subsection.

2.3 Shrinking Regions

We will develop the shrinking method in two steps. First we consider the case of shrinking a region without stopping criterion. That is, the region shrinks until it vanishes. Next, we modify the method so that shrinking stops at the intersection of A and B.

In the first step, we don't need any image information but just the array V in which region A is marked and a queue of drops at the boundary of that region. As we will need two different queues for growing and shrinking later, we denote the queue used here as Q' to avoid confusion. We apply the same operations as in region growing with one exception: We reverse the meaning of V. As a

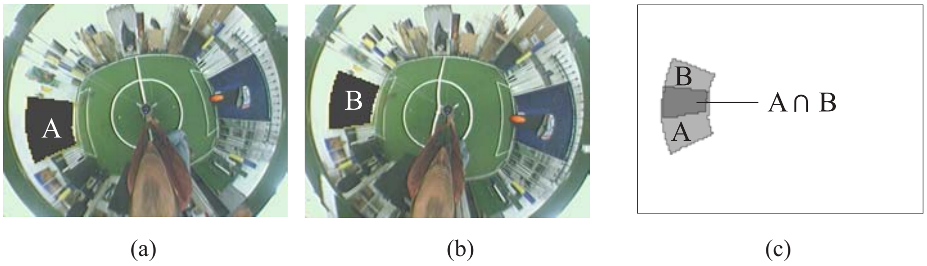

Fig. 3. The regions of the yellow goal are determined (a) before and (b) after the robot has moved. (c) The two regions overlap.

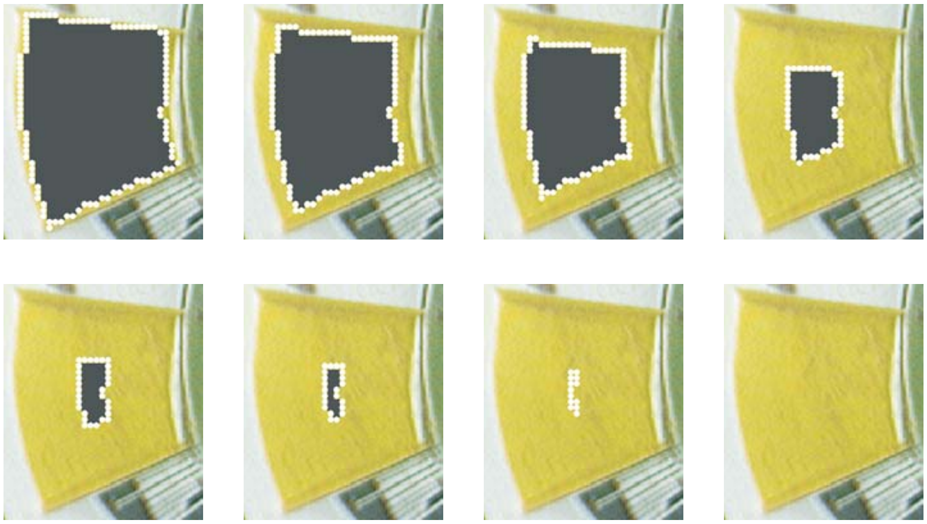

Fig. 4. The region shrinks until it vanishes. For this process no image information is used.

result, the drops can only spread within the region marked in V and since they initially are placed at the boundary their only means of escape is to move from the outer to the inner of the region. Instead of marking elements in V, as in region growing, entries in V are cleared while the drops spread. At the end, Q' is empty and V is cleared completely. This process is illustrated in figure 4.

In the second step we use image I_B to determine when shrinking should stop. We emphasize that the initial region (marked in V) is due to image I_A while only image I_B is referenced during shrinking. Each time after a drop has been extracted from Q' we verify the homogeneity criterion for the corresponding pixel in image I_B. If the pixel belongs to the region, then the drop is not allowed to spread anymore. As a result the region shrinks to the intersection of region A and B as depicted in figure 5. This is exactly inverse to the growing, where drops only spread to neighbors that fulfill the homogeneity criterion.

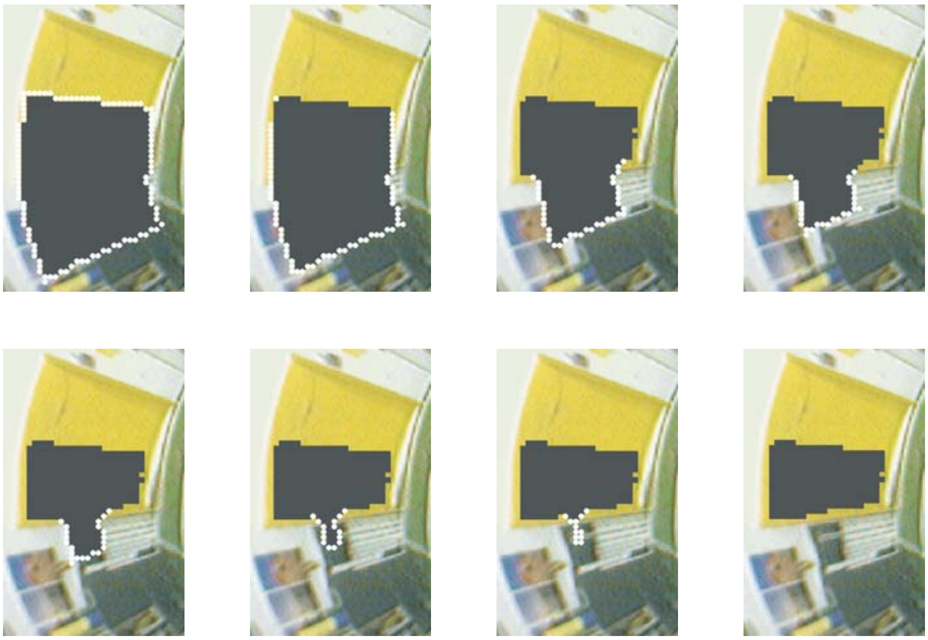

Fig. 5. The region shrinks up to the intersection area, that means, until all drops are within the region of the new image.

2.4 Alternating Shrinking and Growing

Region growing is a well known technique which was extensively studied 20 years ago. However, our new contribution is that region growing can also be used to shrink regions and that alternation of shrinking and growing allows to track large regions extremely efficiently.

To alternate shrinking and growing in order to track regions some problems concerning the interface between the two stages must be solved. The first problem is that after growing the queue of drops is empty but shrinking initially requires a list of drops at the boundary of the region. In the same way shrinking ends with an empty list of drops, but growing requires seed drops. To solve this problem each of the two processes, growing and shrinking, has to build the initial queue for the other procedure. We accomplish this by using two queues, Q and Q'. Growing assumes the initial drops to be in Q and after execution Q is empty and Q' has been built up which stores the initial drops for shrinking. Shrinking runs with these drops and initializes Q for the next growing.

During growing, when the neighbors of an extracted drop are inspected, the drop is inserted into Q' as initial drop for shrinking, if any of its neighbors does not belong to the region. Vice versa, a drop that is extracted from Q' during shrinking is inserted to Q as initial drop for growing, if shrinking stops for this drop.

2.5 Controlling the Tracking

Since the tracked regions can be lost if the movement between two consecutive images is too large, a control mechanism has to be integrated into the loop of shrinking and growing. The control mechanism checks whether tracking has lost the region. In this case an initial search has to be started. Depending on the application this procedure might search within the whole or just within a part of the image for a pixel or a block of pixels having a certain color or texture that satisfies the homogeneity criterion and maybe some other detection criterion. Later, we also describe how to extract the polygonal shape of the regions. If a region gets lost, and an initial search is started, then first several regions can be tracked and by using the information about the size and shape of the regions some of them can be discarded. In this way the computational power is concentrated on the regions of interest. To exclude a region from being tracked one simply has to delete its entries in V and the corresponding drop queues. This can be accomplished by a shrinking without stopping criterion as illustrated in figure 4.

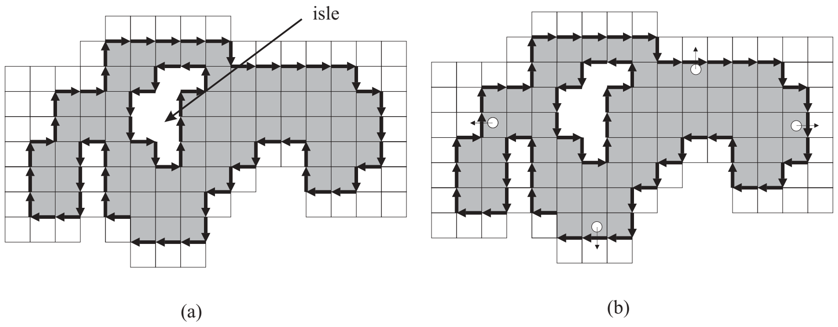

Fig. 6. (a) The boundary of a region is composed of a sequence of edge vectors. (b) The direction of the edge vectors depend on the direction of the spreading at which they are detected during the growing phase.

3 Boundary Extraction

The boundary of each region consists of a chain of small edge vectors (see figure 6a). Each edge vector represents one of the four sides of a pixel and the last edge vector ends at the beginning of the first. During the growing phase, when a drop reaches a border, the corresponding edge vectors are inserted into a special data structure, the *connectivity grid.* It is a two-dimensional array, one wider and one higher than the image. The cells do not correspond to the pixels in the image but to the inter-pixel positions (corners of pixels), respectively. Each cell has 4 bits, marking whether an edge vector starts at the corresponding position, directed up, right, down or left, respectively. Initially, the grid is cleared. After growing the grid contains the edges. Now, starting at any edge, one can follow the edges through the grid and clear them in the same pass. Since situations appear where an edge has two possible successors, it is important to apply a rule: With respect to the current direction, always choose the most left/right turning possibility. In this way, it is guaranteed that all chains of edges are closed. After having extracted the edges, the connectivity grid again is cleared and ready for the next insertion process. This approach is similar to [4], but our method occupies less memory, since [4] employs a supergrid of size $(2n+1) \times (2m+1)$ and we just use a grid of size $(n+1) \times (m+1)$, where $n \times m$ is the size of the image. The result of boundary extraction is as set of point sequences, each forming a closed curve.

4 Homogeneity Criterion

It is important to understand, that the described algorithm works with any homogeneity criterion. We will give some examples. For RoboCup, our images are in YUV 422 format. The YUV bytes of a pixel (24 Bit) are an index into a 16 MB big lookup table. Each entry in the LUT is one byte big, and each of the 8 bits represent a color class. We refer to a pixel's LUT-entry as its class mask. Verifying whether the pixel supports a certain class or collection of classes is then possible with a single AND-operation. To be robust against image noise, we do not run our region-tracking algorithm on single pixels, but on blocks of 4×4 pixels, in case of an image resolution of 640×480. Then, for tracking the green field, we define a block to belong to the green region, if the number of "green" pixels exceed a certain threshold (12 is a typical value). We have tested this with extremely noisy images and we can assert that it works well.

However, the choice of homogeneity criterion is free. For instance, one could define that two blocks of pixels are homogeneous, if there texture is similar (assuming a texture classifier). One could also define, that two pixels are homogeneous, when an edge detector at the respective position yields low response. The region tracking algorithm will run with any criterion. However, it is the task of future research, to find the best criteria, and more important, how the criteria can be calibrated automatically and adapt to changing lighting conditions.

5 Results

The advantage of our region tracking algorithm is that only the non-overlapping parts of corresponding regions of successive images are processed. However, if the regions don't overlap, the algorithm has to lunch initial searches in each frame and degenerates. Therefore the question is, how often do the regions overlap? Of course, the answer depends on the application. In the following, we present results from our RoboCup application. Here, we have tracked three different types of regions: the green playing field and the blue and yellow regions (goal and post markings). While the robot moved through the environment, we computed the fraction of the processed area with respect to the size of the image and the percentage of overlapping of the tracked regions. We determined these values for each pair of successive frames and built the respective average over a longer time

Table 1. The average fraction over a sequence of frames of the processed area with respect to the size of the image and the percentage of the overlapping area with respect to the area of the tracked regions has been determined for different movements and speeds. The last two columns indicate the number of frames and the number of initial searches that had to be started.

Rotation	processed fraction of the image	% overlapping	frames	initial searches
$56°/s$	8.0%	80%	181	11
$74°/s$	8.4%	77%	142	11
$110°/s$	8.8%	71%	97	13
$145°/s$	10.0%	65%	74	13
$200°/s$	11.0%	60%	62	25
Translation	processed fraction of the image	% overlapping	frames	initial searches
$0m/s$	6.3%	93%	138	0
$0.26m/s$	5.9%	89%	114	0
$0.38m/s$	6.8%	84%	82	2
$0.52m/s$	7.0%	82%	61	0
$0.65m/s$	7.2%	80%	48	0
$0.74m/s$	6.9%	81%	42	1
$0.83m/s$	7.9%	75%	40	0
$1.0m/s$	7.8%	73%	31	1
$1.0m/s$	7.7%	79%	24	2

span (see table 1). We also counted the number of initial searches and repeated the experiment for different movements (rotation/translation) and speeds of the robot. Here, we should mention that our robots have omnidirectional wheels and are able to rotate and translate into any direction at the same time.

The tracked regions overlap greatly and only a small fraction of the image data is accessed (below 10%). As expected, the rotation is more disadvantages then the translation. This is because objects that are far away from the robot (as the goals) have a high speed in the image, if the robot rotates. Therefore the regions might not overlap, and initial searches have to be executed. With a rotational speed of $200°/s$ 25 initials searches had to be done within 82 frames. Assuming that an initial search requires accessing all the pixels in the image, the algorithm yet performs well, compared with common methods, which access all the pixels in each frame.

Although the theoretic running time of our tracking algorithm is evidently faster than any algorithm that touches all the pixels in each image, the question is whether this is also true for the practical running time, since the maintenance of the drops, the queues and the connectivity grid might be more time consuming than a straightforward implementation.

Therefore, we decided to compare the algorithm with the color segmentation algorithm proposed in [5], which is believed to be the fastest color tracking algorithm at the moment and which is applied by most teams in RoboCup. The algorithm is based on classifying each pixel into one or more of up to 32 color classes using logical *AND* operations and it employs a tree-based *union find* with path compression to group runs of pixels having the same color class. We calibrated both algorithms to track green, blue and yellow regions. We did not track the orange ball and black obstacles, because our system uses different methods than color class tracking to recognize them. The following table gives the absolute running times of the algorithms over 1000 frames with the robot moving. Each image consist of 640×480 pixels (*YUV 4:2:2*) and we applied a Pentium III 800 Mhz processor.

CMU Algorithm	**Our Algorithm**
37.47 s	22.67 s

Thus, our algorithm is significantly faster. Moreover, our algorithm also extracts the contour curves of all regions at each frame.

However, we do not claim that our algorithm performs better in all possible cases. There might be applications where the other algorithm performs better. This will typically be in cases, where many small and fast moving regions are to be tracked.

6 Conclusion

We have proposed a new method that is able to efficiently track and extract the boundaries of several homogeneous regions. The efficiency is accomplished by not processing the entire images but concentrating on parts where regions are expected. The algorithm exploits the fact that corresponding regions in successive images often overlap and it extends the region growing paradigm: Tracking is accomplished by alternating shrinking and growing. We provide a homepage for the algorithm, with source code available in *C++* and demo videos[1].

The algorithm is also very useful for edge extraction and tracking (also for other features). The advantage is that feature detectors can be very selective and that they have to be applied at a few locations only. For instance, when searching for edges, a detector which responds to edges having a predefined direction can be used. This is possible, because the boundary contour of each region and their corresponding normal directions are computed by our method.

We have also demonstrated the application of the algorithm in a practical problem. In our real-time vision system for RoboCup we use the algorithm to track regions like the goals, in order to locate and recognize these objects. However, we also use the method to detect the marking lines on the playing field, which are matched to a predefined model for the purpose of robot self-localization. What makes the algorithm practical is, that both, region and edge tracking, can be accomplished in the same run.

[1] http://page.inf.fu-berlin.de/˜hundelsh/research/RegionTracking/index.htm

The method also exposes a useful interface for higher level algorithms. For instance, visual attention can be implemented by controlling which regions should be tracked. Here the primary issue is that when excluding a region from tracking, the algorithm is really faster, since less pixels are accessed in the images. Another interface is the extracted boundary contour. Since the algorithm is based on region growing connected boundaries are guaranteed. The boundary can serve as a base for recognition of objects and movements or as a reference into the images. In RoboCup for instance, we track the regions of the green playing field and the boundary helps us to find other objects on the playing field. This is because all objects like the ball and other robots being on the playing field are next to the green regions in the images. Therefore their boundaries can be used as a reference into the image where to search for the objects and they can be rapidly detected. There is another advantage of the method, concerning the interface between vision and path planning. When using potential field methods for path planning [12], defining the field is an expensive operation. The occupancy grid for the tracked green field can be used as potential field. Obstacles must not be inserted, since they are not part the green region. However, the marking lines must be inserted to the grid to allow the robot to pass over them. When using omnidirectional vision, there is also the advantage that the resolution of the path planning will be high for near but rough for distant positions.

There are three open questions future work should concentrate on. The first is related to the homogeneity criterion. In this paper we have assumed a predefined criterion, such as certain color classes for instance. However, rigidly defining a homogeneity criterion will result in an inflexible algorithm. How can the homogeneity criterion be automatically defined? This is not just a question of finding some thresholds but also of the kind of criterion to be used (color, texture,...). Related to the homogeneity criterion is the question of scale [13]. How many pixels should constitute a drop? If, for instance, there was a region whose pixels had all the same color, then the algorithm could work on single pixels and the homogeneity criterion could be based on color difference. But, if texture is present, more than a single pixels has to serve as a unit. The last question concerns the correspondence of regions over time. Can the fact of overlap be exploited for correspondence definition?

References

1. Rolf Adams and Leanne Bischof. Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647, 1994.
2. Neils Gordon Arnaud Doucet, Nando de Freitas, editor. *Sequential Monte Carlo Methods in Practice.* Statistics for Engineering and Information Science. Springer, 2001. ISBN 0-387-95146-6.
3. A. Blake and M. Isard. *Active Contours.* Springer-Verlag, 1998.
4. Claude R. Brice and Claude L. Fennema. Scene analysis using regions. *Artificial Intelligence*, 1:205–226, 1970.
5. James Bruce, Tucker Balch, and Manuela Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of IROS-2000, Japan*, October 2000.

6. John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 8:679–698, 1986.
7. Laurent D. Cohen. On active contour models and balloons. *CVGIP:IU*, 1991.
8. G. W Cook and E J Delp. Multiresolution sequential edge linking. *Proceedings of the IEEE International Conference on Image Processing*, pages 41–44, October 1995.
9. S. A. Hojjatoleslami and J. Kittler. Region growing: a new approach. *IEEE Transactions of Image Processing*, 7(7):1079–1084, 1998.
10. P. V. C. Hough. Method and means for recognizing complex patterns. US Patent 3,069,654, December 1962.
11. P Jonker, J Caarls, and W Bokhove. Fast and accurate robot vision for vision based motion. *Lecture Notes in Computer Science*, 2019:149–155, 2001.
12. J Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991. ISBN 0-7923-9206-X.
13. T. Lindeberg. *Scale-Space Theory In Computer Vision*. Kluwer Academic Publishers, 1994.
14. Feng Lu and Evangelos Milios. Robot pose estimation in unknown environments by matching 2d range scans. *CVPR94*, pages 935–938, 1994.
15. S. Mallat and S. Zhong. Characterization of signals from multiscale edges. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(7):710–732, 1992.
16. R. Mehrotra, K. R. Namudura, and N. Ranganathan. Gabor filter-based edge detection. *Pattern Recognition*, 25:1479–1494, 1992.
17. Theo Pavlidis. Integrating region growing and edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3):225–233, 1990.
18. K. K. Pingle. Visual perception by computer. In A. Grasselli, editor, *Automatic Interpretation and Classification of Images*, pages 277–284. Academic Press, New York, 1969.
19. L. G. Roberts. Machine perception of three-dimensional solids. In J. T. Tippet et al., editor, *Optical and Electro-Optical Information Processing*, pages 159–197. MIT Press, Cambridge, 1965.
20. Mark A. Ruzon and Carlo Tomasi. Color edge detection with the compass operatro. *IEEE Conference on Computer Vision and Pattern Recognition*, 2:160–166, 1999.
21. A. Siebert. Dynamic region growing, 1997.
22. S. W. Zucker. Region growing: Childhood and adolescence. *Computer Graphics and Image Processing*, 5:382–399, 1976.