Automatic Color Calibration on a Robosoccer-field

Technical Report

Mark de Greef
0108898
mgreef@science.uva.nl

Dave van Soest
0122173
dsoest@science.uva.nl

9th June 2006

1 Introduction

This document is the technical report of our project on automatic color calibration (ACC) on a robosoccer-field. The goal of this project is to develop a method which performs ACC on an Aibo robot with the colors of the objects on the field which are predefined by the rules of the RoboCup Four-Legged League.

According to the rules of the RoboCup Four-Legged League [1], the setting of a robot soccer game consists of a set of objects with predefined shapes, sizes and colors. The number of objects, their shapes, sizes and colors may change in future rules. The set of objects in the rule book of 2006 consists of 12 distinct objects, having in total 9 different predefined colors. These colors are *white*, *pink*, *yellow*, *sky-blue*, *green*, *orange*, *red*, *blue* and *black*. The objects on the field are:

- A green field with white lines.
- A sky-blue goal and a yellow goal.
- Four beacons, uniquely identifiable by their color coding, using the colors pink, yellow and sky-blue and a white base.
- An orange ball.
- One team of four white or black Aibo robot dogs, wearing a red outfit and one team of four white or black Aibo robot dogs, wearing a blue outfit.
- One or more (assistant-)referees, wearing black clothing.

Because this is computationally the least expensive, the recognition of these objects by the robot is for the greater part performed based on the objects' colors. Therefore the ability to distinguish between the different predefined colors is very important.

There are various robot soccer tournaments all over the world at which the lighting conditions vary substantially. Although there might not be a noticeable difference for humans, the low-quality camera embedded in the robot perceives quite different colors for the same objects at different sites. These perceived colors depend havely on the light intensity, the color temperature, the color of the light source, the amount of ambient light and any additional factors. The software of the robot uses a color lookup-table to translate perceived colors to the corresponding color class, which is an element of the set of all predefined colors. The lookup table actually is a sample grid of the color lookup function $L : O \to C$, where O is the set of all different possible observed colors $\vec{o}$ in the camera's native YUV color space and C is the set of predefined color classes.

At the moment the color lookup-table is manually calibrated when arriving at a new site. This is done by recording some images with the camera and then assigning color classes to selected parts of the images. However, this job is quite elaborate and time consuming.

The goal of our project comes down to the development of a method for automatical calibration of the color lookup function L. This function can then be sampled to create the color lookup table, which is used by the software to perform fast color class determination.

The document is built up more or less chronologically to the course of this project. During the project we explored many sidepaths, some of which lead to new insights, others seemed not to lead towards any desirable results. In section 2 we briefly describe the codebase we used for this project. Initially the idea for tackling the problem was to record images while looking around on the field and subsequently clustering the observed colors (section 3). We then investigated iteratively assigning each cluster to one of the predefined colors until all objects on the field are recognized. We will describe this approach and why we left it in section 4. For our approach we have tryied several color spaces for the clustering, these color spaces are described in section 5. Section 6 describes the new set up we did our experiments in and why this was chosen. Then section 7 describes the idea of combining cluster so that we have as much groups of clusters as color classes and section 8 shows how to assign color classes to the clusters. The outcome of our experiment is shown in section 9. Finally section 10 discusses the results, what the flaws are and some possibilities for improvements in future work.

2 The codebase

For this project we use the DT2005 codebase of the Dutch Aibo Team [3]. This codebase is derived from the GT2004 codebase of the German Team [2]. The program is built out of several modules, each of which handles a specific part of the robot control or perception interpretation. One of these modules is the ImageProcessor. This module receives the images coming in from the camera and processes them according to the robot's task. When playing soccer, the objects in the view need to be recognized and their positions relative to the robot need to be estimated. In our project, the task of the robot is automatic color calibration. For the ImageProcessor this means that its function is reduced to just observing the colors. An important aspect of this function is correcting the colors of the images coming in from the camera in order to make up for the standard distortion and gain errors of the Aibo's low quality camera. The ColorCorrector class from the DT2005 code is used for color correction.

Next to the modules there is a part of the code dedicated to supporting the modules. Of the supporting code the perception part is important for our project. It has an *Image* class, which stores an image and can perform some operations on them, like color convertion. The *ColorTable** classes are also part of the perception support code and are used to store the color lookup table and to determine color classes based on observed colors.

In addition to the DT2005 codebase we use the code written by Jürgen Sturm for his master thesis [8]. Jürgen wrote a self-locator program for the Aibo. Some parts of his program were also very usuable for our project, although sometimes with some modifications. Especially the EM clustering code has been very important for our project, as this forms part of the first step of our own method throughout the whole project. Jürgen's code also provides an automatic camera calibration module that automatically finds suitable camera settings, i.e. shutter speed and gain, for the current lighting conditions.

3 Color clustering

The first step of ACC is to collect a group of data points in the desired color space. This essentially means that the robot records images which are sampled and optionally filtered. The color of each sample is then stored as a data point in the color space defined by the color components, for example YUV.

If a finite number of colors are present in the recording environment, then the data points will form normally distributed clusters in the color space. Using the expectation-maximization (EM) clustering algorithm, the mean and covariance-matrix of each cluster $p \in P$ can be determined. We will go into detail about the used EM algorithm later in this section. Now for all possible observed colors $\vec{o}$ it can be determined which cluster it most likely belongs to. By this time the first part of the color lookup function L is defined: $L_1 : O \rightarrow P$.

The remaining problem now is the correspondence between the found clusters and the actual color classes, which is a surjective mapping between the clusters and the color classes: $L_2 : P \rightarrow C$. In other words, each color class $c \in C$ has one or more corresponding clusters $p \in P$. The color lookup function now is $L = L_2 \circ L_1$.

EM algorithm

An expectation-maximization (EM) algorithm is a method for finding maximum likelihood estimates of a set of parameters. In our case we want to find the parameters of a mixture of Gaussian densities, which consist of the mean, covariance matrix and weight for all Gaussian distributions describing the data. Each Gaussian density represents a cluster.

An EM algorithm generally consists of two steps, which are repeated until some condition is met, e.g. convergence has been reached. In the first step, called the expectation step, an expectation of the likelihood is computed. In the second step, the expected likelihood from the first step is maximized to determine the maximum likelihood estimates for the parameters. Feeding these parameters to the first step, the procedure is repeated.

The used algorithm initializes the cluster parameters randomly. Then the expectation and maximization steps are executed for a fixed number of 20 times. This number of iterations seems to be sufficient to reach convergence.

4 Learning color classes by object recognition

The initial idea for tackling the ACC problem was to learn the color class assignment by feeding the images to the higher order object-perception modules which then return whether the requested objects are observed. The correct colors may then be learned using Reinforcement learning. This approach has two problems, which caused us not to follow this path.

- The first problem is that in order to keep the use of computational resources as low as possible, the object-detectors heavily rely on the color coding and do not regard the shapes and geometric structure of the objects, except for the beacon-detector, which "knows" that the pink ring should be above or below the sky-blue or yellow ring. The simplicity of the object-detectors cause objects to be seen anywhere on the field when the clusters are assigned incorrectly to the color classes. For example almost all vertical transitions between supposed yellow and pink blobs result in a beacon to be detected.
- The second problem is: how can you know whether a specific object should be in sight? This is important because false positive detection of objects will mislead the learning process. On the other hand, positioning the robot or field objects manually conflicts with the notion of unsupervised learning.

5 Other color spaces

The used implementation of the EM clustering algorithm is not limited to the use of only the camera's native YUV color space. It can rather handle any three-dimensional space. For ACC it is not necessary to try to reduce the required amount of computional power, as it is not a realtime application. As long as the output of the ACC is a lookup table which converts YUV-input to color classes, then the color class lookup works in realtime. It thus could be beneficial to study the clustering in other color spaces than YUV. Moreover, the colors red, pink, orange and yellow are located relatively close together in the YUV space, which makes it difficult for the clustering algorithm to create different clusters for data points with these colors. This is shown in figure 1.

The use of a color space other than YUV can be applied at two different points in the algorithm. Regarding the lookup function and its subdivision

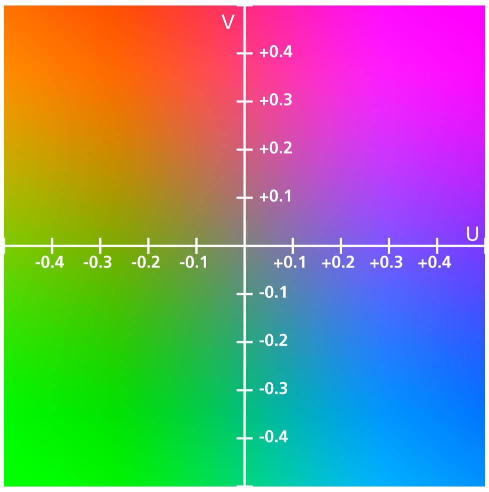

Figure 1: A cross-section of the YUV color space at $Y = 0.5$.

$L = L_2 \circ L_1$, the use of a different color space can benefit both the L_1 and L_2 phase. In the clustering phase L_1 the desired clusters might be better separated and thus easier to distinguish in a different color space. In the L_2 phase, where the cluster to color class mapping takes place, a different color space might make it easier to determine which color class a certain cluster belongs to, possibly because their relative positions rule out some or all other possibilities.

HSI

In the hue-saturation-intensity (HSI) color space the hue H represents the dominant wavelength of the color; the saturation S refers to the purity of the color, i.e. how much light of the other wavelengths is mixed in the color; the intensity corresponds to the amount of light.

The HSI color space may be well suited for the application of robot soccer, as most of the predefined colors in the soccer field can be distinguished very well by concerning only the hue, as can be seen in figure 2. The colors named in the figure are in the ideal case completely pure, i.e. they are fully saturated. However, in practice this is most often not the case, especially not for the colors green, sky-blue and blue, as we will see later.

The predefined colors which are not named in figure 2 are pink, white and black. Pink has approximately the same hue as red, but is distinguished from red by its lower saturation value. White and black are completely non-saturated and therefore they have no specific hue value. They are distinguished by their intensity.

Actually the hue dimension in figure 2 should have a circular shape

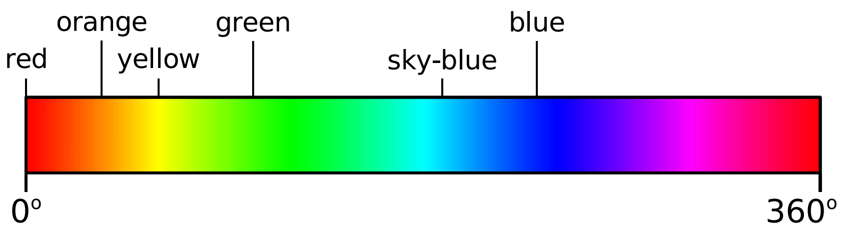

Figure 2: The hue dimension, tagged with the approximate positions of the subset of predefined colors which are theoretically pure.

and thus wrapping the hue around at 360°.

The Image class of the GT2004 codebase provides a function for converting colors from the YUV color space to HSI[1]. We decided not to use this function, but rather define our own, for two reasons. The first reason is that the existing function seems to "compress" the hue range around red, orange and yellow and stretch the hue range around blue, whereas the colors red, orange and yellow already lie very close to each other. The other reason is that the hue between blue and red is not used and can therefore be omitted in order to increase the precision for the rest of the hue range. We use the function of equation (1) to compute the hue out of an RGB color. The mentioned gap between blue and red is actually the reason for permitting the hue to be a linear dimension instead of a circular one in this application, because a cluster will not be formed in this gap. Doing so allows us to use the same clustering algorithm as for other three-dimensional spaces.

$$H = s \cdot (\frac{1}{2}\pi + \arccos\left(\frac{R-\mu}{\sigma\sqrt{2}}\right)) \tag{1}$$

Here, R is the value of the red color channel, $\mu = \frac{R+G+B}{3}$ is the brightness of the color and $\sigma = \sqrt{\frac{(R-\mu)^2+(G-\mu)^2+(B-\mu)^2}{3}}$ is the standard deviation of the three color channels R, G and B and is sometimes used as the saturation value, although we use a different measure for this. The scale factor s is used to map the outcome to the desired range, usually $[0, 255]$ to fit it in an eight-bit byte. The scale factor can also be used to "cut off" the hue range after blue.

The intensity I and saturation S are calculated as shown in equation (2) and equation (3) respectively.

$$I = \max(R, G, B) \tag{2}$$

[1]Actually the YUV value is first converted to RGB, which is then converted to HSI.

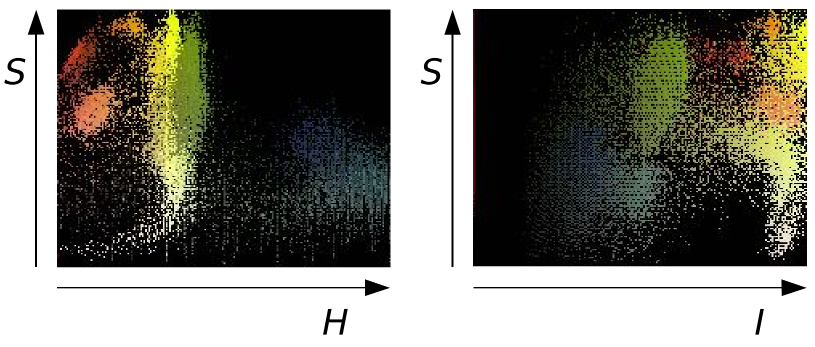

Figure 3: A typical distribution of colors in the HSI space of an image containing all predefined colors except black.

$$S = I_{max} - \frac{I_{max}}{I} \cdot \min(R, G, B) \tag{3}$$

Figure 3 shows a typical distribution over the HSI space of the perception of all predefined colors except black. It can be clearly seen how all colors, including red, pink and orange form their own separated cluster in the HSI space.

[HS]SI

We pointed out before that the colors white and black have no specific hue value. As no camera is perfect, the perceived white and black pixels will not be exactly non-saturated, but will contain some (small) amount of color. Moreover, the color of the light influences the perceived color of a white object and (partial) reflection can make a black object look like any color. The hue values of the perceived white and black will thus be randomly distributed. In other words, the hue becomes less precise as the saturation decreases.

The clustering algorithm will probably not combine the white and black pixels into their respective expected clusters if these pixels are randomly distributed over the hue dimension. Therefore we propose to use a modification of the HSI color space, which we call[2] [HS]SI.

The [HS]SI color space is constructed from the HSI color space, by multiplying the hue value H with the saturation value S. The motivation behind this is that the precision of the hue value in HSI is smaller for

[2]We were unable to find this color space in literature. We don't even know whether it has been used before. So therefore we allow ourselves to create a new name for this color space variant.

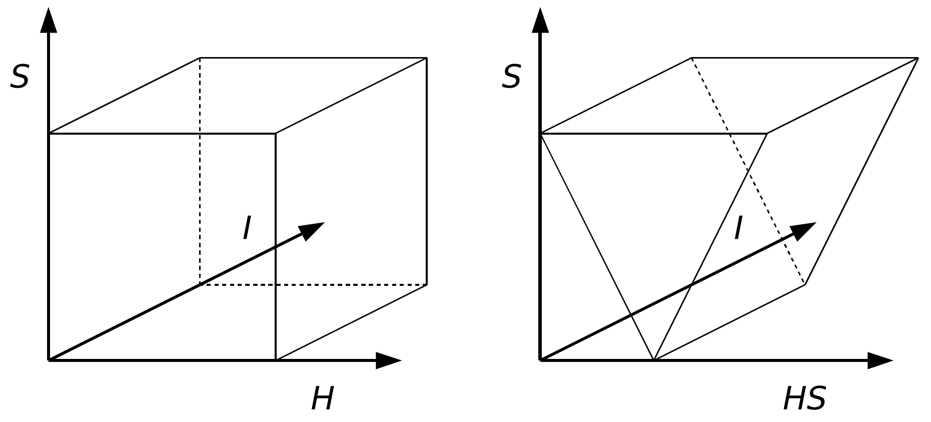

Figure 4: The transformation of the carthesian HSI color space (left) into [HS]SI (right).

decreasing saturation values and even becomes completely random for non-saturated colors, being white, grey tints and black. The [HS]SI color space is still three-dimensional, but it only uses a subset of the available space. The transformation of the HSI color space into [HS]SI is shown in figure 4.

Figure 5 shows a typical distribution over the [HS]SI space of the perception of all predefined colors except black, using the same input image as for figure 3.

6 Omitting the background

In the normal setting of the RoboCup there are a lot of colors from the background in the camera view, which makes the clusters unreliable because the clusters contain the predefined colors as well as colors from the background. We simplified our test circumstances by putting two goals on top of each other (see figure 7) which then remove the background from sight. The camera view now contains exactly those colors that are to be classified. We explicitly choose to not model this setup in the algorithm in order to maintain generality. This means that every setup is allowed, as long as all predefined colors are visible and the background is not.

We decided to only use five of the predefined colors, naemly white, yellow, sky-blue, green, and pink. We only use these colors because the clustering algorithm has problems with orange/pink/red and dark-blue/sky-blue, especially in the YUV color space. If we only use these five colors we do not have the problem of mixing up colors as these colors are well separated in the YUV color space. We can scale our work up later on with the rest of the predefined colors. The only downside to this approach is

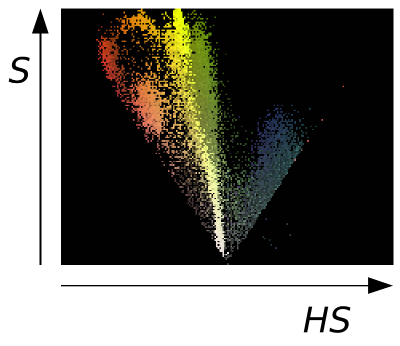

Figure 5: The distribution of colors in the [HS]SI space, using the same image as for figure 3.

that, if we place the Aibo in the field with the beacons and goals in their normal position, the background will also be assigned one of five colors. This problem is due to that all pixels that occur in the image of the Aibo after calibration will be assigned one of the five specific colors.

In the old situation we selected the pixels which we want to add as data points by only taking pixels at edge transitions. In our new setting not all edges where recognized and therefore not all different colors appeared as data points in equal amounts. It proves to work better by sampling the image using a grid. If we look at figure 6 we can see that some edges are not recognized. These edges are sky-blue/green, green/white and yellow/white. We have tried to reduce the conditions in order to correctly recognize these edges, but this causes edges to also be found inside color planes. To bypass this we just use a grid, making sure that all the colors that are in the image will be used to do the clustering on.

7 Combining clusters

As stated in section 6 we only use five color classes. For each of these five color classes we should find the appropriate clusters. As we can take more clusters than we have color classes we should have a way of combining the clusters.

To combine the clusters first a list of all clusters and a reference list is made. This reference list is initialized so that all references of the clusters point to the cluster itself. We then walk through the list of clusters and look which clusters are closest to each other according to some cluster-distance-function and let the reference of the second cluster point to the first cluster. Then all references are checked whether they point to the second cluster, if the reference is set to the first cluster. The second cluster

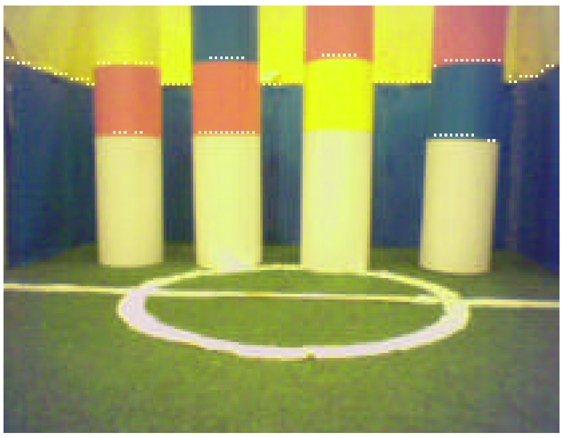

Figure 6: Scanning the image using edge transitions. The white dots are the taken points, together with the point below it.

is then removed from the list and the mean of the first cluster is updated in such a way that the new mean is in the middle between all clusters it is combined with and its own center. We continue to do this until have as much different references as we have color classes.

For the distance function between clusters we have tried several functions. For instance the function in equation (4). We have tried many different functions to combine the clusters correctly, but as it took us to much time we skipped this. Also it was unnecessary as we only took objects in our image which contained in total five different color classes. We found that taking only five clusters was in most cases enough to create the right clusters.

$$d = \Delta H^2 + w_S \cdot \Delta S + w_I \cdot \Delta I \tag{4}$$

8 Assigning color classes by color relation definitions

To identify the predefined colors the relations between these colors need to be defined. We want to be able to deal with different lighting condions and to do this we can not assign colors to specific areas in the color space using absolute boundaries. This is because if the light changes then the colors will be shifted. The relations between colors can be defined in different color spaces. If sufficient relations are defined, then a unique color class can be assigned to each cluster or combination of clusters.

In the YUV color space it is hard to define relations between the colors; e.g. these relations all need to be multi-dimensional. It is easier to define the relations in the HSI color space, because the predefined colors are well separated in the HSI color space by only the hue value (see section 5). Also if the lighting conditions change the relations in the HSI color space will remain the same, while in YUV the relations between colors may change. To define the relations in the HSI color space we have to convert the means of the clusters from YUV values to HSI values. After this we can do define relations in the HSI color space.

For each set of colors and for each color space a different set of relations is needed in order to be able to uniquely identify the color class for each cluster. Therefore the choice of the color space is important for reducing the amount and complexity of the needed relations. As stated before, the HSI space achieves this for the subset of five predefined colors which we use. The set of relations implies an algorithm which uniquely identifies all five clusters and thus explicitly defines L_2. The algorithm for our setting is as follows. Here P initially is the set of clusters or cluster combinations.

$$P_{white} = \arg\min_{p \in P} S_p,\ P := P - \{P_{white}\}$$

$$P_{sky-blue} = \arg\max_{p \in P} H_p,\ P := P - \{P_{sky-blue}\}$$

$$P_{green} = \arg\max_{p \in P} H_p,\ P := P - \{P_{green}\}$$

$$P_{yellow} = \arg\max_{p \in P} H_p,\ P := P - \{P_{yellow}\}$$

$$P_{pink} = \arg\max_{p \in P} H_p,\ P := P - \{P_{pink}\}$$

Adding rules for the other predefined colors is quite trivial. Black can be distinguished from white by its intensity; blue and orange can be distinguished by their hue values, and finally red can be distinguished from pink by its saturation value.

9 Clustering in YUV space and assigning color classes according to HSI relations

For the final version of our method we choose to cluster the colors in the YUV color space, because in this color space the clusters were best separated. To get the right color class for each cluster we first derive the HSI values of the means of the YUV color clusters. We did this because in HSI it was easier to find the relations between the different color classes as described in section 8. Having the method we tested it using different lighting conditions. These conditions have three binary parameters:

- curtains open/closed
- lights on/off
- tl-lights on/off

In figure 7 to 14 in appendix A the results of the clustering and color assignment are shown. We show the original image, the color classes of the image, the YU space and the VU space of the YUV color space and finally the hue of the colors that are sorted on their hue (pink, yellow, green and sky-blue). The results show that despite the variety of lighting conditions the cluster assignment L_2 is often completely correct and the greater part of the pixels is assigned to the correct cluster (L_1). We can see that even when it is very dark (see figure 10) we are able to distinguish some colors.

In two cases the automatic color calibration algorithm did not work flawlessly. The first case is in figure 14 where the sky-blue and green clusters are switched. This can be explained by the fact that there is very little light, making the green very dark. Also the light is in this case blueish (only tl-light). We can also see this by the fact that the hue of the cluster means of green and sky-blue are very close to each other. Because of this the field is classified as being sky-blue and the sky-blue of the goal and beacons is classified as green.

The second error we see in figure 9 and 13, this is due to the fact that we only have the big lights on and not the tl-lights making the light have a yellowish color. Because of the light being slightly yellowish, the yellow and white clusters get to close to each other, making them appear as one cluster to the clustering algorithm.

10 Conclusions

The results in section 9 show that the goal of this project is reached for the greater part. We achieved correct classification of a subset of the predefined colors under the majority of varying lighting conditions. Together with the potential of possible future work, which we discuss later in this section, we think that this may lead to a satisfactory working algorithm, which can replace manual color calibration.

Our search for a correctly working algorithm has lead us along several sidepaths, some of which we have not thoroughly investigated, due to time shortage. The initial idea of learning the color classes by object recognition was left shortly after starting the project, because we concluded that this approach is not feasible.

We then explored the use of alternative color spaces, predominantly HSI and [HS]SI, which show better distinguishable clusters than with YUV. However, we have not managed to let the clustering algorithm provide satisfying results when using these color spaces, because the function for measuring the distance between two colors in HSI and [HS]SI has to be researched first.

When trying to implement the auto color calibrator in the normal setting (goals and beacons in the right place) we see that the we can not find the right clusters. This is because of two things, first of all the the detector for beacons and goals do not take into account the shape and size of the object. So for instance the beacon detector will find a beacon everywhere where it finds the to colors of the beacon below each other. For this reason we cannot find which is the right color class for a cluster. The second reason is that we have lots of background colors making it hard to get a cluster in which only the right color of the object occurs and not background. A solution to this problem is to use more clusters, so the original clusters are split up, and combine these clusters to get the right color class for the clusters. We solved these problems by omitting the background from the camera view, i.e. making sure that the complete camera view contains only objects with predefined colors. In this way we do not have these problems and we can find the right clusters.

Letting the clustering algorithm return more clusters than the expected number of color classes and then combining these clusters could be beneficial, but then the function for determining the distance between two colors needs to be researched first.

For the assignment of color classes to clusters we have used the fact that in HSI the relations between the clusters can be very well defined. In this way we can just walk through a list of clusters and assign the right color classes to them. To do this the centres of the clusters have to be converted from YUV to HSI first. After that the color classes can be assigned relatively easy.

From our results we see that even with very little light we still can find some colors. The only time that our approach makes some errors is when the light is too colored (in our cases blue or yellow).

Future work

There are many possibilities for future work on this project. Most of them depend heavily on each other. We will provide a (possibly incomplete) list here.

- *Determining relations between all nine predefined colors.* In our project we finally only used five of the nine predefined color classes. This of course depends on correct clustering of the color space.
- *Defining color and cluster distance functions.* The clustering and cluster combining algorithms depend on the functions that determine the distance between colors and color clusters. Defining these functions for the HSI or [HS]SI color space would be quite valuable.
- *Determining the color of the light source.* From figure 5 it can be seen that the white cluster stretches in the direction of the yellow hue. Probably this has to do with the light chrominance being yellowish. Thus the 'direction' of the white cluster in the [HS]SI space might reveal the color of the light source. This knowledge can be used to

better separate white and, in this case, the yellow cluster (which are wrongly combined in figure 9 and figure 13).

- *Using an other color clustering or color space partitioning algorithm.* For this project the use of the EM clustering algorithm was more or less one of the starting points. However, it is never argued why EM clustering would work better than other partitioning techniques. Therefore research in this direction might provide a better solution for the L_1 phase. One of the possibilities is watershed segmentation.

References

[1] RoboCup Technical Committee. *RoboCup Four-Legged League Rule Book.* April 2006. http://www.tzi.de/4legged/pub/Website/Downloads/Rules2006.pdf.

[2] T. Röfer et al. German team - robocup 2004. Technical report, Center for Computing Technology et al., 2005. http://www.germanteam.org/GT2004.pdf.

[3] A. Visser J. Sturm and N. Wijngaards. Dutch aibo team: Technical report robocup 2005. Technical report, Universiteit van Amsterdam el al., 2005. http://www.dutchaiboteam.nl/research/publications/DAT2005TechReport.pdf.

[4] M. Jüngel. Using layered color precision for a self-calibrating vision system. In *Proceedings of the 8th International Workshop on RoboCup 2004*, Lecture Notes on Artificial Intelligence. Springer Verlag, 2005.

[5] B. Minten, R. Murphy, J. Hyams, and M. Micire. Low order complexity vision-based docking. *IEEE Transactions on Robotics and Automation*, 17(6):922–930, December 2001.

[6] D. Schulz and D. Foz. Bayesian color estimation for adaptive vision-based robot localization. In *Proceedings of IROS*, 2004.

[7] M. Sridharan and P. Stone. Real-time vision on a mobile robot platform. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, August 2005.

[8] J. Sturm. Master thesis draft. 2006.

A Result figures

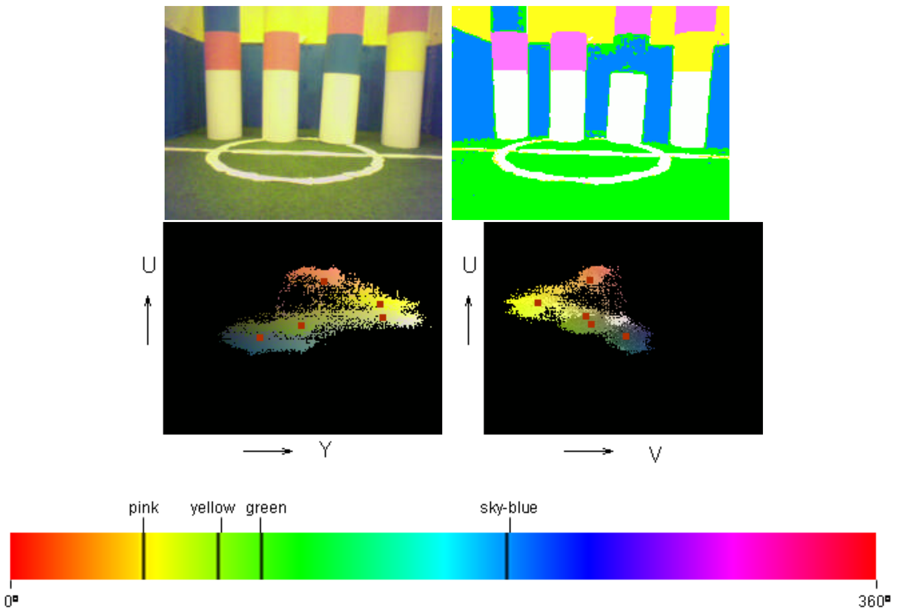

Figure 7: Curtains closed, lights on, tl-lights on

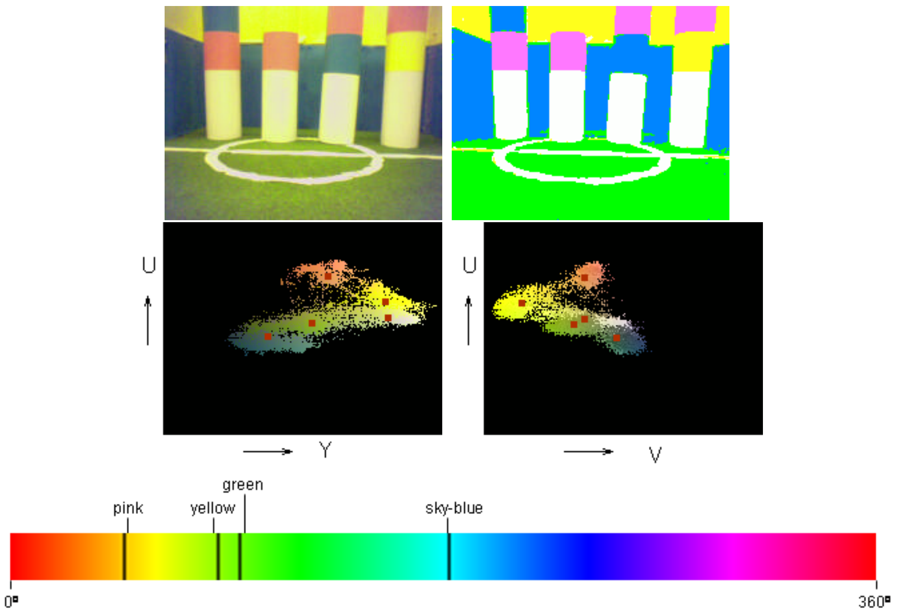

Figure 8: Curtains closed, lights off, tl-lights on

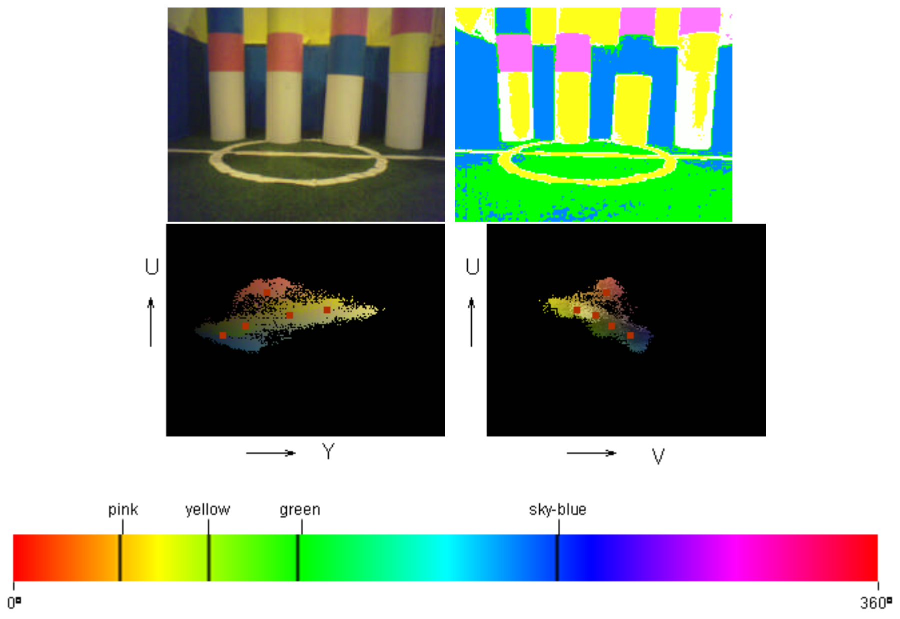

Figure 9: Curtains closed, lights on, tl-lights off

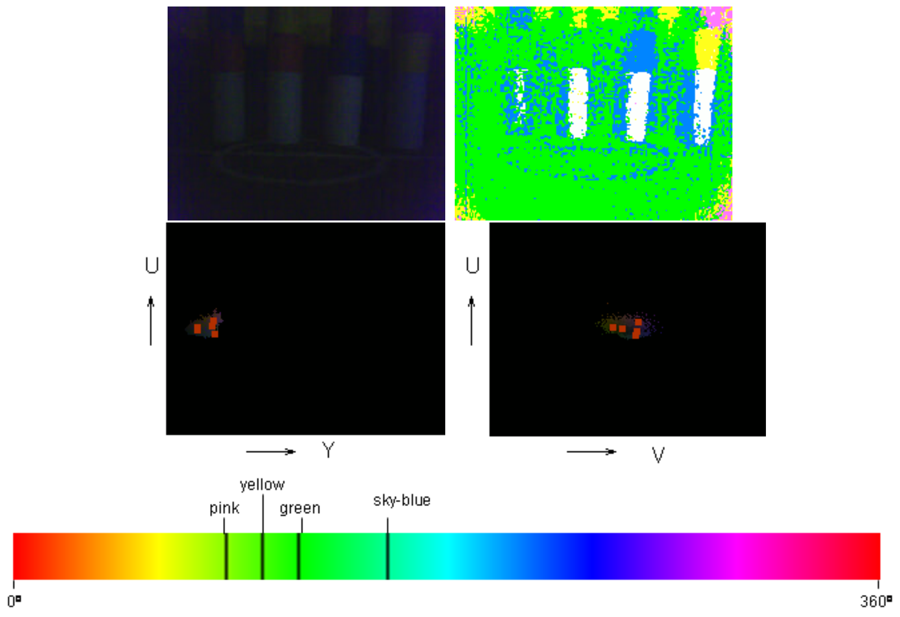

Figure 10: Curtains closed, lights off, tl-lights off

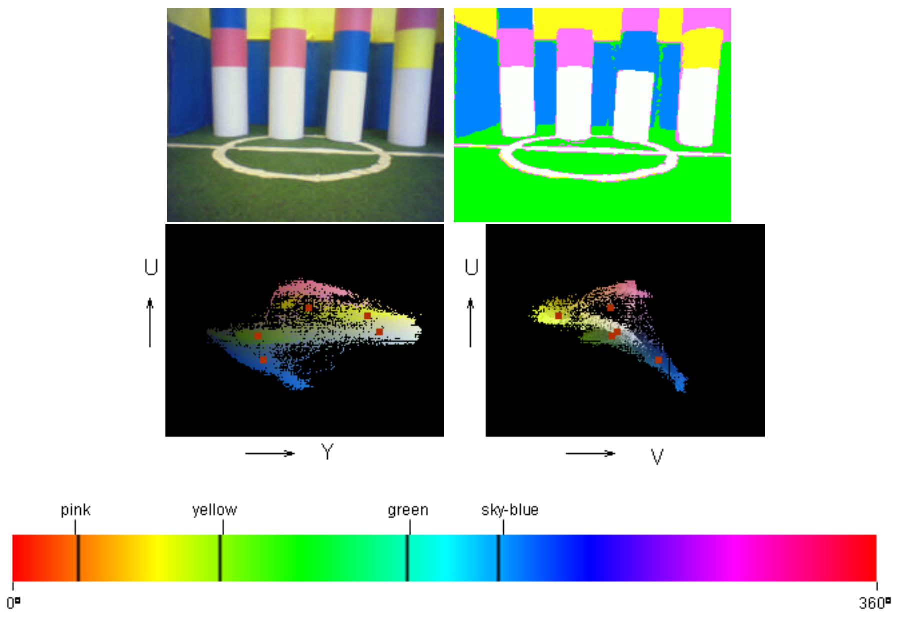

Figure 11: Curtains open, lights on, tl-lights on

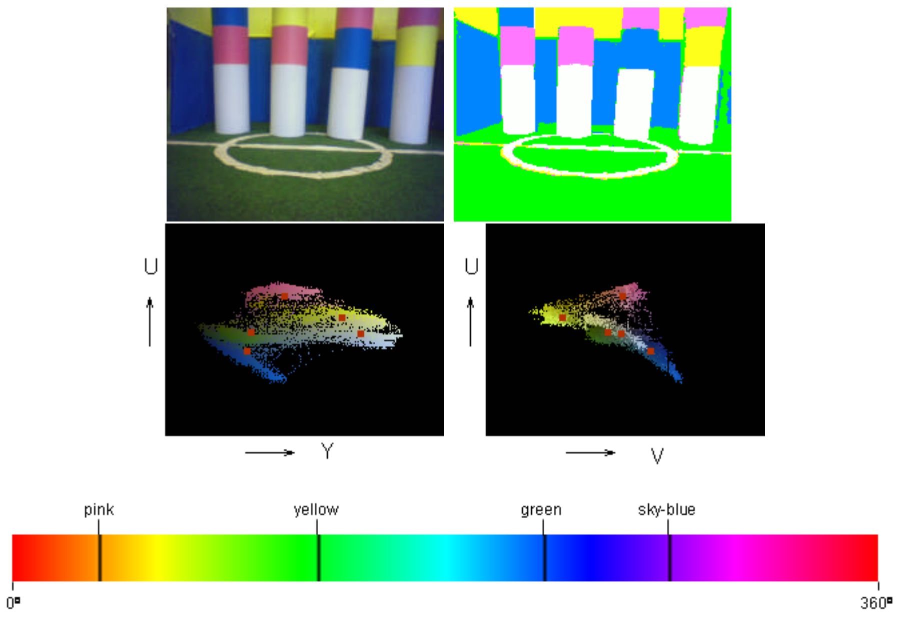

Figure 12: Curtains open, lights off, tl-lights on

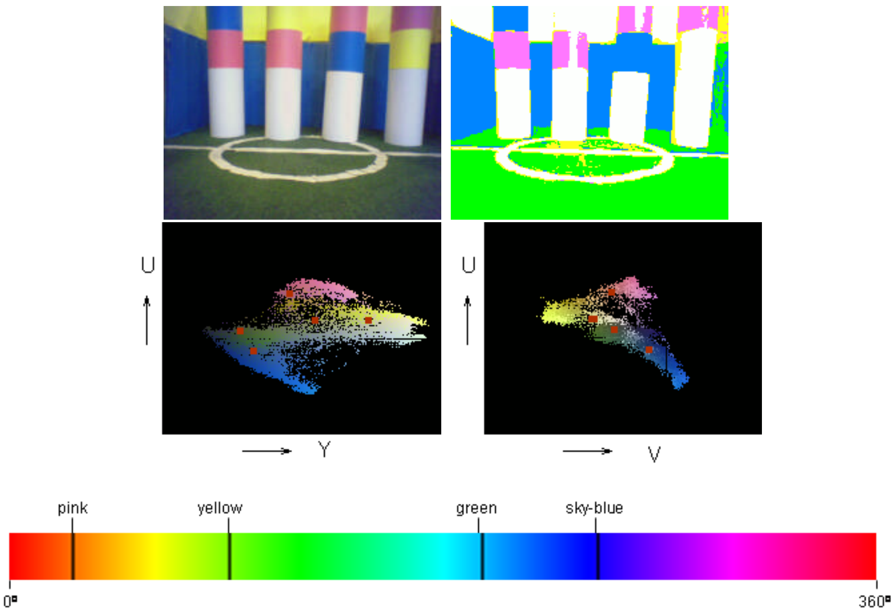

Figure 13: Curtains open, lights on, tl-lights off

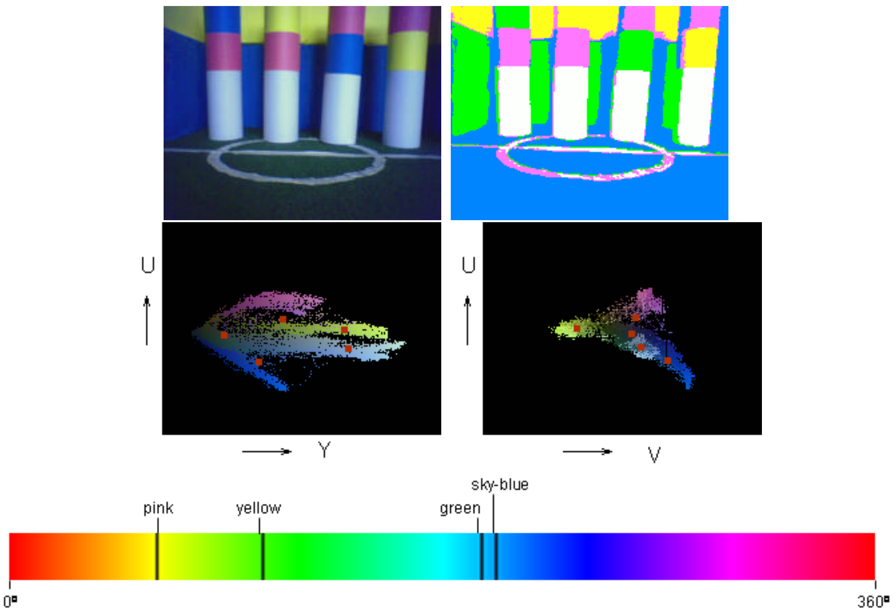

Figure 14: Curtains open, lights off, tl-lights off