# Towards a Calibration-Free Robot: The ACT Algorithm for Automatic Online Color Training

Patrick Heinemann, Frank Sehnke, Felix Streichert, and Andreas Zell

Wilhelm-Schickard-Institute, Department of Computer Architecture,
University of Tübingen, Sand 1, 72076 Tübingen, Germany
{heinemann, sehnke, streiche, zell}@informatik.uni-tuebingen.de

**Abstract.** Many approaches for object detection based on color coding were published in the RoboCup domain. They are tuned to the typical RoboCup scenario of constant lighting using a static subdivision of the color space. However, such algorithms will soon be of limited use, when playing under changing and finally natural lighting. This paper presents an algorithm for automatic color training, which is able to robustly adapt to different lighting situations online. Using the ACT algorithm a robot is able to play a RoboCup match while the illumination of the field varies.

## 1 Introduction

The extraction of landmarks and objects from a camera image is a crucial task for robots. In RoboCup the color of such features is sufficient for their extraction. Although there were attempts to detect objects - mainly the ball - using form or texture [4],[12], the majority of the RoboCup teams uses algorithms exploiting the special colors to reduce the computational load. Most of these algorithms utilize a predefined subdivision of the three-dimensional color space into several color classes. In the Middle-Size-League (MSL) there are 6 color classes corresponding to different objects. If the image pixels are transformed into these color classes the extraction of the different objects can be very efficient [6]. The approaches mainly differ concerning the subdivision of the color space. Bruce *et al.* use a rectangular subdivision with a minimum and maximum threshold for each color class [2]. Others, such as Bandlow *et al.* use an arbitrary shape in the two color dimensions of the YUV color space and two thresholds in the intensity channel in order to achieve some independency of the lighting conditions [1].

However, with changing lighting, the colors change their representation in all three dimensions of the color space (cf. [10]). Therefore, our team uses a color look-up table that maps each color to its corresponding class. With such a table it is possible to model any subdivision of the color space. Nevertheless, even this approach will fail if the amount and the speed of changes in lighting conditions rises when playing at natural light. In addition, the time needed to manually train a color look-up table was still up to 5 minutes per robot which is too much if we want to be able to extremely reduce the setup time for the teams. Thus, there is clearly a need for a fast and automatic training of a look-up table that dynamically maps the colors to different color classes with changing lighting.

Related work on this topic includes the semi-automatic self-calibration system for MSL robots presented in [9], that subdivides the color space into circular clusters. However, the mapping of the clusters to their corresponding color class has to be decided by a human supervisor. Other methods from the Sony four-legged league require special movements of the camera to train the color look-up table [3] or simply train a color mapping for only three different illumination scenarios [11]. A very promising method is presented in [8]. This method retrains a look-up table without an a-priori known subdivision of the color space. However, this method would require too much computation time when applied to a $580 \times 580$ pixel image that is used on our MSL robots.

In this paper we present a new algorithm to automatically train such a table for a RoboCup robot, using only knowledge of the field geometry. By incorporating the pose of the robot computed by our self-localization [5], this algorithm is able to constantly retrain the mapping in conditions where the lighting changes. By keeping the amount of training per cycle as low as possible, the algorithm can be processed 50 times a second on our RoboCup MSL robots, while being capable of adapting to sudden changes in illumination in only a few cycles.

The remainder of the paper is organized as follows: the proposed algorithm is presented in detail in the following section. Section 3 emphasizes the robustness of the algorithm concerning changes in lighting by presenting experimental results, while the last section concludes this paper.

## 2 The Automatic Color Training Algorithm

The main idea of the automatic color training (ACT) algorithm is to automatically train a color look-up table, using the pose of the robot from the self-localization and a model of its environment to compute the expected color class for the image pixels. For this, the a priory known field parameters are used and a mapping from pixel to two-dimensional world coordinates is trained, using the field markings and a predefined pose of the robot [7].

However, the use of self-localization for the automatic color training results in a mutual dependency. Two features are used to overcome this mutual dependency. First, the image that is used for training the mapping from pixel to world coordinates is used for the training of an initial color look-up table. Second, the extraction of the green and the white color class is robust enough to cope with a sudden change in illumination as shown in section 2.1. This enables the self-localization to keep track of the robot until the other classes are adapted.

With the color values of the pixels and the expected color class a look-up table is trained. As there will obviously be errors in the expected color class, ACT tracks clusters of the color classes in the color space with a mean value and standard deviation, to filter out such errors. Only colors of pixels that fit into a sphere centered in the mean value of a color class with radius equal to a multiple of the standard deviation, are added to the look-up table, while colors of pixels that correspond to coordinates outside of the playable field are removed.

## 2.1 Computation of the expected color class

Given the pose of the robot and the mapping from pixel to world coordinates, the algorithm can easily compute, which part of the field should correspond to a given pixel. Every pixel that corresponds to coordinates inside of the field is either classified as white field line or green floor, according to the field model. As green is the predominant color in the image in RoboCup, green pixels are usually classified as green, even if the pose estimation from the self-localization is not very accurate. White, however, is very rare in the image but has the main influence on the landmark-based self-localization. Therefore, a special treatment is used for pixels that are mapped to the white class. Only those pixels that have a higher intensity than their surrounding are ultimately used to train white. Given the intensity of a pixel $I(p_{x,y})$ at position $(x, y)$ this filter is defined as

$$I(p_{x,y}) > \frac{1}{25} \sum_{i=x-2}^{x+2} \sum_{j=y-2}^{y+2} I(p_{i,j}). \tag{1}$$

Objects that extend into the third dimension cannot be mapped correctly. However, the region of the image that displays the goal can be defined, depending on the camera system. Only pixels inside this area are mapped to yellow or blue. To train black, the algorithm uses pixels that correspond to the chassis of the own robot. The ball color, though, is a problem for a calibration-free algorithm, as the ball is not static and there is no way of training the ball color without some previous knowledge about the color or position. All pixels that are mapped to a position outside of the field are assigned to the special color class *unknown*.

## 2.2 Adaptation of the cluster for each color class

For each color class $k = 1 \ldots 6$, ACT tracks a cluster in the color space with a mean value $\mu_k$ and a standard deviation $\sigma_k$ resulting from the color values of the previous cycles. For color values $c = (u, v, w) \in \{0, C_{max}\}^3$, the parameters of the different clusters are initialized as

$$\mu_{k,0} = \frac{1}{2}\left(C_{max}, C_{max}, C_{max}\right) \tag{2}$$

$$\sigma_{k,0} = \frac{\sqrt{3}}{2} C_{max}. \tag{3}$$

Given the set of colors $X_{k,t} = c_1, \ldots, c_m$ of all pixels expected to belong to color class $k$ at cycle $t$, these parameters are updated as

$$\mu_{k,t} = \frac{1}{\eta + 1}\left(\eta\, \mu_{k,t-1} + \frac{1}{m}\sum_{i=1}^{m} c_i\right) \tag{4}$$

$$\sigma_{k,t} = \frac{1}{\eta + 1}\left(\eta\, \sigma_{k,t-1} + \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(c_i - \mu_{k,t})^2}\right). \tag{5}$$

To save computation time, the algorithm uses only every 400th pixel, starting at a random pixel. The choice of $\eta$ determines the responsiveness of the color look-up table update. A value of $\eta = 4$ was empirically determined as optimal, enabling the algorithm to extremely reduce the number of examined pixels. In order to avoid the cluster from collapsing, a lower bound of the standard deviation $\sigma_{min}$ is introduced. As it is possible that the cluster is too small to include the new color values after a change of illumination, the standard deviation is then doubled to increase the size of the cluster until it includes these color values.

### 2.3 Add colors to the color look-up table

To find out which colors are finally mapped to the classes, again a subset of every 400th pixel is selected. After the calculation of the expected color class, each color value is compared to the mean value of the corresponding color class. Given a color value $c$ that is computed to belong to color class $k$, the mapping from $c$ to $k$ is only added into the look-up table if $\|\mu_k - c\| < \zeta\,\sigma_k,$ with $\zeta > 1$, and $\|\cdot\|$ being the Euclidian norm and $\zeta$ being a threshold controlling the ratio between higher adaptability of the color look-up table and a higher false positive rate. The influence of $\zeta$ is investigated through experiments in section 3.1.

### 2.4 Remove colors from the color look-up table

After the addition of mappings to the look-up table, colors that are mapped to the special *unknown* class are removed from the table. To process a large number of pixels outside of the field every 20th pixel is used to completely remove unwanted color mappings. A color value $c$ that is expected to belong to the *unknown* class in this cycle but that was previously mapped to color class $k$ is removed, if $\|\mu_k - c\| > \xi\,\sigma_k,$ with $\xi > 1$, and $\xi$ being a threshold controlling the ratio between a lower false positive rate and lower true positive rate. The influence of this threshold is also investigated in section 3.1.

## 3 Results

### 3.1 Influence of the thresholds

To analyse the influence of the thresholds $\zeta\,\sigma_k$ and $\xi\,\sigma_k$ for adding and removing color mappings from the color look-up tables the algorithm was tested on three images with different brightness. For each image, several runs of the algorithm were started with different values for $\zeta$ using the appropriate pose estimation. After a few cycles the color look-up table converged to a stable state in each run and the quality or fitness of the resulting color look-up table was computed as the sum of the fitness of the $k$ color classes

$$f = \sum_k \left(1 - \frac{TP_k}{TP_k + FN_k}\right), \tag{6}$$

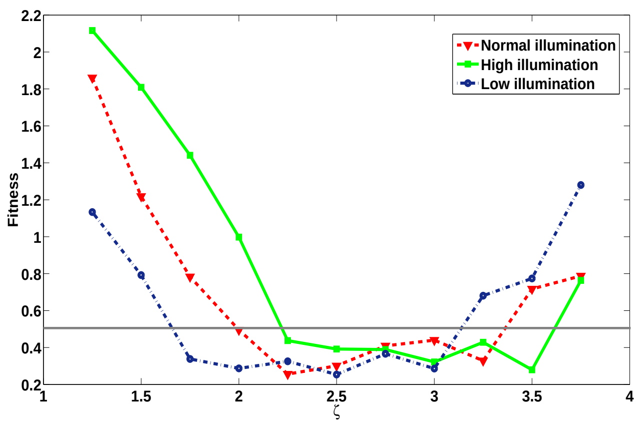

**Fig. 1.** Fitness of the color mapping of the resulting color look-up tables when ACT is applied to the test images.

where $TP_k$ is the number of true positives for class $k$ and $FN_k$ is the number of false negatives for class $k$. The fitness values for all runs are shown in figure 1. For the dark image, the algorithm converges to the best fitness for values $\zeta = [1.75, 3.0]$, while the interval of best fitness is $\zeta = [2.25, 3.25]$ and $\zeta = [2.25, 3.5]$ for the normal and the brightened image, respectively. On the one hand, lower values of $\zeta$ force the color classes to converge to a very small part of the color space that might not include all color values needed to correctly classify all pixels. If the image is very dark, however, the differences between the color classes and the deviation of the colors from the common mean per class are very low. Therefore good results can be achieved with lower values of $\zeta$ in the dark image. On the other hand, higher values of $\zeta$ enable the color class to spread out in the color space, resulting in a high standard deviation. This includes many colors that should not be mapped to this color class. Fortunately, there is a broad range of values $\zeta = [2.25, 3.0]$ for which all three images are classified with a very high fitness. For all subsequent experiments, a value of $\zeta = 2.5$ was used. The experiments done with the three test images to test the influence of $\xi$ show that different values of $\xi$ result in similar classification results for all three images. In fact, the selection of $\xi$ for a good quality of the algorithm is depending far more on the selection of $\zeta$. With $\zeta = 2.5$ the best results were achieved with a value of $\xi = [1.2, 1.5]$. For lower values of $\xi$, too many colors are removed from the table, while for higher values of $\xi$, too many colors from outside of the field remain in the table. For all subsequent experiments, a value of $\xi = 1.35$ was used.

## 3.2 Automatic color training on a static robot

This experiment demonstrates that a robot using the ACT algorithm is able to cope with sudden changes of lighting. First, the image on the left of figure 2 is

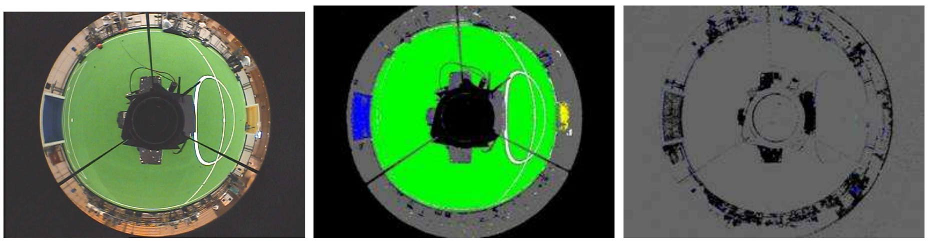

**Fig. 2.** The image in the middle shows the classification results using the look-up table trained by ACT on the left image. Using the same table to classify a darkened version of this image results in the classification shown on the right. Clearly, a robot with a static color mapping would have no chance of playing using such a classification.

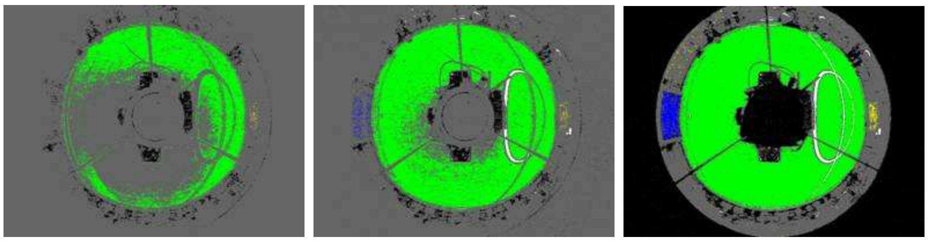

**Fig. 3.** The darkened image classified after 1, 2, and 8 steps of retraining the color look-up table of figure 2 using the ACT algorithm.

used for training a color look-up table. The classified version of this image using the table trained by ACT is shown in the middle in figure 2. Then, to simulate such a sudden change in lighting, the same look-up table was used to classify a darkened version of this image generated by reducing the brightness and contrast to 50%. resulting in the classifaction shown on the right side in figure 2. Clearly, a robot with a static color lok-up table would have no chance of playing using such a classification. With the ACT algorithm, however, the color look-up table is adapted to a stable optimum in only 12 cycles, which would result in only 240ms without color classification. In addition, the look-up table is already very close to the optimum after 2 cycles, at least for the important color classes green and white. Figure 3 shows the classified image after 1, 2, and 8 cycles of adaption.

### 3.3 Automatic online color training on a moving robot

As the presentation of results from a moving robot is very difficult, one experiment was carried out to show that the algorithm embedded in the rest of the robot control system including the self-localization is able to handle a completely wrong pose estimation. For that, three different images were consecutively fed into the ACT algorithm, two from a known pose of the robot and one from a

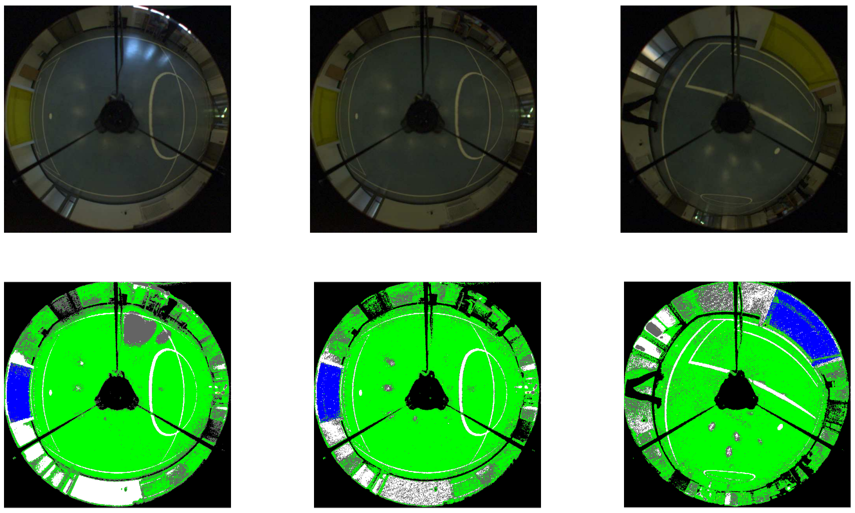

**Fig. 4.** After a few cycles of adaptation the ACT algorithm is able to train a very good color look-up table in all three situations, even with a completely wrong pose estimation (right).

pose where the robot was located 2.5m away from the old pose and rotated by 165 degrees (cf. figure 4, top row). While the self-localization uses the trained color look-up table for the pose estimation, the ACT algorithm always used the known pose as estimation. The classification results are shown in figure 4, bottom row. ACT is able to keep a very good color look-up table, even if the pose used for retraining the table is completely wrong. As the quality of the look-up table adapted by ACT is hardly depending on the quality of the pose estimation, the mutal dependency of the self-localization and the color training is no problem when using the ACT algorithm.

In order to use the ACT algorithm for online training on a moving, soccer playing robot running other processes like self-localization, the time needed to compute one cycle of the ACT algorithm has to be very low. Fortunately, in all the experiments presented in this paper, the computation time was below 4ms for one cycle on an Athlon XP 2400+ with 2GHz. This fits into the main cycle time on our RoboCup MSL robots that are capable of running all processes needed to control the robot in a 20ms cycle on their Pentium-M 2GHz computer.

## 4 Conclusions

This paper presents an algorithm for automatic online training of a look-up table that maps the colors of a three-dimensional color space onto different color classes used for the detection of objects and landmarks in camera images. For that, the ACT algorithm incorporates knowledge about its environment to compute which colors correspond to which color class. ACT consecutively adapts the look-up table to changing lighting situations resulting in a robust classification of the image pixels. The presented results show that the main parameters of the algorithm can be chosen in a way to produce good results over a large variety of lighting scenarios. Finally, the algorithm was implemented on a RoboCup MSL robot for online training of the color look-up table. Here, the mutual dependency of the color training and the self-localization is shown to have very little impact on the robustness of the algorithm, as the color training is very stable, even for a completely wrong pose estimation. With a cycle time of only 4ms the ACT algorithm was easily embedded into the control system of our RoboCup robots with a main cycle time of 20ms.

## References

1. T. Bandlow, M. Klupsch, R. Hanek, and T. Schmitt. Fast Image Segmentation, Object Recognition and Localization in a RoboCup Scenario. In *3. RoboCup Workshop, IJCAI'99*, 1999.
2. J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proc. 2000 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, volume 3, pages 2061–2066, 2000.
3. D. Cameron and N. Barnes. Knowledge-Based Autonomous Dynamic Colour Calibration. In *RoboCup-2003: Robot Soccer World Cup VII*, volume 3020 of *LNCS*, pages 226–237. Springer, 2004.
4. R. Hanek, T. Schmitt, S. Buck, and M. Beetz. Towards RoboCup without Color Labeling. In *RoboCup 2002: Robot Soccer World Cup VI*, volume 2752 of *LNCS*, pages 426–434. Springer, 2003.
5. P. Heinemann, J. Haase, and A. Zell. A Novel Approach to Efficient Monte-Carlo Localization in RoboCup. In *RoboCup 2006: Robot Soccer World Cup X*. Springer, 2006.
6. P. Heinemann, T. Rückstieß, and A. Zell. Fast and Accurate Environment Modelling using Omnidirectional Vision. In *Dynamic Perception 2004*. Infix, 2004.
7. P. Heinemann, F. Sehnke, F. Streichert, and A. Zell. Automatic Calibration of Camera to World Mapping in RoboCup uging Evolutionary Algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2006)*, 2006.
8. M. Jüngel. Using Layered Color Precision for a Self-Calibrating Vision System. In *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *LNCS*, pages 209–220. Springer, 2005.
9. G. Mayer, H. Utz, and G. Kraetzschmar. Towards autonomous vision self-calibration for soccer robots. In *Proc. 2002 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2002.
10. G. Mayer, H. Utz, and G. Kraetzschmar. Playing Robot Soccer under Natural Light: A Case Study. In *RoboCup-2003: Robot Soccer World Cup VII*, volume 3020 of *LNCS*, pages 238–249. Springer, 2004.
11. M. Sridharan and P. Stone. Towards Illumination Invariance in the Legged League. In *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *LNCS*, pages 196–208. Springer, 2005.
12. A. Treptow, A. Masselli, and A. Zell. Real-Time Object Tracking for Soccer-Robots without Color Information. In *European Conference on Mobile Robotics (ECMR 2003)*, pages 33–38, 2003.