Finding Colored Objects in a Scene*

Christian Balkenius Birger Johansson

Lund University Cognitive Science
Kungshuset, Lundagård
222 22 LUND, Sweden

Abstract

This report describes a fast method for finding regions with a particular color in a visual scene. The method is general enough to be used in any situations were a number of objects with known color need to be found in an image. We demonstrate how the algorithm can be used to find and track robots with colored markers, symbols with a particular color as well as for simple color-based face detection.

1 Introduction

In many cases, the color of an object is its most characteristic property and this suggests that it should be possible to find an object in a visual scene based on its color. In principle, this could be done by searching for all pixels in an image with a particular range of RGB values, but in practice, this does not work very well. There are a number of reasons for this, but the most important is that the RGB representation of color mixes intensity and color in all the three channels (Balkenius et al., 2003). For example, if the illumination increases, the values in all three channels will increase although perceptually, the color would not have changed. To allow for such changes, the range of RGB values have to be very large and this would include colors that are not necessarily of the target color.

Another problem is that the color of the illumination may change over the day, which will have different effects on the different RGB channels. This is in general a very hard problem to solve and requires color constancy (see, for example Ebner, 2004). However, it is possible to get by with simpler heuristics in many cases.

Here, we describe a fast algorithm that can compensate for these variations and find pixels in an image with a particular color. There are a number of problems that need to be solved. First, it is useful to map the colors from RGB space to a more useful representation. Here we will map the colors onto the rg-chromaticity plane where the each color is coded as a hue and saturation, but the intensity is discarded. Second, it is often necessary to compensate for variations in illumination. This can be done by approximating the location of the white point in the color space and relating all colors to this point. Here, we assume that the average color of the scene is gray and use this as a reference (Gershon et al., 1987). Third, we need to find the pixels in the image that have the desired color. This is done by first testing if the color of each pixel lays in the target color region. Fourth, the different objects with the target color are found

*Balkenius, C. & Johansson, M. (2007). *Finding Colored Objects in a Scene*. LUCS Minor, 12.

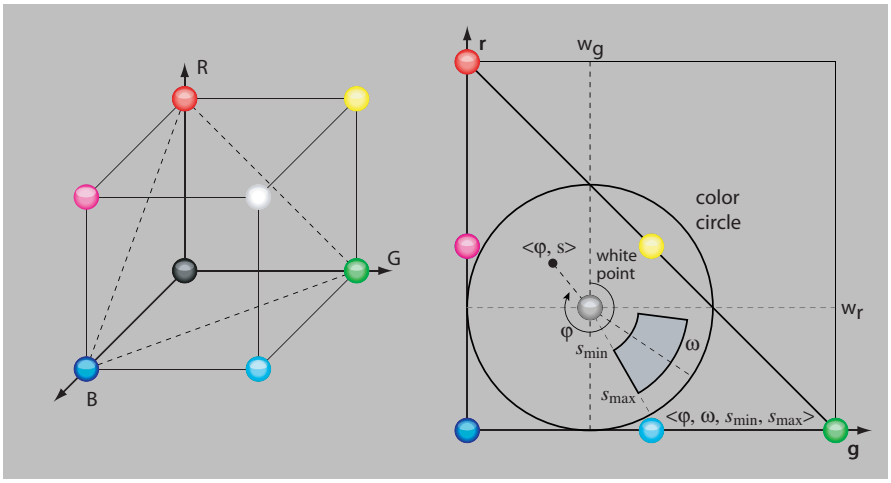

FIGURE 1: *The transformation from RGB-space to the rg-chromaticity plane. A color is defined as a segment of a circle centered at the white point with the least saturated part removed. The color region is specified with the four parameters φ, ω, and $s_{\min}$ and $s_{\max}$.*

using a clustering method which groups together pixels with the target color that lies close to each other in the image. Finally, the clusters can be sorted according to size or matched to the positions of the clusters in the previous image to track objects that are moving.

2 Color Transformations

The first step in the color tracking algorithm is a transformation of the pixel coordinates in the RGB image to the rg-chromaticity plane (Fig. 1). We will assume that the RGB values are in the range $[0..1]$. This transformation is obtained by dividing the R and G components of the pixel coordinates with the sum of the three color channels.

$$r = \frac{R}{R+G+B}$$

$$g = \frac{G}{R+G+B}$$

The two dimensional rg-plane contains the colors of the original RGB space, but the intensity of each pixel has been discarded. The intensity of the color is calculated as

$$I = R+G+B.$$

Note that this is not the best way to estimate perceived intensity since the three colors contribute differently to the perception of intensity, but this approximation is sufficient for our present purposes.

The above transformation maps red, green and blue onto the three corners of a triangle. Gray and white are both mapped to the same point at $\langle 1/3, 1/3 \rangle$. This is called the *white point* of the color triangle. Note that the transformation of black in the RGB-space, i. e. $\langle 0, 0, 0 \rangle$, is not defined (but may be mapped onto the white point).

It is possible to construct a color circle around the white point where the angle describes the hue and the distance from the center depends on the saturation of the color. Because of the shape of the color space, the same distance from the white point does not correspond to the same saturation in each direction.

Calculation of white point The white point of the image can be estimated from the image data if we assume that the average color of the image is gray. The average $\bar{R}$, $\bar{G}$, and $\bar{B}$ of each of the three color channels are calculated and projected onto the rg-plane:

$$\bar{r} = \frac{\bar{R}}{\bar{R} + \bar{G} + \bar{B}}$$

$$\bar{g} = \frac{\bar{G}}{\bar{R} + \bar{G} + \bar{B}}$$

the white point is set to

$$w = \langle \bar{r}, \bar{g} \rangle \, .$$

Assuming that the illumination is equal in the red, green and blue channels, the white point can be directly set to $\langle 1/3, 1/3 \rangle$.

If the illumination varies over the image it can be advantageous to use different white points for different pars of the image, but in this case it is important that the average color of the different areas is the same. Otherwise, the same color in the different regions will not be mapped to the same point.

Color coordinates Each color in the rg-plane is mapped to two values. The the angle around the white point

$$\varphi = \arctan\left(\frac{r - \bar{r}}{g - \bar{g}}\right)$$

and the saturation

$$s = \sqrt{(r - \bar{r})^2 + (g - \bar{g})^2},$$

that is, the distance from the white point (See Fig. 1). This results in the color coordinate

$$p = \langle \varphi, s \rangle \, .$$

Fig. 1 shows the result of mapping all the pixels of images onto the $\langle \varphi, s \rangle$ space. The differently colored object are clearly seen as dots within different regions of the color space.

3 Classifying Colors

To find a region of the image with a particular color, we first need to classify the pixels. We define a target color region using four values, the angle φ, and the minimum and maximum saturation $s_{\min}$ and $s_{\max}$, and finally the width of the region w. We want to

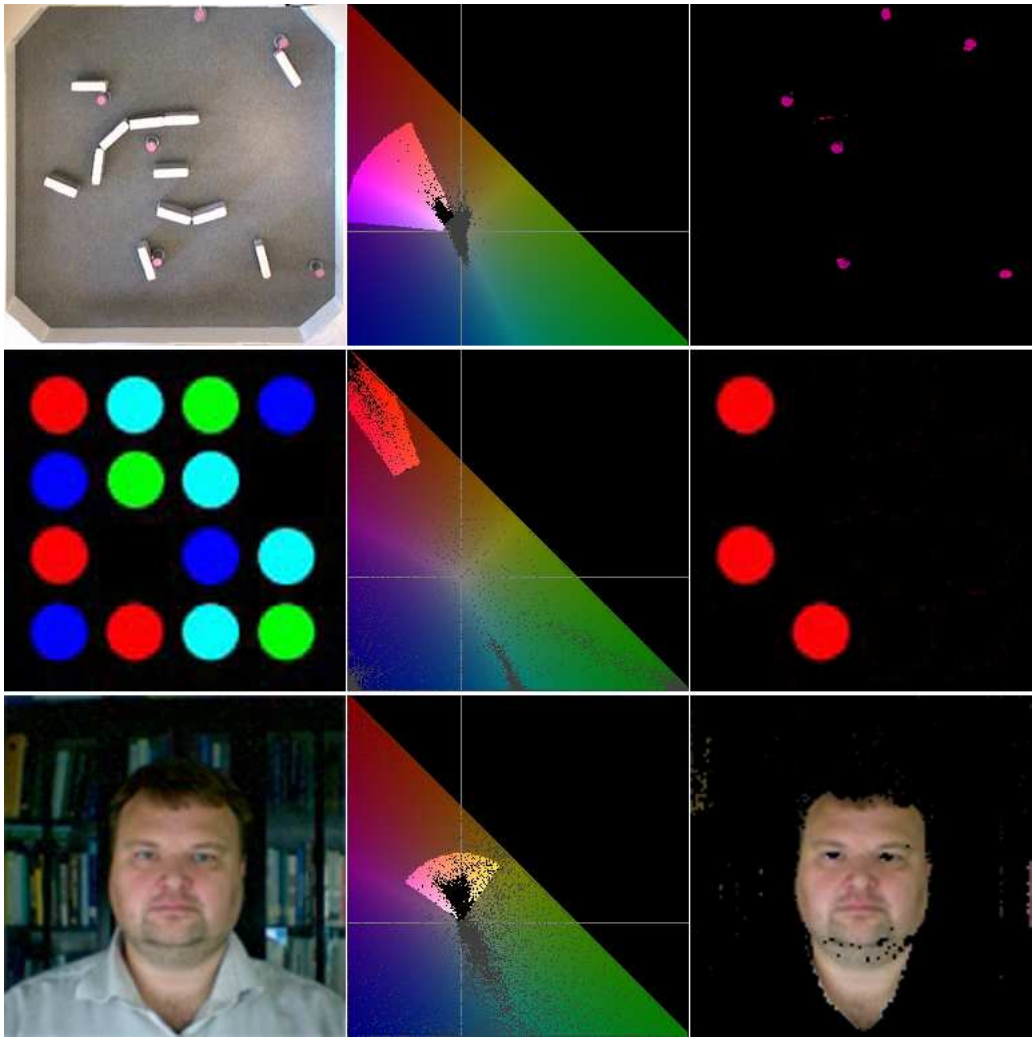

FIGURE 2: *The algorithm applied to three different scenes. Top row: Six robots with colored markers. Middle row: A simple scene with dots of different colors. Bottom row: An image of a face. Left. the input image. Middle. The rg-colorspace with the locations of the color coordinates of the pixels in the image and the target color region highlighted. Right. The detected pixels in the scene.*

find all pixels, the color or which lies within this region. We can do this directly from the $\langle \varphi, s \rangle$ representation by testing

$$s_{\min} < s < s_{\max},$$

and

$$\varphi - \omega < \varphi < \varphi + \omega.$$

However, this method is very time consuming since it involves calculating a square root and arctan for each pixel in the image. There is also a division, that should be avoided if possible. Fortunately, this test can be made much faster by getting rid of all these operations.

A Faster Inclusion Test To make the color test faster, we map the problem onto a vector representation. Assuming that the color angle for the region is given in degrees, the target color vector $c = \langle c_r, c_g \rangle$ for a region is given by

$$c_r = \sin\left(\frac{2\pi\varphi}{360}\right),$$
$$c_g = \cos\left(\frac{2\pi\varphi}{360}\right).$$

The width of the region is represented by the value w which is calculated as,

$$w^2 = \cos^2\left(\frac{2\pi\omega}{360}\right).$$

The above calculations only need to be done once for each target region. To check whether the color of a pixel lays within the target region, we first calculate the square of the length of the color vector $\langle r, g \rangle$ as

$$L^2 = r^2 + g^2,$$

and the scalar product of the color and the target color

$$m = c_r r + c_g g,$$

The following conditions are then tested

$$s_{\min}^2 < L^2 < s_{\max}^2, \tag{1}$$

$$m > 0. \tag{2}$$

and

$$m^2 > w^2 L^2. \tag{3}$$

The color of the pixel lays within the target region if both these conditions are met. Note that this formulation of the condition completely avoids any computationally slow operations such as divisions, square roots or trigonometric functions, except for the three initial calculations of c_r, c_g and w^2, but these are only made once and not for every single pixel.

4 Color Clustering

The second step in finding potential targets is to cluster the points that fulfills the requirement above. A cluster C_i is defined by the set of coordinates of all its included pixels. The center of the cluster $c = \langle c_x, c_y \rangle$ is calculated as the average position of all pixels that have already been included in the cluster. To find the different clusters in the image, we iterate over all pixels that fulfills the requirements (1) and (3) above.

Here, we use a simple algorithm that works well when the number of targets are relatively sparse. For each pixels $\langle x, y \rangle$ with the target color we test whether there is a cluster center within the distance d from it. In that case, the new pixel is added to the cluster. Otherwise, a new cluster is created with this pixel as its center. This procedure continues until all pixels have been assigned to a cluster.

This method works well if the distance d is set to match the approximate size of the targets. It would be possible to use much more advanced clustering algorithms (Duda et al., 2000), but in many cases this simple method is sufficient.

5 Post-Processing

There are a number of methods that can be used after clustering to produce a more stable output. These techniques can be used together but my also hinder each other in some cases.

Noise Reduction After clustering, all clusters with less than N pixels can be discarded as noise. This is useful in many cases but may have the unwanted consequence that a cluster that is temporarily smaller may disappear.

Size Selection If the number of targets are known, for example if we are tracking a fixed number of robots, it is often a good idea to select the k largest clusters and discard any smaller ones.

Tracking If the target objects are known to move continuously in the scene, it is useful to assign the same index to the cluster belonging to the same target at each time step. This can be accomplished either by finding the closest cluster from the previous time step after clustering, or by using the centers of the previous clustering as the starting point for the clusters in the current time step.

We use the former method as the latter is very sensitive to noise. When the new set of clusters have been found, they are matched to the clusters found on the previous time step. This is controlled by a parameter that decides how far a cluster can move and still be matched. Clusters that are not assign a new location are marked as unused and may be reassigned at a later stage.

Although this method is far from perfect and much more advanced tracking methods exist, it works well in situations where the target object are not moving very fast in the image and targets are well separated. In more complicated cases, it is necessary to use a more able tracking system after this post-processing.

6 Implementation Details

The algorithm described above has been implemented as a set of modules in Ikaros (Balkenius et al., 2007). The module *ColorTransform* transforms the input image from RGB-space to rgI-space, that is, the rg-chromaticity plane together with an intensity channel. The result is sent to the *ColorClassifier* module which performs the color classification and sends its result to the module *SpatialClustering* which finds the clusters in the image. This sets up a processing pipeline from the image to the table with the found clusters. The parameters of the modules can be set as follows:

```
<module
      class = "ColorTransform"
      name = "ColorTransform"
      transform = "RGB->rgI"
/>

<module
      class = "ColorClassifier"
      name = "ColorClassifier"
      color = "145 degrees"
      width = "30 degrees"
      saturation_min = "0.05"
      saturation_max = "0.35"
      compensation = "yes"
      statistics = "yes"
/>

<module
      class = "SpatialClustering"
      name = "SpatialClustering"
      cluster_radius = "30"
      min_cluster_size = "50"
      tracking_distance = "0.1"
      no_of_clusters = "6"
/>
```

Most of the parameters are self explanatory. The compensation parameter decides whether the module should try to compensate for the color of the illumination using the gray-world assumption described above.

The output from the module is a table of coordinates for the found clusters. For debugging purposes, there are also two additional outputs. The first one shows the locations of the input pixels in the color space. The second output shows the spatial location of those pixels that pass the color test (See Fig. 2). These outputs can be turned on or off using the parameter *statistics*. Full documentation of the module together with the source code can be found at the Ikaros web site (www.ikaros-project.org)

The performance of the algorithm is reasonably good. We tested an implementation with very little optimization except of the ones described above. On a 2.66 GHz Dual-Core Xeon computer, an image of size 704×480 pixels with the scene shown in Fig. 2 is processed in on the average 7.3 ms. This is fast enough for real-time tracking in a DV-stream.

Acknowledgements

This work was supported by the EU project MindRaces, FP6-511931.

References

Balkenius, C., Johansson, A. J., and Balkenius, A. (2003). Color constancy in visual scene perception. *Lund University Cognitive Studies*, 98.

Balkenius, C., Morén, J., and Johansson, B. (2007). System-level cognitive modeling with ikaros. *Lund University Cognitive Studies*, 133.

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification*. Wiley Interscience.

Ebner, M. (2004). A parallel algorithm for color constancy. *Journal of Parallel and Distributed Computing*, 64(1):79–88.

Gershon, R., Jepson, A. D., and Tsotsos, J. K. (1987). From [R,G,B] to surface reflectance: Computing color constant descriptors in images. In McDermott, J. P., editor, *Proc. of the 10th Int. Joint Conf. on Artificial Intelligence*, volume 2, pages 755–758.