Efficient Color-Based Image Segmentation and Feature Classification for Image Processing in Embedded Systems

Alexander Jungmann, Bernd Kleinjohann, Lisa Kleinjohann
C-LAB
University of Paderborn, Germany
{global, bernd, lisa}@c-lab.de

Maarten Bieshaar
Faculty of Computer Science,
Electrical Engineering and Mathematics
University of Paderborn, Germany
maarten@mail.uni-paderborn.de

***Abstract*—In this paper, a color based feature extraction and classification approach for image processing in embedded systems in presented. The algorithms and data structures developed for this approach pay particular attention to reduce memory consumption and computation power of the entire image processing, since embedded systems usually impose strong restrictions regarding those resources. The feature extraction is realized in terms of an image segmentation algorithm. The criteria of homogeneity for merging pixels and regions is provided by the color classification mechanism, which incorporates appropriate methods for defining, representing and accessing subspaces in the working color space. By doing so, pixels and regions with color values that belong to the same color class can be merged. Furthermore, pixels with redundant color values that do not belong to any pre-defined color class can be completely discarded in order to minimize computational effort. Subsequently, the extracted regions are converted to a more convenient feature representation in terms of statistical moments up to and including second order. For evaluation, the whole image processing approach is applied to a mobile representative of embedded systems within the scope of a simple real-world scenario.**

Keywords-Image Segmentation; Feature Classification; Run-Length Encoding; Moments; Embedded Systems

I. INTRODUCTION

Image processing is a crucial service needed by many intelligent applications like driver assistance, surveillance or production in order to obtain and analyze visual information about the current situation. Usually this data and processing intensive task has strong timing requirements and is often realized on embedded systems. Although the performance capabilities of such embedded systems rise steadily, image processing approaches that stick appropriately to the prevalent restrictions regarding computational power and memory are required. Therefore many image processing solutions are realized on special hardware components such as Field Programmable Gate Arrays (FPGAs) or Digital Signal Processors (DSPs) in order to increase the processing speed. But this reduces the portability of the approach. In contrast, the approach presented in this paper can be applied to any embedded system that provides a Linux based operating system., independent of the underlying CPU instruction set.

The focus of this paper lies on the low level image processing phase, namely the combination of an efficient image segmentation algorithm with a simple but flexible feature classification mechanism. Algorithms and data structures were optimized with regard to computational effort and memory consumption. By image segmentation the original pixel-based data is drastically reduced to a feature-level representation, which requires significantly less memory on the one hand and is more convenient as well as less computational expensive for subsequent information processing steps on the other hand. As a convenient feature descriptor, which further reduces the visual data produced by the image segmentation, an optimized representation of features based on statistical moments is used. For feature classification a memory efficient color class representation together with computationally efficient access methods was developed, that nevertheless allows a representation of sophisticated color spaces made up of several subspaces. The feature classification is realized by two flexible heuristics. They are integrated into the segmentation process, in order to avoid further processing of pixels in the original image that can be classified as not relevant due to their color, thus reducing the computational effort even further.

A simple evaluation scenario with an embedded system - the miniature robot BeBot [1] - was developed to prove the performance of the presented approach. Within this scenario, the BeBot only relies on visual data in terms of images captured by its camera and processed by the image processing approach presented in this paper. By doing so, subsequent object detection and information processing steps for autonomous behavior are facilitated.

This paper is organized as follows. At first, the basic segmentation algorithm as well as a convenient feature representation are presented in Section II. Section III subsequently describes a flexible color-based feature classification mechanism. Afterwards, the real-world scenario, in which the whole image processing approach was successfully applied, is introduced in Section IV, followed by the associated results in Section V. Some existing approaches that are similar with respect to the basic concepts are briefly discussed in Section VI. Finally, the paper concludes with Section VII.

 ISBN: 978-1-61208-188-5

II. Feature Extraction by Image Segmentation

The fundamental idea of image segmentation is the identification of contiguous blocks of pixels (so-called *regions*) that are homogeneous with respect to a pre-defined criterion. Afterwards, those regions are abstracted into *features* that are compactly described in terms of position, size and shape. By doing so, the original pixel-based data is drastically reduced to a feature-level representation, which requires significantly less memory. Furthermore, features are more convenient as well as less computationally expensive for subsequent information processing steps. In fact, the overall challenge of image segmentation on embedded systems is to extract adequate features in order to reduce the memory consumption and computational effort for subsequent processing steps while simultaneously minimizing the computational effort of the actual segmentation process.

For that reason, our approach was developed to process an entire image only once, while simultaneously identifying and extracting regions, which are in turn compactly represented in order to reduce the memory consumption.

A. Region Representation

A convenient representation of a region during the segmentation process is crucial since it depends on the available memory and computational power. For that reason, our segmentation approach incorporates the *Run-Length Encoding* concept [2]. Sequences of adjacent pixels are encoded as runs, where a single run is compactly represented by means of only three integer values:

$$run_i = (x_i, y_i, l_i), \tag{1}$$

with x_i, y_i being the coordinates of the starting pixel and l_i denoting the number of adjacent pixels within the same row. Furthermore, a region R may consist of a set of n runs:

$$R = \{(x_1, y_1, l_1), (x_2, y_2, l_2), \ldots, (x_n, y_n, l_n)\} \tag{2}$$

While adding a pixel to a region is nothing but incrementing the length l_i of the related run run_i, merging two regions is realized by simply appending the sequence of runs of the first region to the sequence of runs of the second region.

B. Segmentation Algorithm

The entire image segmentation process is depicted in Algorithm 1. The main loop (lines 8-43) iterates row by row over the whole image, starting at the topmost row. The inner iteration loop (lines 9-42) processes each pixel within a row once, starting with the leftmost pixel.

After identifying a possibly existing left adjacent run run_{left} as well as its associated region R_{left} (lines 10-14), a heuristic H_i (cf. Section III-B) is applied in order to check if the current pixel is interesting at all (line 15). If so, a heuristic H_m (cf. Section III-B) is applied in order to check if the current pixel and the left adjacent region R_{left} can be merged (line 17). If heuristic H_m succeeds, the *region growing* step takes place by adding the current pixel to the left adjacent run run_{left} (length of run_{left} is incremented by value 1) and updating the associated region R_{left} with respect to its average color. Afterwards, the algorithm directly continues with the next pixel (line 19).

If either heuristic H_i or H_m fails, the left adjacent run

Algorithm 1 Segmentation Algorithm

```
 1: img = latest camera image
 2: regions = {∅}
 3: length_min = minimal length of a single run
 4: /* heuristic for deciding if a pixel is interesting at all */
 5: H_i = configure(interesting_heuristic)
 6: /* heuristic for deciding if pixels and/or regions can be merged */
 7: H_m = configure(merge_heuristic)
 8: for all row ∈ img do
 9:    for all pixel ∈ row do
10:       run_left = left_adjacent_run(pixel)
11:       if run_left not exists then
12:          goto line 35 /* start new run */
13:       end if
14:       R_left = region(run_left) /* run_left's associated region */
15:       if H_i(pixel) is TRUE then
16:          /* current pixel is interesting */
17:          if H_m(pixel, R_left) is TRUE then
18:             add(run_left, pixel) /* region growing */
19:             goto line 9 /* continue with next pixel */
20:          end if
21:       end if
22:       /* finalize a previously built left adjacent run */
23:       if length(run_left) < length_min then
24:          remove(regions, R_left)
25:       else
26:          regions_top = top_adjacent_regions(run_left)
27:          for all R_top ∈ regions_top do
28:             if H_m(R_left, R_top) is TRUE then
29:                /* region merging */
30:                merge(R_left, R_top)
31:                remove(regions, R_top)
32:             end if
33:          end for
34:       end if
35:       if H_i(pixel) is TRUE then
36:          /* start new run with current pixel as first pixel */
37:          run_new = new_run(pixel)
38:          /* start new region with current run as first run */
39:          R_new = new_region(run_new)
40:          add(regions, R_new)
41:       end if
42:    end for
43: end for
44: return regions
```

run_{left} is considered to be complete. If its length is smaller than the pre-defined minimal length value $length_{min}$ (line 23), run_{left} as well as its associated region R_{left} are discarded by removing R_{left} from the set of heretofore identified regions (line 24). Otherwise, the algorithm proceeds with its *region merging* mechanism. For this purpose, all regions bordering run run_{left} at the top are identified. Subsequently, another iteration loop tries to identify every region R_{top} within $regions_{top}$ that can be merged with the run_{left} associated region R_{left} (lines 27-33) by again applying heuristic H_m. If H_m succeeds for two regions R_{top} and R_{left}, the regions are merged by appending all runs of region R_{top} to R_{left} (by simply changing pointers). Furthermore, region R_{left} is updated with respect to its average color, whereas region R_{top} is completely discarded by removing it from the set of heretofore identified regions.

Independent of the result of the previous region merging mechanism, a new run run_{new} with the current pixel as its starting position as well as a new region R_{new} with run_{new} as its first run are allocated. But only, if the current pixel is interesting. Last but not least, the region R_{new} is added to the set of heretofore identified regions.

In fact, the arrangement of the several conditional clauses are optimized in the final implementation in order to minimize their redundant invocations. However, the depicted algorithmic structure was chosen to ease understanding of the algorithm's overall functionality.

C. Feature Descriptor

For high level object detection processes, a convenient feature descriptor, which further reduces the visual data in comparison to the run-based representation, is required. Furthermore, it should incorporate a model of uncertainty that takes stochastic errors such as sensor noise into account. For that reason, we decided for an implicit representation in terms of statistical quantities.

By interpreting a region and its associated pixels as a two-dimensional Gaussian distribution in the image plane, a region can be implicitly described by means of statistical parameters, namely the two mean values m_x and m_y, the two variances σ_x^2 and σ_y^2, and the covariance σ_{xy}. A generalization of these specific parameters are the statistical moments. In this context, the two mean values correspond to the two moments of first order (m_{10} and m_{01}), whereas the two variances and the covariance correspond to the centralized (or central) moments of second order (μ_{20}, μ_{02} and μ_{11}):

$$\vec{m} = \begin{pmatrix} m_x \\ m_y \end{pmatrix} = \begin{pmatrix} m_{10} \\ m_{01} \end{pmatrix} \tag{3}$$

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} = \begin{pmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{pmatrix} \tag{4}$$

In the field of digital image processing, this statistical approach can be directly applied in terms of discretized moments [3]. In general, a discretized two-dimensional moment m_{pq} of order $p+q$ belonging to a region R is defined as

$$m_{pq} = \sum_{(x,y)\in R} x^p y^q, \tag{5}$$

meaning nothing but only the coordinates (x,y) that belong to R have to be considered for computing the sum. Furthermore, in our optimized approach, the required central moments of second order are efficiently computed by means of the moments up to and including second order:

$$\begin{aligned} \mu_{20} &= m_{20} - \frac{m_{10}^2}{m_{00}}, \\ \mu_{02} &= m_{02} - \frac{m_{01}^2}{m_{00}}, \\ \mu_{11} &= m_{11} - \frac{m_{10}\cdot m_{01}}{m_{00}}. \end{aligned} \tag{6}$$

Consequently, for representing a single region as a two-dimensional Gaussian distribution, our basic feature descriptor only comprises the following set M of moments:

$$\begin{aligned} M &= \{m_{pq} | p+q \leq 2\} \\ &= \{m_{00}, m_{10}, m_{01}, m_{11}, m_{20}, m_{02}\} \end{aligned} \tag{7}$$

For efficiently converting the region representation in terms of runs into the feature descriptor, the *Delta "δ" Method* [4] is applied by directly incorporating the numerical quantities of a run, namely the coordinates x_i and y_i of its starting pixel as well as its length l_i. For this purpose, let $S1_i$ and $S2_i$ be defined as follows:

$$\begin{aligned} S1_i &= \sum_{k=0}^{l_i-1} k = \frac{(l_i^2 - l_i)}{2} \\ S2_i &= \sum_{k=0}^{l_i-1} k^2 = \frac{l_i^3}{3} - \frac{l_i^2}{2} + \frac{l_i}{6} \end{aligned} \tag{8}$$

The required geometric moments m_{00_i}, m_{10_i}, m_{01_i}, m_{11_i}, m_{20_i} and m_{02_i} of a single run run_i are then computed in the following optimized way:

$$\begin{aligned} m_{00_i} &= l_i \\ m_{01_i} &= l_i \cdot y_i \\ m_{10_i} &= l_i \cdot x_i + S1_i \\ m_{02_i} &= l_i \cdot y_i^2 \\ m_{20_i} &= l_i \cdot x_i^2 + 2 \cdot S1_i \cdot x_i + S2_i \\ m_{11_i} &= l_i \cdot [l_i \cdot x_i + S1_i] = y_i \cdot m_{10_i} \end{aligned} \tag{9}$$

Finally, the moments of a region R correspond to the sums of the particular moments of all related n runs:

$$M = \{m_{pq} | m_{pq} = \sum_{i=1}^{n} m_{pq_i} \wedge p+q \leq 2\} \tag{10}$$

As shown by Prokop and Reeves [5], an adequate set of additional attributes can be derived from these moments

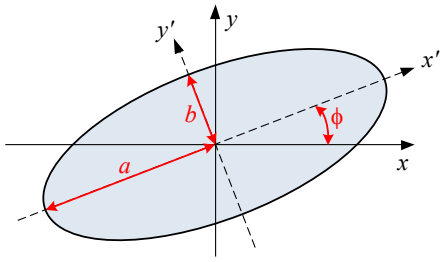


Figure 1. Representation of an ellipse by its major axis x', minor axis y' and angle of inclination ϕ.

(and central moments), in order to additionally describe a feature in a more explicit manner. In this context, the *mass* of a feature corresponds to the number of related pixels. It is, therefore, equivalent to the zeroth order moment m_{00}. Furthermore, the coordinates $(\bar{x},\bar{y})$ of the *center of mass* of a feature in the image plane are defined by means of the moments of zeroth and first order:

$$\bar{x} = \frac{m_{10}}{m_{00}} \quad \text{and} \quad \bar{y} = \frac{m_{01}}{m_{00}} \tag{11}$$

In addition, according to [6], by statistically describing a feature in terms of moments up to and including second order, the feature is equivalent to a *elliptical disk* (see Figure 1) with constant intensity, having definite size, orientation and eccentricity and being centered at the origin of the image plane. Its major radius a and minor radius b as well as its orientation ϕ (see Table I) can be in turn derived from the central moments in the following way:

$$a = \sqrt{\frac{2\left(\mu_{20}+\mu_{02}+\sqrt{(\mu_{20}-\mu_{02})^2+4\mu_{11}^2}\right)}{\mu_{00}}} \tag{12}$$

$$b = \sqrt{(\frac{2\left(\mu_{20}+\mu_{02}-\sqrt{(\mu_{20}-\mu_{02})^2+4\mu_{11}^2}\right)}{\mu_{00}}} \tag{13}$$

III. Feature Classification

The classification mechanism enables the user to define so called color classes for classifying pixels, thus features depend on their color. These color classes provide the basics for the heuristics H_i and H_m needed by the segmentation algorithm (cf. Section II-B).

A. Representation and Definition of the Color Classes

Color Classes in our context are nothing but an id representing a subspace within the working color space (i.e., YUV or RGB), which is in turn represented in terms of a cube (see Figure 2) with the particular color channels corresponding to the three dimensions. Each channel has a maximal resolution of 8bit, leading to $256\times256\times256$ possible combinations for addressing a position in the respective

Table I
Determining the angle of inclination ϕ as a function of the central moments of second order.

$\mu_{20}-\mu_{02}$	μ_{11}	ϕ	
0	0	0	
0	+	$+\pi/4$	$\xi = \frac{2\mu_{11}}{\mu_{20}-\mu_{02}}$
0	−	$-\pi/4$	
+	0	0	
−	0	$-\pi/2$	
+	+	$(1/2)\cdot\arctan\xi$	$(0<\phi<\pi/4)$
+	−	$(1/2)\cdot\arctan\xi$	$(-\pi/4<\phi<0)$
−	+	$(1/2)\cdot\arctan\xi+\pi/2$	$(\pi/4<\phi<\pi/2)$
−	−	$(1/2)\cdot\arctan\xi-\pi/2$	$(-\pi/2<\phi<-\pi/4)$

space. By realizing the color space as a look up table in terms of a one-dimensional array of integer values instead of a three-dimensional one, the number of memory accesses are minimized. For indicating, whether a coordinate within the color space is assigned to a color class with id i, the ith bit of the respective integer value is set to 1. Consequently, the number of possible color classes is bound to the chosen integer size (e.g., an 8bit integer allows the application of 8 different color classes). Additionally, by following this bit based approach, a coordinate can be simultaneously assigned to more than one color class at the same time. Furthermore, necessary mechanisms such as *inserting* and *removing* a color class as well as *testing* if a color tuple is assigned to a color class can be consequently implemented in terms of simple but very efficient bit operations.

The resolution of the channels can be restricted by defining a quantization factor $\lambda \in \{2^i | i = 1,2,\ldots,8\}$ in order to drastically reduce the over all memory consumption. Graphically, this means that the whole color space is no longer represented by $256\times256\times256$ different color combinations, but $256/\lambda\times256/\lambda\times256/\lambda$ statically allocated sub-cubes, where color combinations that are located within the same sub-cube are not further differentiated but considered as identical. A color class, thus in general is defined by a set of sub-cubes of size $\lambda\times\lambda\times\lambda$ in the color space, where sub-cubes may belong to several color classes as already men-

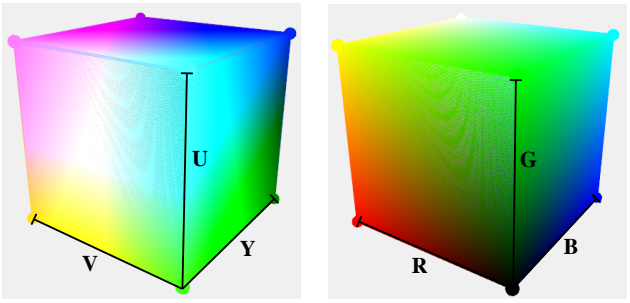


Figure 2. YUV and RGB color spaces, represented as three-dimensional cubes.

tioned above. As a consequence, the resolution and hence the granularity of the color space is significantly decreased, while the robustness of the overall feature classification is increased with respect to image noise. Dependent on the chosen quantization factor λ, the reduced granularity still leaves enough space for adequately distinguishing different colors.

Accessing the integer value of a color tuple (c_1, c_2, c_3) in the look up table *lut* is done by

$$lut[f(c_1, c_2, c_3)] \tag{14}$$

with

$$\begin{aligned} f(c_1, c_2, c_3) =& (c_1/\lambda) \cdot (256/\lambda)^2 \\ &+ (c_2/\lambda) \cdot 256/\lambda \\ &+ (c_3/\lambda) \end{aligned} \tag{15}$$

By only allowing powers of two as values for λ, the devision by λ can be replaced by a bit shift mechanism:

$$\begin{aligned} f(c_1, c_2, c_3) =& (c_1 \gg \lambda_s) \cdot (256 \gg \lambda_s)^2 \\ &+ (c_2 \gg \lambda_s) \cdot (256 \gg \lambda_s) \\ &+ (c_3 \gg \lambda_s), \end{aligned} \tag{16}$$

with

$$\lambda_s = log_2 \lambda. \tag{17}$$

The two multiplications can be also replaced as follows:

$$\begin{aligned} f(c_1, c_2, c_3) =& ((c_1 \gg \lambda_s) \ll c_{1s}) \\ &+ ((c_2 \gg \lambda_s) \ll c_{2s}) \\ &+ (c_3 \gg \lambda_s), \end{aligned} \tag{18}$$

with

$$c_{1s} = log_2((256/\lambda)^2) = log_2((256 \gg \lambda_s)^2) \tag{19}$$

and

$$c_{2s} = log_2(256/\lambda) = log_2(256 \gg \lambda_s). \tag{20}$$

Since λ_s, c_{1s} and c_{2s} have to be calculated only once whenever the quantization factor λ is changed (and hence the overall resolution of the color space), calculating the position as a function of a color tuple (c_1, c_2, c_3) for accessing the appropriate integer value in the look up table is very efficient.

In order to define a color class i as a subspace within the color space, the well known *Flood fill* [7] approach is applied. In this context, a user has to select a particular pixel in order to indicate the starting position in a camera image. Dependent on the configuration parameters, the algorithm recursively detects pixels with similar color in the adjacent neighborhood, taking each detected pixel as a new starting position. After termination, the color tuples (c_1, c_2, c_3) of all identified pixels are marked within the color space to be associated to color class i by setting bit i of the related sub-cube (i.e. of the integer value at the particular position in the one-dimensional array) to 1.

B. Heuristics for Segmentation

The necessary heuristics H_m for deciding whether a region growing or region merging step should take place and H_i for determining if a given pixel is interesting at all can now be easily constructed.

Let (c_1, c_2, c_3) be the color values of the current pixel. By applying (18), the associated position in the previously mentioned look up table can be computed and accessed. Then, by simply checking whether the value read from the memory is different to zero or not (i.e., if at least one bit is set), the heuristic H_i is able to determine, if the pixel is interesting or not:

$$\begin{aligned} H_i(pixel) &= H_i(c_1, c_2, c_3) \\ &= \begin{cases} 1 & if \quad lut[f(c1, c2, c3)] \neq 0 \\ 0 & else. \end{cases} \end{aligned} \tag{21}$$

For allowing that, e.g., two regions can be merged, their estimated average color values (c_1, c_2, c_3) and (c_4, c_5, c_6) have to be assigned to the same color class. For that purpose, the respective entries in the look up table have to be read and compared. Since the assignment of a color class in the look up table is done by activating bits, the comparison of the two integer values can be simply implemented by a bitwise *AND* operator (&) and a subsequent test for 0, meaning that there are no bits, which are set to 1 in both values. Hence, heuristic H_m can be realized as follows:

$$\begin{aligned} H_m &= H_m((c_1, c_2, c_3), (c_4, c_5, c_6)) \\ &= \begin{cases} 1 & if \quad \begin{array}{c} (lut[f(c1, c2, c3)] \\ \& \\ lut[f(c4, c5, c6)]) \neq 0, \end{array} \\ 0 & else. \end{cases} \end{aligned} \tag{22}$$

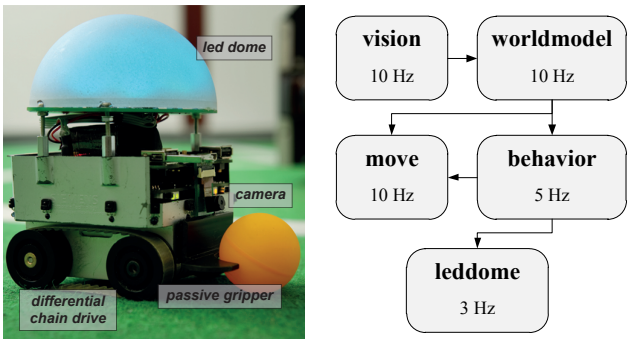
IV. Evaluation Scenario

For evaluating the functional principle of the presented approach, a simple real-world scenario that incorporates the miniature robot BeBot [1] was implemented.

A. BeBot Specifications

The BeBot is a miniature chain driven robot (shown in Figure 3(a)), which has been developed at the Heinz Nixdorf Institute of the University of Paderborn. Despite its small size (approximately 9 cm x 9 cm x 8 cm), it is equipped with modern technology distributed on two modular boards. The lower one of these boards implements basic functionalities such as controlling the motors by means of an ARM 7 based microcontroller. In contrast, the upper board constitutes a more powerful computing platform. Whereas a camera that is mounted at the front allows the BeBot to perceive its environment within a limited field of view, algorithms for convenient information processing steps can be executed in an embedded Linux environment running on an ARM Cortex-A8 CPU with a maximum frequency of 600 MHz

 ISBN: 978-1-61208-188-5



(a) (b)

Figure 3. (a) The miniature robot BeBot, pushing an orange ping-pong ball. (b) The node structure of the implemented behavior system.

accessing up to 256MB RAM of main memory. The BeBot is additionally equipped with a passive gripper, which enables the robot to push small objects such as ping-pong balls. Furthermore, by illuminating its so called led dome in arbitrary colors during runtime, the miniature robot can express its current state to human observers and other robots, respectively.

The network based TCP/IP communication is exclusively based on Bluetooth. For that reason, an approach that relies on sending raw visual data to external systems in order to apply computational expensive image processing algorithms is not feasible.

B. Scenario Description

The overall task of a BeBot in our evaluation scenario is to push a single-colored ping-pong ball through a slalom course, which consists of small pylons that are arranged in a straight line (see Figure 4 for a schematic illustration). For facilitating the identification of the start and the finish of the course, simple markers with a single color are attached to the respective points. Furthermore, the ping-pong ball is initially positioned somewhere around the course.

First of all, the robot has to search and catch the ball and return with it to the start point. Afterwards, it has to push the ball through the slalom course without loosing it. If the BeBot looses the ball, it has to start all over again. When successfully finished its drive through the course, the robot has to stop while blinking with its led dome.

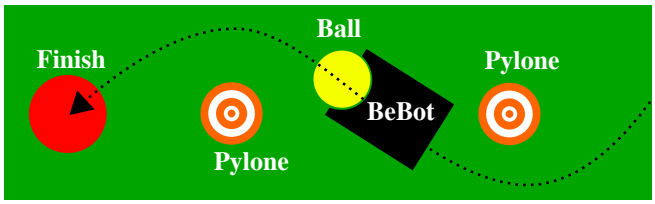


Figure 4. Schematic bird's eye view of the evaluation scenario.

C. Realization of the BeBot Behavior System

The node-based implementation of the BeBot's behavior system is depicted in Figure 3(b). The *vision* node incorporates all image processing steps including the presented segmentation and classification approach as well as a simple object detection mechanism in order to recognize the scenario's particular objects. The reduced visual information is subsequently transmitted to the *worldmodel* node, in which a very abstract representation of the BeBot's state within the environment is accumulated. The overall strategy is computed in the *behavior* node on a very abstract level. Those abstract behaviors (e.g., *DriveSlalom*, *CatchBall*) do not change quickly and do not contain any detailed information about how to achieve the respective goal. Furthermore, the connected *leddome* node changes the color of the led dome according to the current overall behavior. Finally, the low level behaviors (e.g., *MoveCurve*, *SearchBall*) that output the commands for the differential chain drive are implemented in the *move* node.

V. Results

The presented approach was completely implemented in C++ and was successfully applied to the BeBot within the scope of the evaluation scenario. The resolution of the images grabbed by the BeBot camera in the YUV color space was set to 320×240 pixels. Furthermore, the quantization factor λ was set to 16, resulting in a resolution of each channel in the YUV color space of $256/\lambda = 16$. Due to the setup of the scenario, the number of color classes were restricted to less then 8. For that reason, the look up table was initialized with 8bit integer values. Consequently, the overall memory consumption of the basic classification array was

$$(256/16 \cdot 256/16 \cdot 256/16) \cdot 1\,Byte = 4096\,Byte = 4\,kB \quad (23)$$

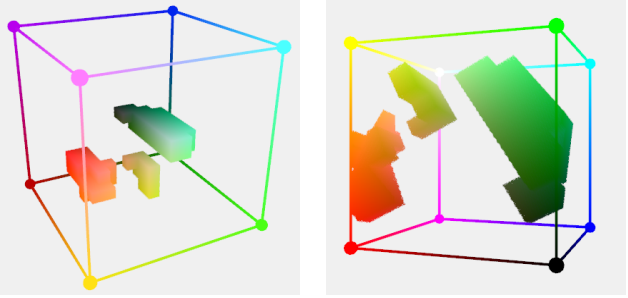
The subspaces corresponding to the different color classes within the working color space are depicted in Figure 5.

In order to demonstrate the efficiency of the introduced approach, a typical subjective view of the scenario setup (see Figure 6(a)) was selected for further experimental investigations. Within the scope of these experiments, the frequency of the upper board's ARM CPU was fixed to 520Mhz. Furthermore, the behavior system was disabled in order to run the image processing as standalone process.

Four different test cases were considered. The particular results are shown in Table II. In this context, case *A* solely contains the image acquisition step as evaluation basis. In addition, case *A* was used for identifying the maximum possible frame rate (16 fps) currently supported by the BeBot.

Case *B* incorporates a modified version of the presented segmentation approach: the classification mechanisms were completely discarded. Consequently, all pixels were supposed to be of interest and therefore considered during the

 ISBN: 978-1-61208-188-5



(a) YUV (b) RGB

Figure 5. (a) Representation of the different color classes (pylones, ball, marker) in the YUV color space, in which the whole image processing takes place. The influence of the quantization factor λ can be seen in terms of the topology of the particular subspaces, which consist of small cubes. Additionally, (b) depicts the defined color classes transformed into the RGB color space.

segmentation step. Furthermore, the original heuristic H_m (22) for deciding whether a region growing or region merging step should take place was replaced by a heuristic, which determines, whether two color tuples reside in a predefined neighborhood within the YUV color space. Figure 6(b) shows the processed image as well as the extracted features in terms of their respective centers of mass and ellipses.

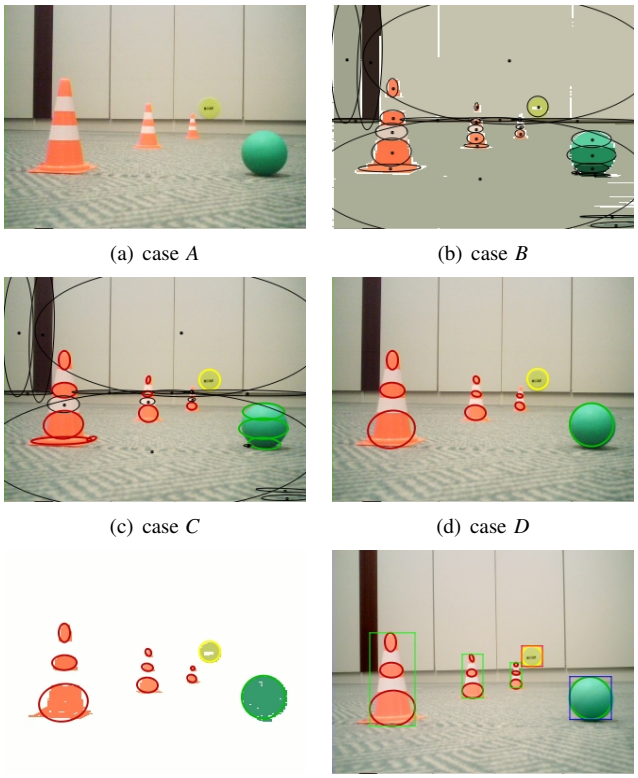
Based on the classification independent segmentation in case *B*, case *C* additionally incorporates the presented classification approach. However, the classification was not applied during the segmentation process on pixel level, but as a subsequent processing step on feature level. The classified features can be seen in Figure 6(c) in terms of their colored ellipses, where the color of an ellipse represents the related color class.

The advantages of the original approach (case *D*), where the classification heuristics H_i and H_m are applied during segmentation as described in Algorithm II-B, become obvious by comparing the performance values as well as the results that are depicted in Figure 6(d) and Figure 6(e) with the previously mentioned results. Since all redundant pixels that are not of interest are already discarded during the segmentation step, the amount of visual data is drastically reduced and only scenario relevant features are extracted. As a consequence, the workload of the CPU is significantly decreased. Furthermore, since only pixels that belong to the same color class are merged, regions being more proper are constructed during the segmentation step (compare, e.g., the three extracted and classified features of the ball in Figure 6(c) with the one feature of the ball in Figure 6(d)).

Table II
PERFORMANCE VALUES OF THE FOUR DIFFERENT TEST CASES.

Case	Related figures	CPU	Mem.	P. time	#F	fps
A	Fig. 6(a)	4%	8.6%	-	-	16
B	Fig. 6(b)	79%	10.3%	46.7 *ms*	29	16
C	Fig. 6(c)	80%	10.5%	47.1 *ms*	29	16
D	Fig. 6(d), Fig. 6(e)	29%	10.5%	15.3 *ms*	10	16

P. time: Average processing time (ms).
#F: Average number of extracted features.
fps: Frames per second.



(a) case *A* (b) case *B*

(c) case *C* (d) case *D*

(e) case *D* (f)

Figure 6. A BeBot's typical subjective view of the scenario setup: (a) original image, (b) result of image segmentation without incorporating classification, (c) subsequent feature classification, (d) original and (e) result image of the combined approach, (f) detected scenario relevant objects.

Finally, Figure 6(f) shows a representative result of the entire image processing (presented approach + object detection) within the scope of the evaluation scenario. While the classified features are again represented by means of their colored ellipses, the identified scenario relevant objects are represented by means of their bounding boxes.

VI. RELATED WORK AND DISCUSSION

A similar color-based segmentation and classification approach was presented in [8]. However, in their context, an image has to be processed in several steps. After an optional step of color space projection, each pixel is classified by one of up to 32 color classes. In comparison to our classification mechanism, the associated subspaces within the working color space can only take shape of convex cuboids, whereas our subspaces may also take shape of concave structures or either may consist of two and more unconnected subspaces.

Afterwards, each row is processed again in order to encode adjacent pixels of the same color class into runs.

 ISBN: 978-1-61208-188-5

Finally, connected runs belonging to the same color class are efficiently grouped into regions by means of a tree-based union find algorithm, whereas the constructed regions are represented in terms of their bounding boxes, centroids and sizes. Although the basic concepts are similar to our approach, there are significant differences with respect to the number of processing steps, the flexibility in representing color classes, the method and point of time of grouping runs as well as the robustness of the actual feature descriptor. In the latter case, especially the bounding boxes are error prone and cannot appropriately compensate the negative effects of stochastic errors such as image noise.

Another quite similar approach was presented in [9]. Whereas the classification mechanism is identical to the previously discussed one, the number of image processing steps are reduced to two. At first, similar to our approach, the image is compressed by comparing adjacent pixels with respect to their associated color classes and by encoding each contiguous block of pixels into a run. After completing a run, neighboring runs belonging to the same color class are identified and marked as connected. However, in comparison to our approach, they are not yet merged. For increasing the performance of the algorithm, pixels that are not of interest are discarded, just as our segmentation algorithm does. In the second step, a recursive depth-first traversal is performed in order to identify runs that were marked as connected and to assign them to the same feature, which is in turn represented by its center of gravity efficiently derived from the runs.

A fast component labeling and description algorithm that also bases on run length was introduced in [10]. Similar to our approach, the main idea is to scan an image from left to right row by row starting at the topmost row in order to construct runs of connected pixels. However, only binary thus already segmented images are considered. Consequently, in comparison to our approach, the entire image has to be processed at least twice in order to identify and extract regions. Furthermore, again, no robust feature representation in terms of statistical moments is derived from the grouped runs, but only the area, the center and the bounding box.

VII. Conclusion

In this paper, an efficient color-based image segmentation and feature classification approach was presented. The entire image has to be processed only once in order to identify regions by means of an efficient classification mechanism. During the segmentation process, regions are compactly represented in terms of runs in order to drastically reduce the overall memory consumption. Furthermore, the computational effort for constructing and merging regions is significantly reduced. The classification approach additionally provides an efficient mechanism for deciding whether a pixel is relevant for the segmentation process or not. As a consequence, redundant visual data is already discarded on pixel level. A second and final step efficiently converts the extracted regions into a convenient and robust feature representation in terms of statistical moments by directly incorporating the previously assembled runs. The presented results show the functional correctness of our combined approach as well as its efficiency in comparison to a sequential one. The presented approach constitutes a portable and flexible solution for any Linux based embedded system in order to overcome the data intensive and computationally extremely expensive task of image processing.

Acknowledgment

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901).

References

[1] S. Herbrechtsmeier, U. Witkowski, and U. Rückert, "Bebot: A modular mobile miniature robot platform supporting hardware reconfiguration and multi-standard communication," in *Progress in Robotics*. Springer Berlin Heidelberg, 2009, vol. 44, pp. 346–356.

[2] R. Jain, R. Kasturi, and B. G. Schunck, *Machine vision*. McGraw-Hill, 1995, (verified: 10.01.2012). [Online]. Available: http://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.htm

[3] M.-K. Hu, "Visual pattern recognition by moment invariants," *IEEE Transactions on Information Theory*, vol. 8, no. 2, pp. 179–187, 1962.

[4] M. F. Zakaria, L. J. Vroomen, P. J. A. Zsombor-Murray, and J. M. H. M. van Kessel, "Fast algorithm for the computation of moment invariants," *Pattern Recogn.*, vol. 20, no. 6, pp. 639–643, 1987.

[5] R. J. Prokop and A. P. Reeves, "A survey of moment-based techniques for unoccluded object representation and recognition," *CVGIP: Graph. Models Image Process.*, vol. 54, no. 5, pp. 438–460, 1992.

[6] M. R. Teague, "Image analysis via the general theory of moments," *Journal of the Optical Society of America (1917-1983)*, vol. 70, pp. 920–930, 1980.

[7] L. Vandevenne, "Lode's computer graphics tutorial - flood fill," (verified: 10.01.2012). [Online]. Available: http://lodev.org/cgtutor/floodfill.html

[8] J. Bruce, T. Balch, and M. Veloso, "Fast and inexpensive color image segmentation for interactive robots," in *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2000, pp. 2061–2066.

[9] C. Messom, S. Demidenko, K. Subramaniam, and G. Gupta, "Size/position identification in real-time image processing using run length encoding," in *Proceedings of the 19th IEEE Instrumentation and Measurement Technology Conference (IMTC)*, 2002, pp. 1055–1059.

[10] L. Qiu and Z. Li, "A fast component labeling and description algorithm for robocup middle-size league," in *7th World Congress on Intelligent Control and Automation (WCICA)*, 2008, pp. 6575–6579.

 ISBN: 978-1-61208-188-5