

# Robust Pixel Classification for RoboCup

Andreas Fürtig, Holger Friedrich, Rudolf Mester

Visual Sensorics and Information Processing Lab

J.W. Goethe-Universität Frankfurt am Main

Robert-Mayer-Str. 10, D-60325 Frankfurt am Main

eMail: {fuertig,friedrich,mester}@vsi.cs.uni-frankfurt.de

URL: <http://www.vsi.cs.uni-frankfurt.de>

RoboCup is a challenging environment for vision algorithms, requiring real-time data processing on very constrained hardware. Although objects have defined colors, a fact that greatly simplifies the recognition process, every year the setup is chosen more and more demanding. The optimal and efficient handling of illumination changes is currently the most challenging problem to solve. In this paper we present a statistical approach towards robust color-based pixel classification building on models for color distributions. A fast implementation of our method, using lookup tables, allows the proposed algorithm to run on the hardware mandatory in the Standard Platform League of RoboCup.

## 1 Introduction

Color-based image interpretation is a central task in the RoboCup robot soccer competitions. In the '*Standard Platform League*' of RoboCup, all important objects can be discriminated by colors: blue and yellow goals, an orange ball, a green field and white field lines. The robots from different teams are distinguished by pink and light blue bands around their hip. In spite of this color coding rules, implementing a reliable pixel classifier remains a challenge, due to the reduced computational power available during the competition. In fact, participant teams are constrained to use the standard hardware platform, the humanoid robot *Nao* from *Aldebaran Robotics* [1]. The limited hardware prevents computationally demanding algorithms to be applied.

Since any further behaviour of the robot is based on a precise recognition of all relevant objects, a robust classification is essential: the robot needs to find the ball, walk to it, and then has to decide to which goal the ball must be kicked to avoid an own goal.

We propose a purely color-based robust pixel classification, which is well-suited to the hardware constraints found in the RoboCup competition. We do not exploit neighborhood relations in favor of a computationally tractable solution on the standard RoboCup

hardware. For color based pixel labeling, especially illumination changes constitute a problem, particularly in the face of RoboCup rules focusing more and more on real-world light conditions.

The native color space of the camera is YUV, which divides the signal into one luminance and two chrominance components. This allows to disregard minor changes in illumination, but the standard classifiers (based on the distance to a given fixed color value) fail in presence of significant changes, requiring manual tuning of color tables, etc.

In contrast to common algorithms applied in the RoboCup environment, we use first and second order statistics (mean vectors and covariance matrices) to describe the class-specific *distribution* of colors. This results in a statistically well-founded distance measure between colors, which in turn is used to decide whether a pixel belongs to a given object class.

The statistical distribution of color vectors corresponding to the individual semantic classes is currently obtained using a supervised learning step, based on the manual labeling of a very large set of training images, containing different illumination conditions.

To speed up computation during the game, training of the classifier is done offline, providing a lookup table for fast online classification on the robot.



Figure 1: RoboCup 2010 game setup. Objects are color coded, but illumination conditions may differ. Background is varying.

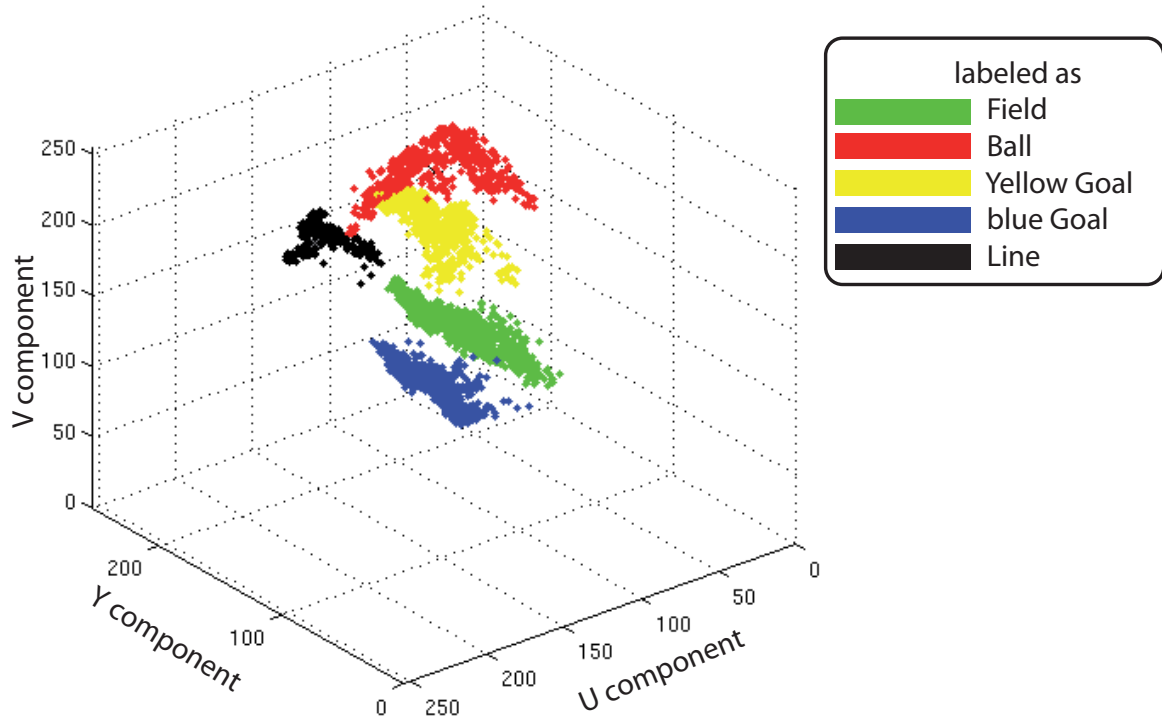


Figure 2: Distribution of a small set (about 5000) labeled pixels from different images obtained from the built-in camera of one robot (drawn in the YUV color space).

## 2 State of the Art

For years, classical *thresholding* techniques [11] dominated pixel classification in RoboCup. These algorithms worked well under constant illumination, but required manual tuning of thresholds each time the lighting conditions changed [3, 2]. In more recent work, advanced methods have been proposed, for example the concept of *soft colors* [9, 7, 10, 6]. It allows a pixel to belong to more than one class, which is beneficial for colors of objects which are ambiguous under extreme lighting conditions. In the typical RoboCup setting, this may apply to pixels belonging to the ball and pixels from the yellow goal.

The approach closest to us is by Sridharan and Stone [13]. It is about learning of a color models based on Gaussians and has also been applied successfully to the RoboCup setup.

## 3 Second Order Image Statistics

A sample set of input images typically provides thousands of labeled pixels. Each class typically forms a cloud of points in a three dimensional YUV space, which can be described by mean and covariance. See Fig. 1 for a description of the setup and Fig. 2 for a sample distribution of labeled pixels.

**Definitions:** Let  $\vec{x}$  be the spatial image coordinates, and  $\vec{\mathcal{I}}(\vec{x})$  the according color vector in the YUV space:

$$\vec{\mathcal{I}}(\vec{x}) = \left\{ (y, u, v)^T \mid y, u, v \in \{0, 255\} \right\}. \quad (1)$$

We define a set of labels  $\omega_k$ , which will represent the color-coded objects of the RoboCup scenario, namely five different labels  $\omega_1, \dots, \omega_5$  denoting the blue or yellow goal, lines, field or ball.

The training set contains manually labeled pixels  $i$

$$\mathcal{S} = \left\{ \left( \vec{\mathcal{I}}(\vec{x}_i), \omega_j \right) \right\} \quad (2)$$

from a set of images, representing different illumination conditions and game situations. Let  $\mathcal{S}(k)$  be the set of all color vectors labeled  $\omega_k$ . Now, we can estimate the mean  $\vec{\mu}_k \in \mathbb{R}^3$  and the covariance  $\mathbf{C}_k \in \mathbb{R}^9$  [8] of the intensity vectors  $\vec{\mathcal{I}}(\vec{x}_i)$  contained in  $\omega_k$ .

**Classification Task:** Our goal is to label every pixel in an image by designing a classifier based on the information collected in the training set. We do not want to adapt to one given image obtained under one illumination condition, but want to have good classification results for a broad range of situations. This comes at the price of performing worse for a given lighting condition compared to an algorithm tuned exactly for this setup. But it comes with the benefit of being able to handle a range of situations without tuning any parameters.

For our statistical approach we need a distance measure between a given pixel color and each class  $k$ . Let us have a look at Fig. 2. Empirically, the shape of the point cloud of pixels indicates how the distance should be computed: Most of the clouds form an ellipsoid, and moving along the longer main axis should be cheap because it is likely to stay within one class, whereas moving perpendicular to that axis should be more expensive because it is more likely to leave the class. The weighting described here is typically achieved by applying the principal component analysis (PCA), also known as Karhunen–Loève transform (KLT) [12], to identify the main directions of information in the data. As a result of the PCA, we obtain the Mahalanobis distance [4] between  $\vec{\mathcal{I}}(\vec{x}_i)$  and the class  $\mathcal{S}(k)$ , given by

$$d_k(\vec{\mathcal{I}}) = (\vec{\mathcal{I}} - \vec{\mu}_k)^T \mathbf{C}_k^{-1} (\vec{\mathcal{I}} - \vec{\mu}_k). \quad (3)$$

The Mahalanobis distance provides a reasonable distance measure which takes the shape of the distribution into account. To compute the distance between two color vectors, the inverse covariance matrix is used to compensate for the shape of the distribution. The decision whether a pixel belongs to the class is performed by thresholding, based on the weighted distances.

Using the distance measure  $d_k(\vec{\mathcal{I}}(\vec{x}_i))$ , the label is chosen as the one providing the smallest distance  $d_k$ . The labeling function  $\ell$  is defined as

$$\ell(\vec{\mathcal{I}}) = \underset{k}{\operatorname{argmin}} d_k(\vec{\mathcal{I}}). \quad (4)$$

Since we do not have a rejection class yet, we introduce the label  $\omega_0$  for pixel colors, which do not belong to one of our defined classes. We use a single threshold  $T$  to indicate whether a pixel color is rejected or not.

$$\hat{\ell}(\vec{\mathcal{I}}) = \begin{cases} \ell(\vec{\mathcal{I}}) & \text{if } d_{\ell(\vec{\mathcal{I}})}(\vec{\mathcal{I}}) \leq T \\ \omega_0 & \text{else} \end{cases} \quad (5)$$

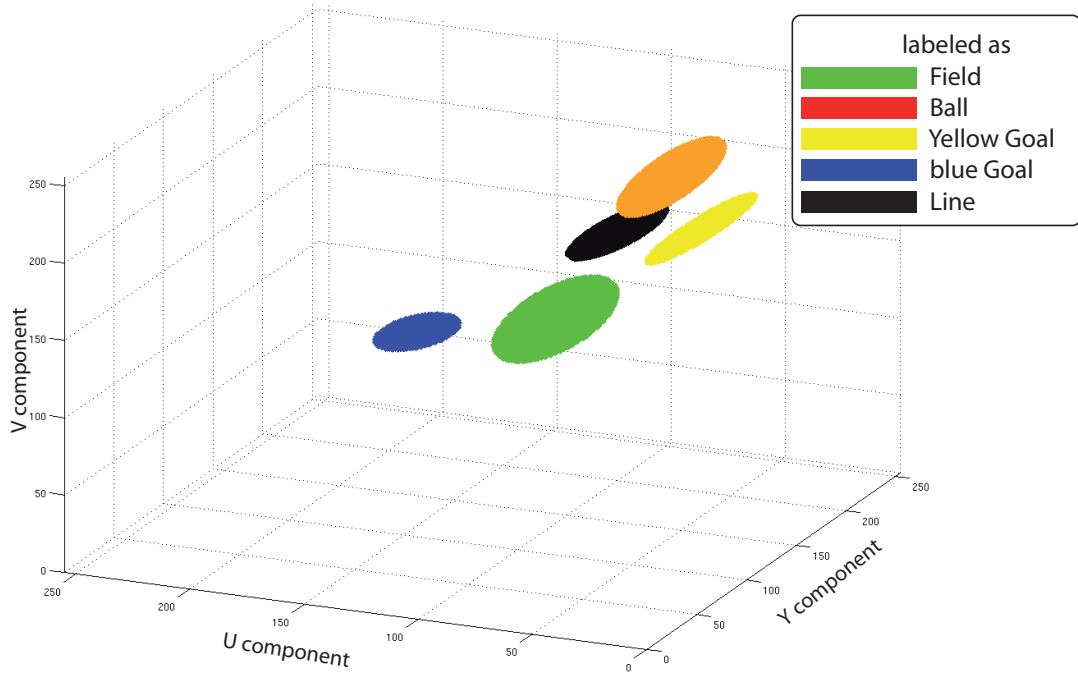


Figure 3: Different classes in the three dimensional YUV color space: Ellipses depict mean and covariance of the color distribution. The classes shown here have different distances to other classes. This can be exploited to get even better classification results on previously unlabeled pixels.

**Further Improvements:** The algorithm described before takes only the distance of a pixel color to all given classes into account (using a reasonable weighting), but disregards the relationship *between* different classes. Fig. 3 depicts a set of ellipses visualizing mean and covariance of the colors for different classes. The distance between these classes differs largely, a fact which can be exploited to improve the classifier. Previously unlabeled pixels may be added by enlarging ellipses if the separation to other classes is still retained, while other ellipses may be scaled down to improve separation between classes. This improves the results for unexpected extreme lighting conditions, but comes at the risk to label pixels which obviously do not fit into any class at all. Finally, we replace the common threshold by per class thresholds  $T_k$ : Let  $\mathcal{D}(\vec{\mathcal{I}}) = \{i | d_i(\vec{\mathcal{I}}) < T_i\}$ .

$$\tilde{\ell}(\vec{\mathcal{I}}) = \begin{cases} \operatorname{argmin}_{k \in \mathcal{D}(\vec{\mathcal{I}})} d_k(\vec{\mathcal{I}}) & \text{if } \mathcal{D}(\vec{\mathcal{I}}) \neq \emptyset \\ \omega_0 & \text{else} \end{cases} \quad (6)$$

Choosing different values gives us a better handling of unlabeled pixels and allows us to tune the classification process using our GUI shown in Fig. 4.

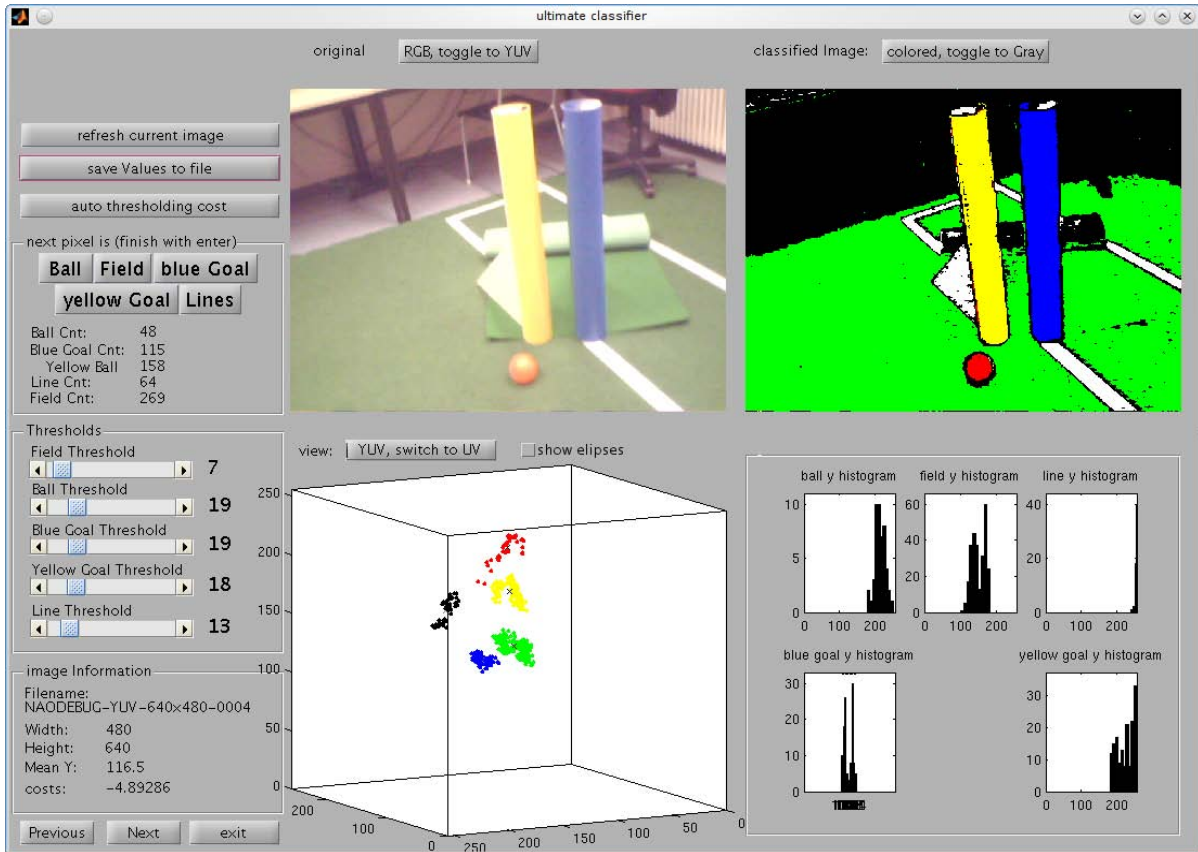


Figure 4: Graphical user interface of the classifier. The software is written in Matlab and available from our web page [5].

## 4 Threshold Learning

Given a large set of pixel-to-class correspondences (our training set  $\mathcal{S}$ ), the decision thresholds  $T_k$  for each class are chosen such that a cost functional  $\zeta$  is minimized (i. e. the number of false labeled pixels is minimized).

$$\zeta(\mathcal{S}, \vec{T}) = \sum_i \phi \left( \underbrace{\mathcal{S}_i}_{\left( \vec{\mathcal{I}}(\vec{x}_i), \omega_j \right)} \right), \quad \text{with } \vec{T} = (T_1 \dots T_5)^T \text{ and} \quad (7)$$

$$\phi \left( \vec{\mathcal{I}}(\vec{x}_i), \omega_j \right) = \begin{cases} -1 & \text{if } \tilde{\ell}(\vec{\mathcal{I}}) = \omega_j \quad (\text{matches ground truth}) \\ 5 & \text{if } \tilde{\ell}(\vec{\mathcal{I}}) = \omega_0 \\ 10 & \text{if } \tilde{\ell}(\vec{\mathcal{I}}) \neq \omega_j \end{cases} \quad (8)$$

We distinguish between three different cases: The computed class is correct, the pixel has not been labeled at all, or got a wrong label. An exhaustive search among all reasonable thresholds  $\vec{T}$  minimizes the cost function  $\zeta$ ; for a larger set of classes one may need a coarse to fine search to cope with the combinatoric explosion.

## 5 Results

Based on our training set  $\mathcal{S}$ , the classifier estimates the parameters  $\vec{\mu}_k$  and  $\mathbf{C}_k$  for each class. Decision thresholds  $T_k$  are obtained as described in Sec. 4. To validate our results, we use another test set, which was created in the same way like the training set, but with different images.

The qualitative analysis gives as a very good result like the one shown in Fig. 5. Severe illumination changes can be handled without having too much impact on the performance of the classifier. Using a single threshold for all classes as described first leads to good results (cf. middle column of Fig. 5). However, results can be improved with a threshold for every class, which can be seen especially on the ball surface or on the edges of the goals (cf. right column of Fig. 5).

To measure the quality of our classifier, we also applied a quantitative analysis. The classifier is run with the covariances, means and thresholds learned from our training set. As input the tuples (color value, label pairs) gathered from the test set were used. The results were combined with the ground truth and conduct us to a accuracy of nearly 97% with a single threshold used for all classes. Using a threshold for every class improved the accuracy to a quality of 99% correctly labeled pixels. However, this analysis relies only on a subset of all pixels that are contained in the images, exactly the which were labeled by hand. We do not have fully segmented and labeled pictures yet, which then would contain a class of unrelated, unlabeled pixels (e. g. from obstacles or from the audience). Hence, these numbers should be handled with care since the topic is still lacking further research.

Further work could be to combine our work with the 'soft colors' approach [9], to get an even better classification of ambiguous colors (which can clearly be seen at the border of the pole in the middle row of Fig. 5).

## 6 Real-Time Implementation for Nao Robots

As said before, the robots of the Standard Platform League come with very constrained hardware: AMD Geode LX800 CPU, 500 MHz, 256 MB RAM. This is standard x86 hardware, but compared to current hardware it is slow in floating point computations. A straightforward implementation, thresholding pixels colors against given classes and considering inverse covariance matrices, will cause a lot of multiplications and cannot be done on the Nao in realtime. Even a fixed point representation of the values is tricky, requiring a broad range of numbers.

The classifier result is only based on the pixel value itself. Since we do not need any other information, we can speedup the classification process by storing the classification result for all possible colors in a three dimensional look up table  $\mathcal{L}$  in the memory of the robot.

$$\mathcal{L} : \{0..255\}^3 \rightarrow \{0..5\}. \quad (9)$$

Eq. 1 indicates the required range for a component of the YUV color value is between 0 and 255, so we need an array which consists of  $256^3$  entries. The trivial implementation requires one byte per color, resulting in a lookup table of about 16 MB. Accessing the table is much faster than doing all the floating point computations (in practice more than 20 times), even though each access will typically cause a cache miss (the caches are very

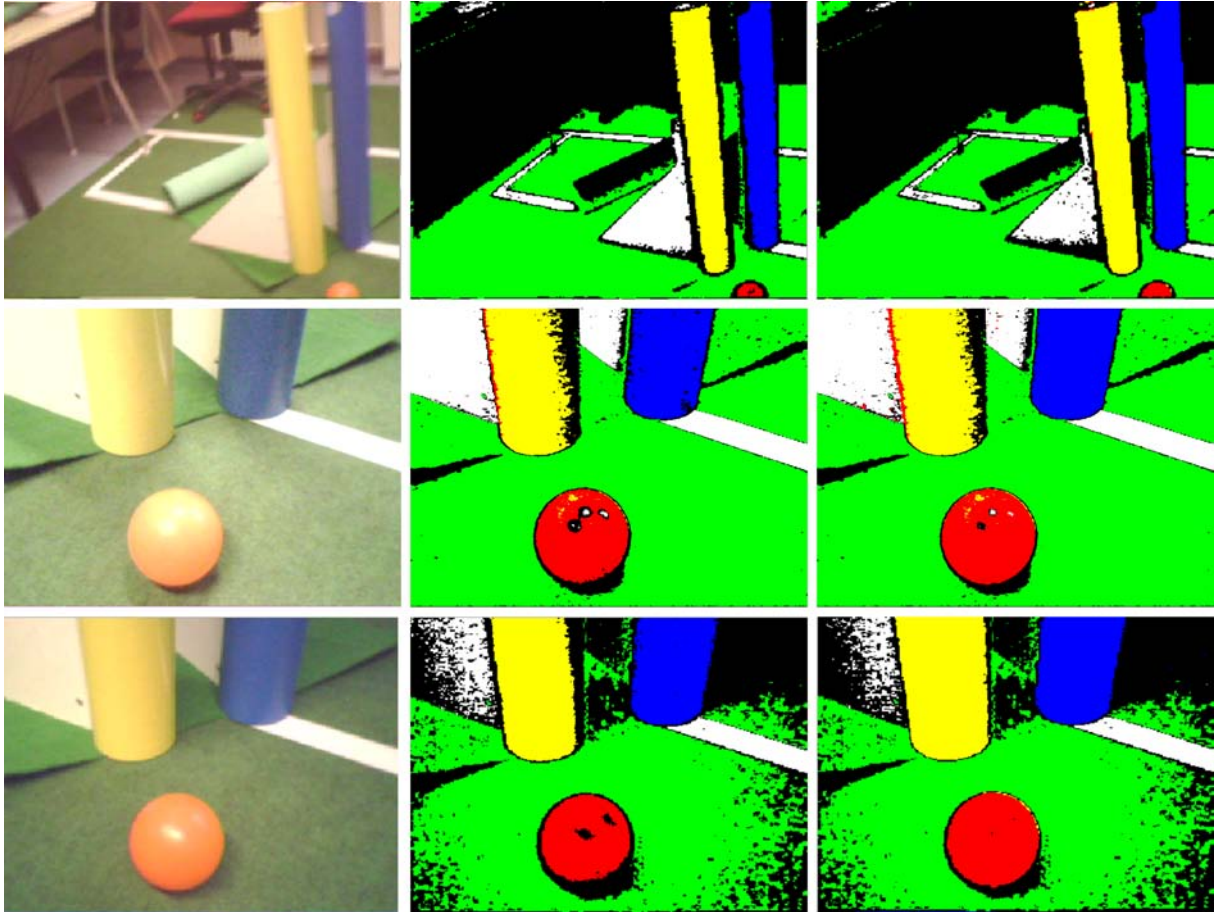


Figure 5: Result of pixel based classification. *From top to bottom:* Standard situation with usual illumination, with normal light, in a dark environment. *Middle column:* One common threshold for every class. *Right column:* Separate thresholds for every class.

small, processing a whole input image will cause both access to source and destination image and to colors (typically) distributed throughout the whole lookup table).

Anyways, a memory requirement of 16 MB is not acceptable because this will fill most of the memory available to the user after starting the robot control software. To save space, one could try to remove the channel Y (it encodes the intensity of illumination and we want to be independent of different lighting conditions, anyways). This would imply to rewrite the classifier and presumably decrease the performance for extreme illumination conditions. Another possibility is to compress the data stored in the lookup table (since we only have 5 classes, 4 bit would be sufficient) or to compress the input space (dividing all color values by a power of 2 will drastically reduce the required space, since this number is raised to the power of 3). We decided to implement the latter method. We shift the input data by one bit or more, reducing the size of the table to 2 MB or even less. Shifting by one bit has been proven to work without noticeably reducing the quality of the classification result. Reducing the size speeds up the whole process, not only by reducing the number of bytes to be processed but also by raising the probability for cache hits.

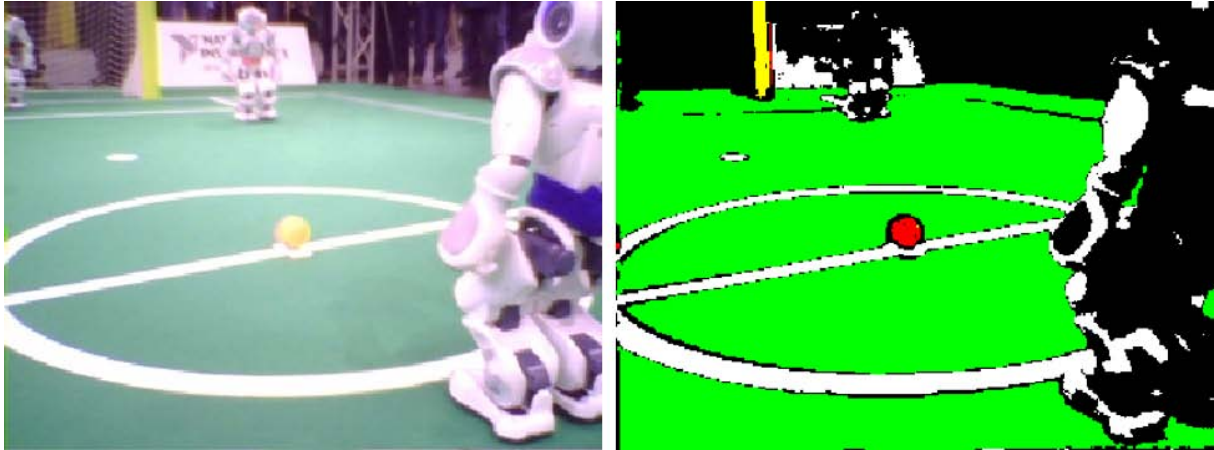


Figure 6: Test of the robust classification result during the soccer competition: German Open 2010 in Magdeburg.

## 7 Conclusion and Outlook

The classification algorithm has been evaluated at RoboCup German Open 2010. During the matches the labeling algorithm had to process images containing both very different game situations and largely varying lighting conditions. The good classification result depicted in Fig. 6 is typical and indicates that our method works well in a typical RoboCup setting. We can avoid tedious tuning of color tables and thresholds in spite of the highly variable environment.

## References

- [1] ALDEBARAN ROBOTICS: *Nao Academics Edition*. [http://www.aldebaran-robotics.com/Files/DSV3\\_En.pdf](http://www.aldebaran-robotics.com/Files/DSV3_En.pdf), 2009. Ref. 100609 - cfAcad vers. 4.1.
- [2] BAUMGARTNER, JOSEF: *Automatische Farbklassifikation zur Anwendung im RoboCup*. Master's thesis, Technische Universität Darmstadt, Department of Computer Science, 2008.
- [3] BRUCE, JAMES, TUCKER BALCH and MANUELA VELOSO: *Fast and Inexpensive Color Image Segmentation for Interactive Robots*. In *In Proceedings of IROS-2000*, pages 2061–2066, 2000.
- [4] DUDA, RICHARD O. and PETER E. HART: *Pattern Classification and Scene Analysis*. John Wiley & Sons Inc, 1973. ISBN 0471223611.
- [5] FÜRTIG, ANDREAS: *Robust Pixel Classification Tool for Matlab*. <http://www.bembelbots.de>, 2010.
- [6] HENDERSON, NAOMI, ROBERT KING and RICHARD H. MIDDLETON: *An Application of Gaussian Mixtures: Colour Segmenting for the Four Legged League using HSI Colour Space*. In VISSER, UBBO, FERNANDO RIBEIRO, TAKESHI OHASHI and FRANK DELLAERT (editors): *RoboCup 2007: Robot Soccer World Cup XI*, volume

- 5001 of *Lecture Notes in Computer Science*, pages 254–261. Springer Berlin / Heidelberg, 2008.
- [7] PALMA-AMESTOY, RODRIGO A., PABLO A. GUERRERO, PAUL A. VALLEJOS and JAVIER RUIZ-DEL SOLAR: *Context dependent color segmentation for Aibo robots*. In *IEEE 3rd Latin American Robotics Symposium, 2006. LARS '06*. IEEE, 2006.
- [8] PAPOULIS, ATHANASIOS: *Probability & Statistics*. Prentice Hall, Englewood Cliffs, NJ, 1990.
- [9] QUINLAN, MICHAEL J., STEVEN P. NICKLIN, KENNY HONG, NAOMI HENDERSON, STEPHEN R. YOUNG, TIMOTHY G. MOORE, ROBIN FISHER, PHAVANNA DOUANGBOUPHA and STEPHAN K. CHALUP, RICHARD H. MIDDLETON and ROBERT KING: *The 2005 NUbots team report*. Technical Report, The University of Newcastle, Callaghan 2308, Australia, 2006.
- [10] RÖFER, THOMAS: *Region-Based Segmentation with Ambiguous Color Classes and 2-D Motion Compensation*. In VISSER, UBBO, FERNANDO RIBEIRO, TAKESHI OHASHI and FRANK DELLAERT (editors): *RoboCup 2007: Robot Soccer World Cup XI*, volume 5001 of *Lecture Notes in Computer Science*, pages 369–376. Springer Berlin / Heidelberg, 2008.
- [11] SAHOO, P. K., S. SOLTANI, A. K.C. WONG and Y. C. CHEN: *A survey of thresholding techniques*. *Comput. Vision Graph. Image Process.*, 41(2):233–260, 1988.
- [12] SHLENS, JONATHON: *A Tutorial on Principal Component Analysis*. <http://www.sn1.salk.edu/~shlens/>, April 2009. Version 3.01; Systems Neurobiology Laboratory, Salk Institute for Biological Studies and Institute for Nonlinear Sciences, University of California, San Diego.
- [13] SRIDHARAN, MOHAN and PETER STONE: *Color learning on a mobile robot: Towards full autonomy under changing illumination*. In *The International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.