Region-Based Segmentation with Ambiguous Color Classes and 2-D Motion Compensation*

Thomas Röfer

Deutsches Forschungsinstitut für Künstliche Intelligenz GmbH,
Sichere Kognitive Systeme, Enrique-Schmidt-Str. 5, 28359 Bremen, Germany
E-Mail: Thomas.Roefer@dfki.de

Abstract. This paper presents a new approach for color segmentation, in which colors are not only mapped to unambiguous but also to ambiguous color classes. The ambiguous color classes are resolved based on their unambiguous neighbors in the image. In contrast to other approaches, the neighborhood is determined on the level of regions, not on the level of pixels. Thereby, large regions with ambiguous color classes can be resolved. The method is fast enough to run on a Sony AIBO in real time (30 Hz), leaving enough resources for the other tasks that have to be performed to play soccer. In addition, the paper discusses the problem of motion compensation, i. e. reversing the effects of a rolling shutter on the images taken by a moving camera.

1 Introduction

In most RoboCup leagues, cameras are the central sensor of the robots. As the environment is color-coded, image segmentation is one of the most important topics in soccer robot's image-processing systems. Especially in leagues with limited on-board computing power, such as the Four-Legged League and the Humanoid League, extremely fast and robust image segmentation algorithms are a necessity. A new approach to this problem is presented in Section 2. Also typical in these leagues is the use of inexpensive CMOS cameras, e. g., the Sony AIBO, the standard platform used in the Four-Legged League, is equipped with such a sensor. Section 3 deals with the central problem of this kind of sensor, the so-called rolling shutter.

2 Image Segmentation

2.1 Current Methods

There are two general image-processing approaches in RoboCup. The first one is the *blob-based* approach (e. g. CMVision [1]), the second one is the *grid-based*

* This work has been funded by the Deutsche Forschungsgemeinschaft in the context of the Schwerpunktprogramm 1125 (*Kooperierende Teams mobiler Roboter in dynamischen Umgebungen*).

approach (e. g. the vision system of the GermanTeam [2]). While blob-based approaches such as CMVision entirely rely on color segmentation, the vision system of the GermanTeam still does it for the most part. The general problem is that a color classification is hard to find in which, e. g., the ball is completely orange, but orange is detected nowhere else on the field (neither in yellow goals, nor in red uniforms). Hence, teams such as the GermanTeam or the NUbots [3] have started to use additional color classes such as *yellow-orange* or *red-orange* that delay the decision about which color the pixels actually have. Quinlan *et al.* [3] call these color classes *soft colors*. They resolve these soft colors after the blob formation based on relations between the bounding boxes of the blobs. The relations are object-specific. Palma-Amestoy *et al.* [4] present a more general approach for defining and resolving soft colors. From example images that are labeled manually with the unambiguous color classes that the objects in the images should have, a color table with soft colors is automatically generated. Images are segmented using the soft colors and afterwards a mode filter is applied that assigns the color class to each pixel that is the most frequent in its 3×3 neighborhood.

2.2 Ambiguous Color Tables

A color table is a mapping from image colors to color classes. Typically 256^3 possible colors exist in an image (one byte for each color channel; Y, Cr, and Cb in case of the AIBO), and they have to be reduced to only a few color classes. In case of the Four-Legged-League in 2007, these are the seven classes orange, yellow, sky-blue, red, blue, green, and white. An *ambiguous color class* is a color class that contains more than one of these base classes, e. g. yellow-orange or red-orange. Ambiguous color classes are represented as a bit-set, i. e. each bit of a byte stands for one of the color classes. A color table can be implemented as a simple lookup table, i. e. the color values function as indices into a 3-D array of color classes. Since a color table of 16 MB would be quite big, the table takes only the six highest bits of each color channel into account. Thus the size of the color table is $64^3 = 262144$ bytes. In contrast to the work described in [4], the color tables are created manually, using a color table editor.

2.3 Image Segmentation with Ambiguous Color Classes

In general, image segmentation with ambiguous color classes uses the same base algorithms as segmentation with unambiguous color classes [1], but it applies them several times with slight differences to resolve the ambiguities. Since the whole image is processed, no color correction is performed to eliminate the bluish corners of the camera images of the AIBO. Instead it is assumed that this problem is handled by the ambiguous color classification.

In a first step, all pixels are segmented determining their (ambiguous) color class from the color table. Successive pixels with the same color class are combined to *runs*. Each run contains its color class, its y-coordinate, and its first and last x-coordinate. For the image size of 208×160 pixels used in the AIBO, all

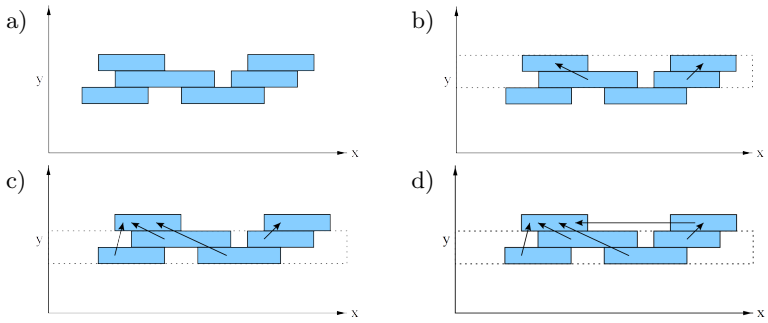

Fig. 1. Grouping runs to regions (taken from [1]). a) Runs start as a fully disjoint forest. b) Scanning adjacent lines, neighbors of the same color class are merged. c) New parent assignments are to the furthest parent. d) The first run in a region is always the parent of all others.

these values can be represented by bytes. Since the camera images of the AIBO are rather noisy when taken with the fastest shutter setting, the run length encoding typically reduces the image size only by factor 10, i. e. around 3000 runs will be generated. Please note that an ambiguous segmentation creates more runs than an unambiguous one, because there are simply more color classes.

The collection of runs is traversed five times. In the first phase, only runs of the same ambiguous color class are grouped together, because these regions are required for the next two phases. Runs with unambiguous color classes are simply skipped. Grouping runs to regions is based on the work of Bruce et al. [1], who describe it as a *union-find* problem that can be solved highly efficiently (cf. Fig. 1). The collection of runs is processed from top to bottom and from left to right. There are always two current runs in two adjacent rows. Whenever they overlap and have the same color class, they are grouped together. Groups are defined as a tree structure with each run having a pointer to its parent. During merging, path compression is applied, and it is ensured that the first run in a group is always the parent of all others.

Regions of ambiguous color classes inherit the unambiguous color class of which they are surrounded most. This extends the mode filter approach described in [4] to the level of regions. Therefore, a histogram has to be calculated for each region with an ambiguous color class on which unambiguous color class is neighbored how often, i. e. how many pixels on the region's perimeter are neighbored to which color class. Therefore, a second pass over the runs is performed. This time it is focused on the neighborhood between runs of unambiguous and ambiguous color classes, and for each such pair, the histogram of the ambiguous region involved is updated. In this pass, the neighborhood between runs in the same row has also to be considered.

After all the histograms have been calculated, the ambiguous color classes can be resolved. This is done in a third pass. Each ambiguous color is replaced by the unambiguous class that reached the highest score in the histogram of the

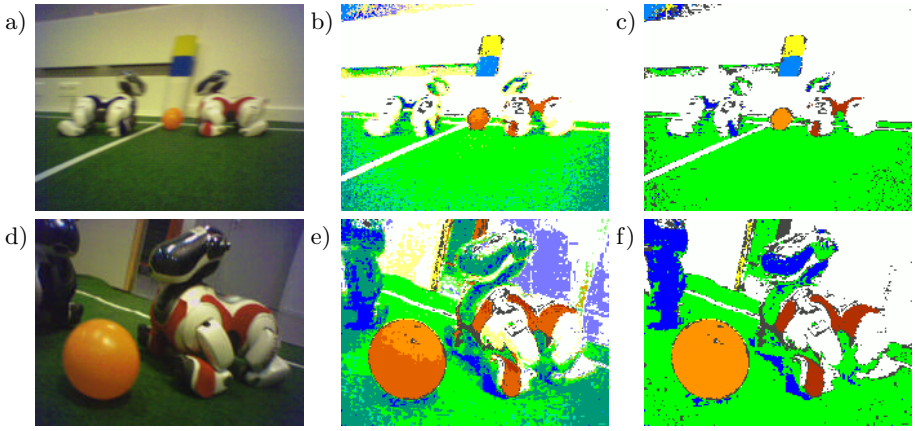

Fig. 2. Image segmentation with ambiguous color classes. a, d) Original camera images. b, e) Segmentation with ambiguous color classes. c, f) Resolution of ambiguous color classes.

region, but only if that score is above a threshold (e. g. more than two pixels). Regions that do not satisfy this criterion are deleted. Thus, ambiguous regions that are only surrounded by other ambiguous regions are removed. It would be possible to delay the resolution of the color class of such a region until its neighbors have unambiguous color classes, but this would take considerably more computing time, and therefore it was not implemented.

After the ambiguous color classes are resolved, a final merging pass over the runs is performed to group all regions. Again, some special treatment is required, because now there are successive runs within the same row that have the same color class. They are grouped together to single runs on the fly while merging the regions. In the final phase, the blobs are collected.

2.4 Results

Figure 2 shows some examples of segmented images. The first column shows two images as they are taken by the camera of the AIBO with the fastest shutter setting. The camera was turning quickly while the image shown in Figure 2a was taken, resulting in rather blurry colors. The column in the middle shows the result of the segmentation using an ambiguous color table. Please note that large parts of the ball and the uniform of the red robot are segmented as the ambiguous color class red-orange. Since the blue uniform is very dark, it was decided to use the same color class for black and blue. Hence, not every blue region on the field is an indication for a robot of the blue team, but this problem exists as long as the Four-Legged League, and cannot be solved on the level of color segmentation. However, as can be seen in the right column, other ambiguities can be resolved very well. The red uniforms are entirely red and the ball is always completely orange. The robots and the walls are white and the field is green, even in the

Module	Min	Max	Avg	Freq
AmbiguousRLEColorClassImage	1.0	9.0	2.7	30.0
RLEColorClassImage	2.0	15.0	6.6	30.0
groupAmbiguousRegions	0.0	7.0	0.9	30.0
calcPerimeterStats	0.0	6.0	2.6	30.0
resolveAmbiguousColors	0.0	4.0	1.3	30.0
groupRegions	0.0	4.0	1.1	30.0
Blobs	0.0	3.0	0.5	30.0

Fig. 3. Runtime measurements on an AIBO in ms

corners of the image. Figure 3 shows the runtime of the individual parts of the system.

3 Motion Compensation

The AIBO is equipped with a simple CMOS camera. Such cameras can also be found in PDAs such as the Siemens Pocket Loox 720 that is used by several teams in the Humanoid League, e. g. by the BredoBrothers [5]. Such cameras have a central weakness, the so-called rolling shutter. Instead of taking images at a certain point in time, a rolling shutter takes an image pixel by pixel, row by row. Thus the last pixel of an image is taken significantly later than the first one. The general problem is depicted in Figure 4. The AIBO is equipped with a camera in its head that takes 30 images per second. By moving its head, the AIBO can point the camera in different directions. Since an image is not taken all at once, the camera may point to a different direction when the first pixel is recorded than when the last pixel is taken. In Figure 4a the AIBO turns its head from right to left. Thus the head is pointing further right when the upper image part is taken and further left when the lower part is taken. This results in a distorted image as depicted in Figure 4b. In fact, the effect is not only present during panning the camera, but also when it is tilted or even rolled.

3.1 State of the Art

The problem of the rolling shutter was first mentioned by Nistico and Röfer in [6], and later analyzed in more detail by Nicklin *et al.* [7]. The approach of compensating the effect of the rolling shutter described by Nistico and Röfer is the most general one. However, it requires calculating the positions of all percepts twice, and in merging the pairs of locations or bearings a rather heuristic approach is followed. Nicklin *et al.* concentrate on the compensation of horizontal distortions resulting from panning the camera [7]. They determined that the camera of the AIBO actually takes the full 33.3 ms to take a picture, i. e. there is basically no delay between recording the last pixel of one image and the first pixel of the next one. The camera images and the joint angles on which the calculation of the camera position is based do not arrive at the same time, but they are time-stamped. As is indicated in the team report of the Microsoft Hellhounds [8], it seems that the timestamp of the joint angles matches the time when 3/4 of the image is recorded.

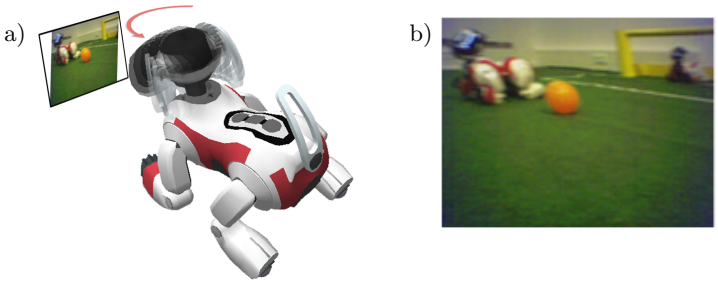

Fig. 4. Image distortion by a rolling shutter. a) Robot is turning its head while taking an image. b) Resulting camera image.

3.2 2-D Motion Compensation

Nicklin *et al.* [7] only compensate for horizontal distortions of the image. However, as can be seen in Figure 5b/e and c/f the rolling shutter shrinks the image if the head is turning downwards and stretches it when turning upwards. Because of the gravity, the head can be tilted faster downwards than upwards, resulting in the stronger effect shown in Figure 5b/e. The distortion in vertical direction impedes the calculation of distances to objects if the calculation is based on their position in the image, e. g. the distance to field lines can be determined this way. The distortion can add systematic errors to these calculations. For instance if the head is always turning upwards when looking to the right and turning downwards when looking to the left, as it is often done when scanning for the ball, distances to field lines are overestimated in one direction and underestimated in the other, resulting in a bad self-localization.

The motion of the camera between the previous image and the current image can be determined from the rotation matrices R_t and $R_{t-\Delta t}$ that describe the orientations of the camera as $\Delta R = R_t^{-1} R_{t-\Delta t}$. From ΔR the relative pan angle α and relative tilt angle β can be calculated and used when applying the equation from [7] to the 2-D case:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} cx - f \tan\left(\arctan\frac{cx-x}{f} - d\alpha\right) \\ cy + f \tan\left(\arctan\frac{y-cy}{f} - d\beta\right) \end{pmatrix} \quad \text{where} \quad d = \frac{\frac{y}{imageHeight} - 0.75}{\Delta t 30\text{Hz}} \tag{1}$$

(x', y') is the corrected position of the pixel (x, y). (cx, cy) is the image center and f the focal length of the camera. The decimal number 0.75 is used because the camera position was measured when 3/4 of the image were taken (cf. previous section). The factor d is only dependent on the y-coordinate, because the effect of the x-coordinate on the distortion of the image is far below a single pixel. The equation also works if Δt is more than the delay between two successive images, e. g., if an image was skipped.

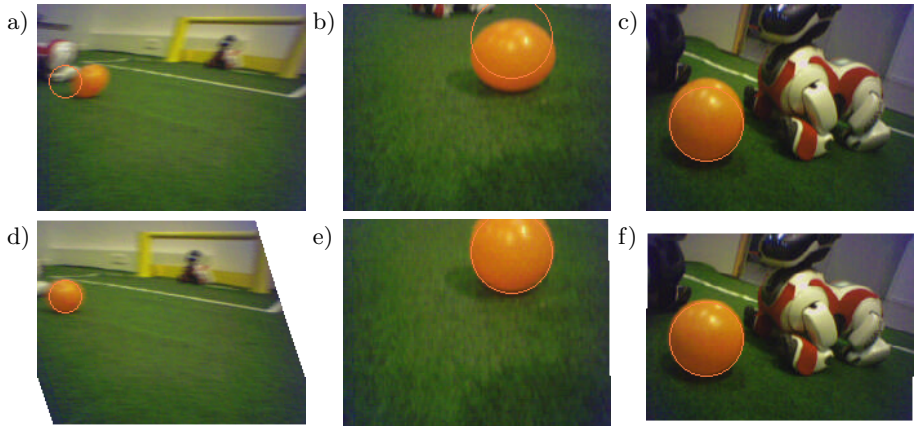

Fig. 5. Images taken will the camera was moving a) right, b) down, c) up. d–f) Corrected images. The orange circles represent the ball positions, also projected back to the original images.

3.3 Results

The simplest way to determine the improvements achieved by motion compensation is to let the AIBO turn its head quickly while the image-processing system determines the bearing to a static object in the environment. Here, as well as in [7] and [8], the static object is a ball. Figure 6 shows the results of the experiment using a ball detector based on the approach presented in Section 2, and the current head motion of the GermanTeam, which has a slight stop in the middle. The standard deviation of the bearings without motion compensation is 3.59° (without the stop it is 5.54°). When the ball position is determined from the distorted image using a Levenberg-Marquardt least-square fitting [9] and the position of its center is corrected afterwards, the standard deviation of the bearings is 1.81°. If the edge points of the ball are corrected before they are used in the fitting algorithm, the standard deviation is reduced to 1.76°. The influence of stopping or not stopping the head on these results was negligible.

4 Conclusions

This paper describes two methods to improve image-processing in RoboCup. The image segmentation using ambiguous color classes is a general solution to blob-based image-processing. It is fast and simplifies post-processing significantly, because many of the problems that result from noisy camera images are already dealt with during image segmentation. The section on motion compensation brings together all the findings of different teams on this topic and adds a compensation for up and down rotations of the camera. Thus the precision of bearings to objects could significantly be improved.

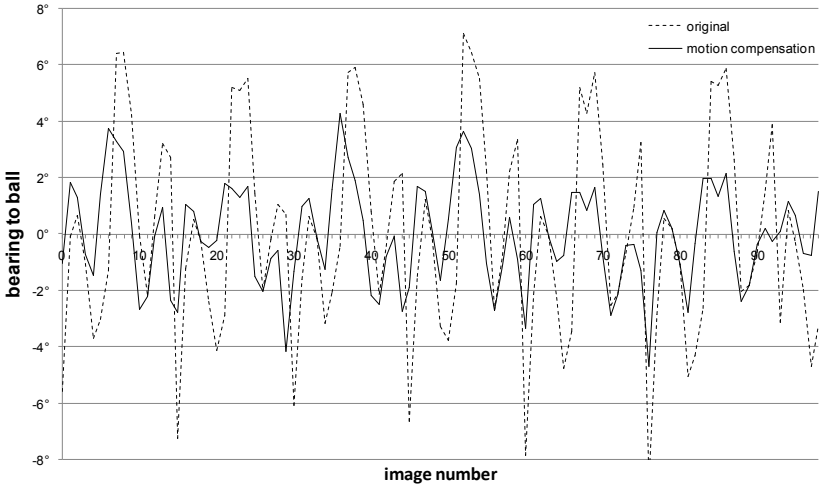

Fig. 6. Errors in horizontal bearings to the ball when turning the head with 300°/s.

References

1. Bruce, J., Balch, T., Veloso, M.: Fast and inexpensive color image segmentation for interactive robots. In: Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on. Volume 3. (2000)
2. Röfer, T., Jüngel, M.: Vision-based fast and reactive Monte-Carlo localization. In: IEEE International Conference on Robotics and Automation, Taipei, Taiwan, IEEE (2003) 856–861
3. Quinlan, M.J. et al.: The 2006 NUbots Team Report. http://robots.newcastle.edu.au/publications/NUbotFinalReport2006.pdf (2007)
4. Palma-Amestoy, R.A., Guerrero, P.A., Vallejos, P.A., del Solar, J.R.: Context – dependent color segmentation for Aibo robots. In: Proc. of the 3rd Latin American Robotics Symposium, IEEE, Robotics and Automation Society (2006)
5. Laue, T., Röfer, T.: Getting upright: Migrating concepts and software from four-legged to humanoid soccer robots. In Pagello, E., Zhou, C., Menegatti, E., eds.: Proceedings of the Workshop on Humanoid Soccer Robots in conjunction with the 2006 IEEE International Conference on Humanoid Robots. (2006)
6. Nistico, W., Röfer, T.: Improving percept reliability in the Sony Four-Legged League. In Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y., eds.: RoboCup 2005: Robot Soccer World Cup IX. Number 4020 in Lecture Notes in Artificial Intelligence, Springer (2006) 545–552
7. Nicklin, S.P., Fisher, R.D., Middleton, R.H.: Rolling shutter image compensation. In Lakemeyer, G., Sklar, E., Sorrenti, D., Takahashi, T., eds.: RoboCup 2006: Robot Soccer World Cup X. Lecture Notes in Artificial Intelligence, Springer (2007)
8. Hebbel, M. et al.: Microsoft Hellhounds Team Report 2006. http://www.microsoft-hellhounds.de/fileadmin/pub/MSH06TeamReport.pdf (2007)
9. Seysener, C.J., Murch, C.L., Middleton, R.H.: Extensions to object recognition in the four-legged league. In: RoboCup 2004: Robot World Cup VIII. Number 3276 in Lecture Notes in Artificial Intelligence, Springer (2005) 274–285