

A Connected Domain Analysis Based Color Localization Method and Its Implementation in Embedded Robot System

Fei Guo

School of Information Engineering, Zhengzhou University, Zhengzhou, China
Email: gallonfr@gmail.com

Ji-cai Deng and Dong-bo Zhou

School of Information Engineering, Zhengzhou University, Zhengzhou, China
Email: iejcdeng@zzu.edu.cn

Abstract—A target localization method based on color recognition and connected component analysis is presented in this paper. The raw image is converted to HSI color space through a lookup table and binarized, then followed by a line-by-line scan to find all the connected domains. By setting appropriate threshold for the size of each connected domain, most pseudo targets can be omitted and coordinates of the target could be calculated in the mean time. The main advantage of this method is the absence of extra filtering process, therefore real-time performance of the whole system is greatly improved. Another merit is we introduce the frame difference concept to avoid manually presetting the upper and lower bound for binarization. Thirdly, the localization step is combined with target enumeration, further simplified the implementation.

Experiments on our ARM system demonstrate its capability of tracing multiple targets under a mean frame rate of 15FPS, which satisfied the requirement of real-time video processing on embedded robot systems.

Index Terms—connected domain, color recognition, target localization, embedded system

I. INTRODUCTION

In many computer vision systems, color recognition always proves to be a significant means of acquiring target or environment information [1]. As a typical characteristic of visual objects, color information can be used to locate targets independently [2][3]. However, like almost all the low-level image features, color feature is also interfered greatly by noise. Although plenty of methods have been proposed, complicated filtering algorithms yield high requirement for hardware computational capability and a common embedded processor could barely meet [4]. Especially in terms of embedded systems, besides image processing, the processor also needs to take charge of other tasks such as movement scheduling. When a control algorithm is relatively sophisticated, system resources will soon be exhausted and result in an unpredictable system failure. Hoping to avoid the complexity, we intend to adopt a simple and efficient approach on the premise of a proper positioning accuracy. In general, most classic denoise

techniques can be ascribed to some sort of mask-based transformation sliding over the whole frame, which would involve all the pixels repeatedly hence inducing large amount of computation. At this point, a basic motive is to find a way of locating target without particular denoise filtering.

To begin with, it is indispensable to analyze the distribution feature of image noise. Considering the general situation, employing Gaussian model to describe noise distribution is a common manner. Moreover, the output of color recognition is accompanied with a binarization process. When the binarization process is applied with Gaussian model, it will pose a polarized probability distribution function [5]:

$$p(z) = \begin{cases} P_0 & , z = 0 \\ P_1 & , z = 1 \\ 0 & (\text{actually no}) \end{cases} \quad (1)$$

That is to say, the original Gaussian model will transform to a certain salt-and-pepper noise model. On that basis, if make an assumption that the probability of noise occurrence for each pixel is independent, the probability of consecutive noise occurrence can be ignored:

$$\lim_{N \rightarrow \infty} \prod_{i=1}^N P_{li} = 0 \quad , 0 < P_{li} < 1 \quad (2)$$

Following the discussion, we can develop a conclusion that a connected domain with only few pixels should be regarded to be noise, and hence by counting the pixels within regarded target areas and excluding the fewer ones, noise pixels would be suppressed notably. Furthermore, while calculating the centroid, pixel-counting is also needed; therefore both procedures can be merged without introducing extra computation.

After briefing several basic concepts above, the proposed method is summarized as follow: First, find a proper threshold from differential image for color recognition. Second, binarize the image according to this threshold. Then, labeling the image and calculate the size of each connected domain. The following verifying process will remove most noise pixels, and in the

meantime, prepare necessary parameters for subsequent step. Finally, compute the centroid of each color domain coordinates and mark them on the display.

II. PROPOSED METHOD

The entire algorithm is comprised of three main steps shown as below: color recognition, connected domain computation, as well as positioning and marking.

A. Color recognition

Color recognition is a typical classification problem. That is to say, according to some certain standards, we divide the whole color set into several subsets and each subset is defined as a color. For now, we often employ the term "color space" to describe the abstract mathematical model suggesting that colors can be represented as tuples of numbers, typically as three or four values or color components, such as the most used RGB color space. By checking the scope of each component, a color can be classified into predefined subsets.

Yet different color space models possess a variety of features. The RGB color space is influenced by luminance tremendously [6]. Table I. shows that little change in luminance (I) would bring on certain change in RGB components.

TABLE I. RGB VERSUS HSI

R	G	B	H	S	I
40	40	220	120	60	100
36	36	198	120	60	90
32	32	176	120	60	80

Actually, human eyes will recognize all the three color as dark green even they boast obvious difference mathematically. Therefore choosing an appropriate color space is the first thing to fix. Alike the HSI color space, YUV color space also separate the luminance from other components. Yet, the implementation in YUV color space is relative complicated [7], thus our adoption of HSI color space here.

As we know, in HSI color space, sole Hue component is able to represent different color types individually. Nevertheless, when the Saturation component is lesser, validity of the Hue component would decrease, hence categorizing a color with at least both Hue and Saturation components is plausible. What is noteworthy is the conversion from RGB to HSI is a nonlinear floating point operation; the normalized formula is given below.

Normalize the raw RGB data:

$$\begin{aligned} r &= \frac{R}{R+G+B} \\ g &= \frac{G}{R+G+B} \\ b &= \frac{B}{R+G+B} \end{aligned} \quad (3)$$

Calculate each component:

$$\theta = \cos^{-1} \left\{ \frac{\frac{1}{2}[(r-g)+(r-b)]}{\sqrt{(r-g)^2 + (r-b)(g-b)}} \right\} \quad (4)$$

$$h = \begin{cases} \theta & h \in [0, \pi], \text{ for } b \leq g \\ 2\pi - \theta & h \in [\pi, 2\pi], \text{ for } b > g \end{cases}$$

$$s = 1 - 3 \cdot \min(r, g, b) \quad s \in [0, 1]$$

$$i = (R + G + B) / (3 \cdot 255) \quad i \in [0, 1]$$

Range conversion:

$$H = h \times 180 / \pi$$

$$S = s \times 100 \quad (5)$$

$$I = i \times 255$$

If applying these formulas to each pixel, definitely it should exert a heavy load to the system hardware. To the embedded system whose resources are limited, this would pose an impossible task, especially for the real-time processing. An optional way to resolve the problem is importing a lookup table (LUT) for the fast conversion. Creating the LUT is rather simple and direct: generating a 3-D array whose indexes correspond to the three RGB components, each cell comprises pre-calculated HSI components. Such as (refer to Table I.):

$$\text{LUT}[40][40][220].H = 120$$

$$\text{LUT}[40][40][220].S = 60$$

$$\text{LUT}[40][40][220].I = 100$$

However, creating LUT according to the original 256 gray-levels consumes large amount of memory, which encourage us to use less gray-level. Considering another fact that we choose a 16bpp LCD in the hardware system, following the 16bpp format, also known as RGB565 format, would be a wise choice, simply because there is no need to add in an extra conversion between LUT accessing and image display.

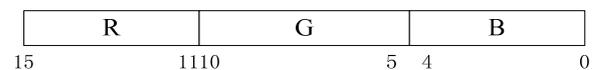


Figure 1. RGB565 format.

In RGB565, the Red and Blue components are represented by 5-bit binaries (32 gray-levels), while the Green component is represented by a 6-bit binary (64 gray-levels). Conversion from standard 256 gray-level (8-bit length) to these formats could be reached through integer division or logical shift. The later operation is preferred because of slightly higher efficiency, since logical instructions often use less time to get the final result. But we should note that both integer division and logical shift will omit the fraction part of quotient, which raises higher conversion error. Table II. demonstrates the integer conversion error between 256 gray-level and 32 gray-level, which is similar to other bit-lengths.

TABLE II. INTEGER CONVERSION ERROR

raw gray-level	convert to 5-bit length	restore to 8-bit length	error
128	16 (16)	128	0
132	16 (16.5)	128	4
135	16 (16.875)	128	7
136	17 (17)	136	0

For the last row of the table, “135” is the most closed value to “136”; rather than the 5-bit gray-level “16”, it should be converted to “17” to minimize the conversion error variance. So we need an adjustment to the dividend as to obtain a rounded value. Equation (6) defines the half correction

$$I^* = \frac{I + 2^{7-b}}{2^{8-b}} = (I + 2 \ll (6-b)) \gg (8-b) \quad (6)$$

, where I is the original 256-level grayscale, I^* is the rounded objective grayscale, b means the desirable bit length. For the example given above, 8-bit gray-level “135”, it would be:

$$l = 5 \text{ for 5-bit grayscale} \\ I^* = (135 + (2 \ll 1)) \gg 3 = 139 \gg 3 = 17 \quad (7)$$

To successfully classify a color, the next problem is how to ascertain a suitable range for each component, or to say upper and lower bound. Hoping to enhance the adaptability, a thresholding method based on frame difference (FD) is affiliated to this section. Since the system is designed to track the motion of color target, classical FD method [8] could be used to roughly locate the region of interest (ROI). By examining the occurrence probability of ROI pixels, we can obtain a fitting range for binarization. The following flowchart could help to understand the technique.



Figure 2. Thresholding flowchart for color recognition

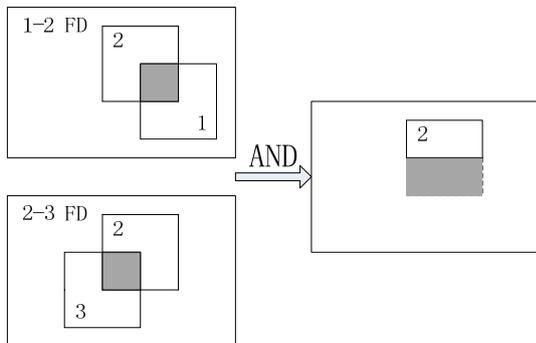


Figure 3. Using FD to find ROI.

As our interest is to find an available color region within the motion target, therefore no need to wholly

fulfill the method proposed in Ref. [8] which includes a gray-level gradient calculation. Directly compute the two FD images with logical AND would meet our requirement. In Fig. 3, the upper white rectangle region enclosed by solid line is supposed to be our ROI. Color distribution within that area could be utilized to calibrate our threshold for color recognition processing. Take the gray-level image for instance; assume the typical histogram of ROI to be:

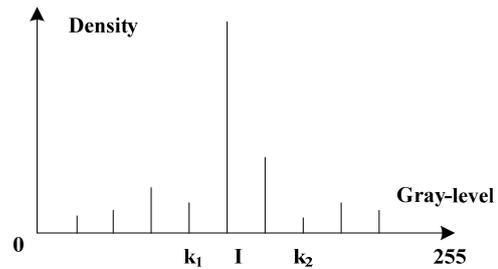


Figure 4. typical histogram of ROI

Since most pixels within ROI concentrate around a solo intensity point (“I” in this example), what we need to do is find k_1 and k_2 to separate the axis into three classes. Otsu in his article Ref. [9] suggested a well-known method for multi-thresholding. Considering the three classes to be C_0 for $[1, \dots, k_1]$, C_1 for $[k_1 + 1, \dots, k_2]$ and C_2 for $[k_2, \dots, L]$, the number of pixels at gray-level i is denoted by n_i and the total number of all the pixels is donated by $N = n_1 + n_2 + \dots + n_L$, the probability distribution would be:

$$p_i = \frac{n_i}{N}, p_i \geq 0 \text{ and } \sum_{i=1}^L p_i = 1 \quad (8)$$

The class occurrence probabilities are given by

$$\omega_k = P(C_k) = \sum_{i \in C_k} p_i, k = 0,1,2 \text{ and } \sum_{k=0}^2 \omega_k = 1 \quad (9)$$

Mean levels of each class are

$$\mu_k = \sum_{i \in C_k} i \cdot P(i | C_k) = \sum_{i \in C_k} i \cdot \frac{p_i}{\omega_k} \quad (10)$$

Class variance is given by

$$\sigma_k^2 = \sum_{i \in C_k} (i - \mu_k)^2 \cdot P(i | C_k) = \sum_{i \in C_k} (i - \mu_k)^2 \cdot \frac{p_i}{\omega_k} \quad (11)$$

The total mean level of ROI and total variance are

$$\mu_k = \sum_{i=1}^L i \cdot p_i \\ \sigma_T^2 = \sum_{i=1}^L (i - \mu_T)^2 \cdot p_i \quad (12)$$

And the most important three criteria are respectively defined as

$$\lambda = \frac{\sigma_B^2}{\sigma_W^2}, \quad \kappa = \frac{\sigma_B^2}{\sigma_W^2}, \quad \eta = \frac{\sigma_B^2}{\sigma_T^2} \quad (13)$$

where

$$\sigma_W^2 = \sum_{k=0}^2 \omega_k \cdot \sigma_k^2 \quad (14)$$

$$\sigma_B^2 = \sum_{k=0}^2 \omega_k \cdot (\mu_k - \mu_T)^2$$

Given that σ_W^2 is based on the class variance, a second-order statistics, while σ_B^2 is base the first-order statistics, mean level, and all these three criteria are equivalent, η is chosen to evaluate the satisfaction of thresholds due to the simpler computational complexity. Besides, σ_T^2 is a constant value for each ROI, therefore by maximizing the criterion of between-class variance:

$$\sigma_B^2(k_1^*, k_2^*) = \max_{1 \leq k_1 < k_2 \leq 255} \sigma_B^2(k_1, k_2) \quad (15)$$

, we can get a set of fitting thresholds k_1^* and k_2^* .

If apply this method to the Hue and Saturation components of pixels within ROI, surely we can get the upper and lower bound for color recognition. Taking into consideration the fact that most factual targets seldom change their surface color, we can even put the thresholding route ahead of all the procedures as initialization. In this manner, we don't have to change the thresholds frequently on behalf of system efficiency. The final routine could be summarized as:

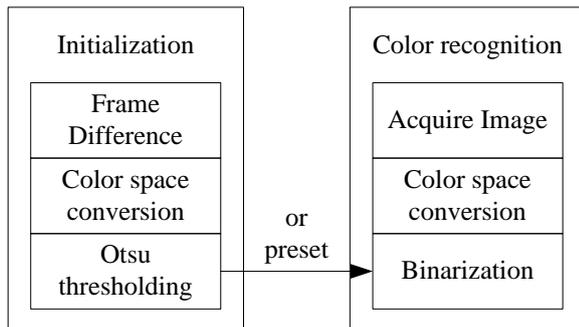


Figure 5. Full flowchart of color recognition.

B. Connected domain computation

Based on previous work, we can separate a binarized image into several connected domains. Presented labeling algorithms can be classified to three categories based on the methodology adopted for the scan over the images, i.e. multi-scan, two scan and contour tracing [10]. Multi-scan techniques [11] have lower efficiency compared to the other two classes. Contour tracing algorithm [12] indeed boasts a higher speed, which includes two steps of external contour tracing and internal contour tracing. After doing some research on the noise pixels from

gathered frames, we find that most stains are solid disk-like simple connected domains. Note that the run process searching for internal tracing is of no practical use in some sense; furthermore, calculation for centroid in such a case is more complex, a two scan based approach is adopted.

The first scan is made by lines to fill an equivalence table, which depicts the starting point and end point of each line segment in the raw image, i.e. handling the horizontal connectivity. Structure of the equivalence table is:

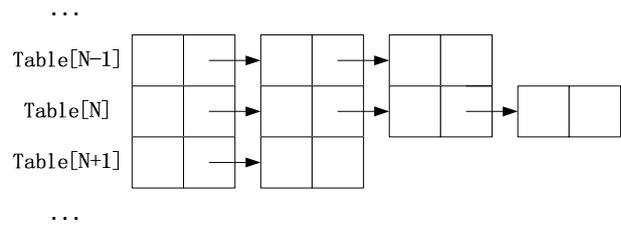


Figure 6. Structure of equivalence table

In this table each node of the linked list stands for a line segment. The second scan is made against this table. By checking the connectivity of line segments from two neighboring lines, it can be decided whether to create a new domain or merge two existing one. When the whole table is parsed, the searching is over and all the domains are found. Principle pseudo code is given below:

```

cur_cursor=table[i].entry
a:
pre_cursor=table[i-1].entry
b:
if ( pre_cursor == null ){
    area=new area
    area.next_node=cur_cursor
}
else{
    if ( pre_cursor ∈ N8( cur_cursor ) ){
        if ( cur_cursor.area_head != null ){
            if( cur_cursor.area_head !=
                pre_cursor.area_head )
                merge( cur_area , pre_area )
        }
        else{
            pre_area.next_node=cur_cursor
        }
    }
    pre_cursor=pre_cursor.next
    if ( pre_cursor != null )
        loop b
    if ( cur_cursor.area_head == null ){
        area=new area;
        area.next_node=cur_cursor
    }
    cur_cursor=cur_cursor.next
    if ( cur_cursor != null )
        loop a

```

An example is given to illustrate the whole procedure:

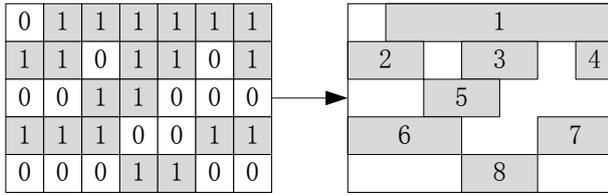


Figure 7. Result of the line scan

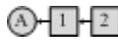
1st line:

pre_cursor is null and *cur_cursor* points to line segment [1,6,6,0], i.e. [start, end, length, line index]; so create a new area *A* and attach the line segment to it:

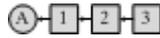


2nd line:

pre_cursor points to "1" and *cur_cursor* points to "2"; they are 8-connected and "2" hasn't been linked to any area; so attach it to area *A*, which "1" belongs to:



pre_cursor points to "1" and *cur_cursor* points to "3"; they are connected and "3" hasn't been linked to any area; so attach it to *A*:



same thing to do when *cur_cursor* points to "4":



3rd line:

pre_cursor points to "2" and *cur_cursor* points to "5":



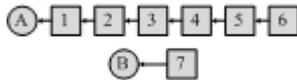
pre_cursor points to "3" and *cur_cursor* points to "5"; a little different here because "5" has already been linked to *A* and share the same area with "3", so nothing is done until the next line.

4th line:

pre_cursor points to "5" and *cur_cursor* points to "6":

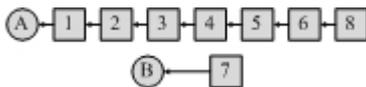


pre_cursor points to "5" and *cur_cursor* points to "7"; "7" isn't connected with any line segment (here we refer to "5") in the 3rd line; so create a area *B* for it:

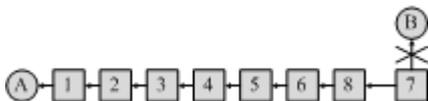


5th line:

pre_cursor points to "6" and *cur_cursor* points to "8":



pre_cursor points to "7" and *cur_cursor* points to "8"; they are connected but belong to different area, so merge *B* into *A*:



In this manner, we get a single connected domain marked as "A". In contrast with the method proposed in [13], based on tree structure, a simpler linked list

structure is employed here, which reduces operations in adding nodes and releasing memory.

C. Positioning and Marking

The main purpose of color locating is to obtain target coordinates and mark them. As discussed in the introduction, if pixels of a connected domain are too few, it's reasonable to regard it as a pseudo target. Coordinates can be calculated by the formula below:

$$x = \frac{\sum_{i=1}^N x_i}{N} \quad (16)$$

$$y = \frac{\sum_{i=1}^N y_i}{N}$$

, where x_i and y_i represent the horizontal and vertical coordinates of each pixel in one same domain, N is count of pixels. To highlight a valid target, drawing a rectangle box on the raw image would be all right, that is to ascertain the upper and lower bounds for both directions.

When in implementation, necessary parameters could be computed in the meantime of linking line segments and merging areas. For instance, while linking line segment "6" to area *A*, Table III. exemplifies the parameter updating procedure:

TABLE III. UPDATING PARAMETERS

	Before			After
	A		"6"	A
left	0	=	0	0
right	6	>	2	6
top	0	<	4	0
bottom	3	<	4	4
count	13	+	3	16
sigma_x	40	+	3	43
sigma_y	9	+	9	18

Like so all the required parameters could be fixed just when an area is determined, and then we can derive the final result.

III. EXPERIMENT AND RESULT

A. Experiment Platform

We made our real-time experiment on an ARM system based on S3C2440, whose working frequency is 400MHz, with an external memory configuration of 64MB SDRAM working at 100MHz. Besides standard UART and Ethernet interface, a 3.5-inch LCD is also applied to display result. A common webcam (Sonix) is used as image acquisition device. Fig. 8 shows the system diagram of our experiment platform.

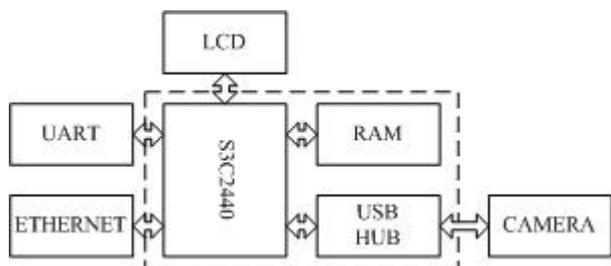


Figure 8. Hardware diagram

The Operating System is Linux and kernel version is 2.6.32 (V4L2 and UVC enabled). Despite that the camera can support YUYV mode, the CPU only provide a USB1.1 port, which means a compulsory MJPEG mode due to the restriction of lower bandwidth. However, performance of inexpensive webcam is often limited, whether the data are compressed won't affect the result too much. Yet extra decoding process is needed, which considerably increases the burden of our system. Flowchart of the entire system is shown as Fig. 9:

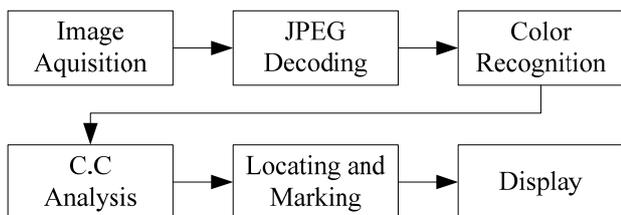


Figure 9. System flowchart

B. Results and discussion

In order to testify the effectiveness of our approach, experiments are made under the following configuration:

- Camera sample rate is 25FPS
- Sample and Display resolutions are both 320*240
- Preset target color is pink (Hue range from 310 to 325; Saturation is not checked)
- Pixel count threshold is 64 (nearly millesimal of total area)

The marking example is given below:

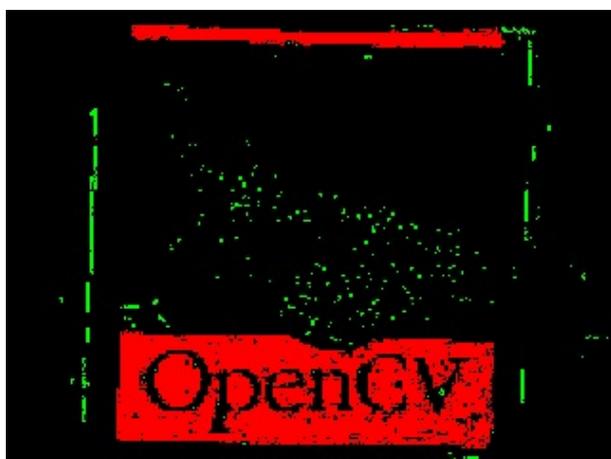


Figure 10. Binarization result

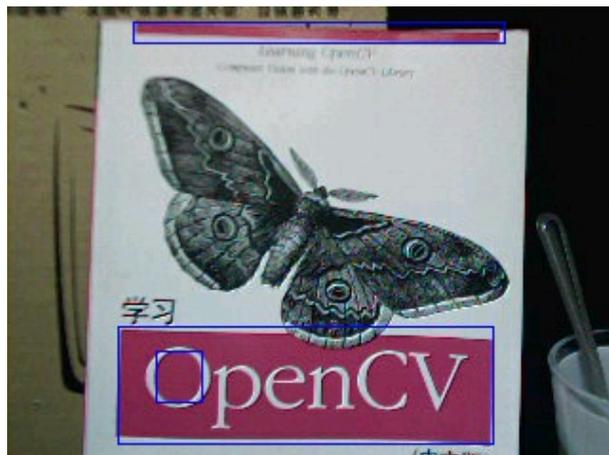


Figure 11. Marking result

After several tests, we get an output at a rate of 15 FPS on average, which meets our requirement of real-time processing. Fig. 11 illustrates a raw frame gathered from the camera and the blue rectangles marked three target color regions. Upper region is a simple connected domain, while lower regions comprise one simple connected domain (Capital letter "O") and one complex connected domain. Fig. 10 demonstrates the result of binarized image, in which red pixels indicate regarded color region. Green pixels represent noise pixels which are recognized as "pink" yet exempted by checking the count threshold. Obviously, noise pixels located within the butterfly figure and those who surround the edge of the book are inhibited effectively.

Besides above experiment using preset color threshold, we also test the adaptive threshold approach. By monitoring a ping-pong ball painted in several different colors, we get a set of threshold value. Meanwhile, this motion sequence was dumped and marked manually under the standard of maximizing the marking similarity. By checking the output of auto thresholding, we can tell whether our system could work without human interference. Since the marked image could only prove our locating approach, we draw a data table here to compares the two groups of result:

TABLE IV. AUTO AND MANUAL COMPARISON

Hue	Auto thresholding		Manual Marking	
Red	2	19	0	22
Green	108	125	105	129
Blue	235	247	233	252
Pink	311	328	310	325
Orange	23	36	19	34

From the table above, we can notice that our adaptive mechanism successfully realize its intension.

IV. CONSLUION

In this article, we proposed a color localization method based on connected domain analysis. It improves the efficiency by excluding additional filtering procedure. Experiments on our ARM system proved its lower computational complexity and effectiveness of noise reduction. Also, by introducing an adaptive thresholding route, it can track most targets holding relatively obvious chromatic feature without any prior knowledge or human calibration.

Admittedly, in many occasions such as targets with complex chroma composition, the thresholding approach won't work well. Because the histogram of ROI boasts several peak values, attempting to divide all the pixels into only three classes couldn't represent the real case. Moreover, for objects whose color may change, the system also would fail to make self-adjustment to fit the new situation. Even if we brought the adaptive thresholding into each frame, large amount of computation would still act as an insurmountable obstacle.

This system has already been applied to our robot contest, used to recognize the robot soccer. Our future research will focus on multi-feature recognition and simplifying algorithm, hoping to balance the complexity and efficiency.

ACKNOWLEDGMENT

The research was done thanks to financial support of the Natural Science Foundation project 2011B510017 of Education Department of Henan Province of China "Research on visual processing system of intelligent mobile robot". The authors wish to thank all the reviewers for their valuable comments and suggestions that improved the quality of this paper.

REFERENCES

- [1] R.K. McConnell, When trees are not green: Recent developments in an off-the-shelf system for robust color and multispectral based recognition and robot control. Proc. TePRA-2009 , p.204-209,2009.
- [2] Y. Kuno, K. Sakata, Y. Kobayashi, Object recognition in service robots: Conducting verbal interaction on color and spatial relationship. Proc. ICCV Workshops, p.2025-2031, 2009.
- [3] G. Yasuda, Bin Ge. Color based object recognition, localization, and movement control for multiple wheeled mobile robots. IECON 2004, p. 395- 400,2004.
- [4] B. Browning, M. Veloso. Real-time, adaptive color-based robot vision. IROS 2005, p. 3871- 3876,2005.
- [5] R.C. Gonzalez, R.E. Woods. Digital Image Processing, 2nd Edition, Prentice-Hall,2002.
- [6] Gibson, J.J., On constant luminance, gamma correction, and digitization of color space, Consumer Electronics, Proceedings of International Conference, pp.178-179, 7-9 Jun1995
- [7] P. Vadakkepat, P. Lim, L.C. De Silva, Jing Liu, Li Li Ling. Multimodal Approach to Human-Face Detection and Tracking. IEEE Transactions on Industrial Electronics, 55(3): 1385-1393, 2008.
- [8] C. Vieren, F. Cabestaing, J. G. Postaire, Catching moving objects with snakes for motion tracking, Pattern Recognition Letters, Vol.16, No.7, 679-685, 1995.
- [9] Otsu, Nobuyuki, Threshold Selection Method from Gray-Level Histograms, IEEE Transactions on Systems, Man and Cybernetics,, vol.9, no.1, pp.62-66, Jan. 1979.
- [10] C. Grana, D. Borghesani, R. Cucchiara. Optimized Block-Based Connected Components Labeling With Decision Trees. IEEE Transactions on Image Processing , 19(6): 1596-1609, 2010.
- [11] K. Suzuki, I. Horiba, N. Sugie. Linear-time connected-component labeling based on sequential local operations. Comput. Vis. Image Underst., 89(1)1-23, 2003.
- [12] Fu Chang, Chun-Jen Chen. A component-labeling algorithm using contour tracing technique. Proc. ICDAR, p. 741- 745, 2003.
- [13] Xiao Tu, Yue Lu, Run-Based Approach to Labeling Connected Components in Document Images. Proc. ETCS, p.206-209, 2010.

Fei Guo received his Master's degree in Measure & Control Technology and Instrumentation from the School of Electronic Information in Wuhan University in 2005. He is now a sophomore graduate student majoring in Circuit and System in the School of Information Engineering. His research interests mainly focus on image processing, pattern recognition and artificial neural network and he also boasts a good background in embedded system.

Ji-cai Deng was born in the city of Anyang, Henan province, P.R.China. He received his PhD degree in Physics from Zhengzhou University, China in 2007. Currently he works as a professor for the School of Information Engineering in Zhengzhou University and have published several papers concerning digital image processing. His research interest primarily includes digital video processing, machine vision, embedded system and image coding.