Research Article

Visual Sensor Based Image Segmentation by Fuzzy Classification and Subregion Merge

Huidong He,¹ Xiaoqian Mao,¹ Wei Li,² Linwei Niu,³ and Genshe Chen⁴

¹School of Electrical and Information Engineering, Tianjin University, Tianjin 300072, China
²Department of Computer & Electrical Engineering and Computer Science, California State University, Bakersfield, CA 93311, USA
³Department of Math and Computer Science, West Virginia State University, Institute, WV 25112, USA
⁴Intelligent Fusion Technology, Inc., Germantown, MD 20876, USA

Correspondence should be addressed to Wei Li; wli@csub.edu

Received 16 June 2017; Revised 4 August 2017; Accepted 9 August 2017; Published 25 September 2017

1. Introduction

Image segmentation is a fundamental issue in image processing and computer vision as it is a preliminary process in many applications that need to segment an image into meaningful regions. Applications of segmenting gray-level images are very limited so studies on color image segmentation become interesting because color images provide much more information than gray-level ones. A number of image segmentation techniques have been proposed in past decades [1–3]. A physical model in [4] is proposed for color image understanding. A widely used technique for color image segmentation is based on the idea of region growing. A color segmentation algorithm combining region growing and region merging processes is discussed in [5]. The fuzzy growing and expanding approach presented in [6] uses histogram tables for fine segmentation. A method for segmentation of color texture region using the defined class maps is presented in [7]. An approach to contrast analysis using fuzzy region growing is documented in [8]. Color image segmentation using seeded region growing is presented in [9–11]. Most pixel clustering algorithms work in the RGB color space in which the pixels are represented by red, green, and blue components. Clustering methods [12, 13], for example, c-means fuzzy classification [14], are often considered as an unsupervised classification of pixels as a prior knowledge of the image. Both supervised and unsupervised segmentation of textured color images are presented in [15, 16]. The level setting approaches to process a color image for segmentation are discussed in [17, 18]. All of the proposed techniques above provide a crisp segmentation of images, where each pixel is classified into a unique subset. This classification may not reflect an understanding of images by humans very well, as the work [1] stated “the image segmentation problem is basically one of psychophysical perception, and therefore not susceptible to a purely analytical solution.” Probably, one of
the difficulties in color image segmentation is how to define color conception, since an understanding of colors in an image varies by different observers.

Considering uncertainty and ambiguity in many real applications, this paper presents an approach to color image segmentation based on a fuzzy color extractor (FCE). Its preliminary version appears in [19]. The FCE is directly extended from the fuzzy gray-level extractor, which was applied to recognize landmarks on roads for autonomous vehicle navigation in [20–22]; that is, the fuzzy rules for extracting a road from a gray-level image are as follows:

If the gray level of a pixel is close to the gray-level of the road,

Then one can make it black, or

Else one can make it to white.

If these are modified to the rules for extracting an object from a color image, then

if the color components of a pixel closely matches the color components of an object,

then one can extract this pixel as a Matched pixel of the object, or

else one can extract this pixel as an Unmatched pixel.

The FCE viewed as a mapping operator of measuring colors is able to extract colors that are similar to a color pattern (seed). If a pixel is extracted it is defined as a Matched pixel; otherwise it is defined as an Unmatched pixel. In doing this, we develop procedures IniSeedList and UpdateSeedList for automatic selection of seeds. Our approach mainly consists of two algorithms: CreateSubImage and MergeSubImage. The CreateSubImage algorithm always extracts the seed as the first Matched pixel into a subimage and continually uses the FCE to check neighbors of each Matched pixel to grow the subimage. The subimage is created when neighbors' colors of all Matched pixels are “not close to” the seed's color. The CreateSubImage algorithm iteratively invokes UpdateSeedList to select new seeds to create next subimages until all the pixels in the image are processed. Different from all of the color segmentation techniques discussed above, the CreateSubImage algorithm most likely classifies an individual pixel into several subimages by fuzzy sets. As a result, two created subimages may share a common area, which can be considered as a color similarity metric for merging the subimages. The MergeSubImage algorithm first removes insignificant subimages from the set of subimages and merges significant subimages in order to extract meaningful segmented images. The insignificant subimages usually have a “too small” or a “too large” ratio of their area sizes over the source image. A “too small” insignificant subimage is mainly caused by a tiny fragment or a noise pixel in the source image, while a “too large” insignificant image is created by a “bad” seed. Then, the algorithm merges two subimages together according to their color similarity described by the size of their common pixels. We tested the proposed approach using exemplar color images from the database at UC Berkeley [23] in three different color spaces and discussed its effectiveness and robustness by comparing segmentation effect with two edge detection algorithms, namely, Canny edge detection and Log edge detection.

2. Three Color Spaces

In many color-based computer vision applications, it is very important to segment colors in an image, but it is very difficult due to the fact that colors are ambiguous in uncertain environments. In this study, we apply the FCE to segment colors in the image represented in the RGB, HSV, and YUV spaces.

2.1. RGB Color Space. RGB is the most traditional color space and is widely used in image segmentation because of its simplicity and fast processing [24, 25]. Colors consist of red (R), green (G), and blue (B) components, as shown in Figure 1(a).

2.2. HSV Color Space. HSV is another color space widely used for image processing, where a color is decomposed into hue (H), saturation (S), and value (V). The distribution of color in the HSV color space is shown in Figure 1(b).
color components in the RGB space can be transformed into the HSV space by
\[
\begin{align*}
\text{max} &= \max (R, G, B), \\
\text{min} &= \min (R, G, B), \\
V &= \max \left( \frac{(G - B)}{\text{max} - \text{min}} \times 60 \right) \\
R &= \max \left( \frac{(B - R)}{\text{max} - \text{min}} \times 60 \right) \\
G &= \max \left( \frac{(R - G)}{\text{max} - \text{min}} \times 60 \right) \\
H &= H + 360, \text{ if } H < 0, \\
H &= H \times 255 + 360.
\end{align*}
\]

Usually, the RGB values are between 0 and 255, so the HSV values are also between 0 and 255 through the transformation by (1).

2.3. YUV Color Space. YUV is the color space which is used in an analog color TV system, which translates color image components obtained in the RGB space into the luminance Y and the colors B - Y (U) and R - Y (V) by
\[
\begin{align*}
Y &= 0.299R + 0.587G + 0.114B, \\
U &= (-0.1687R - 0.3313G + 0.500B) + 128, \\
V &= (0.500R - 0.4187G - 0.0813B) + 128.
\end{align*}
\]

The YUV values are between 0 and 255 through the transformation by (2). The advantage of using the YUV color space for color segmentation is that the luminance Y is separated from the colors U and V.

3. Fuzzy Color Extractor

The fuzzy color extractor proposed here enables us to deal with such uncertainty. We will only discuss how to use the FCE-based algorithm defined in the RGB color space to segment color images, because the algorithm can directly process the color images in the HSV and YUV spaces, which are converted from the RGB space. Figure 1(a) shows the RGB space, where colors are represented by their red, green, and blue components in an orthogonal Cartesian space. The upper-left corner in the image is the origin. The color of each pixel \(p(n,m)\) denoted by \(p(n,m)_{RGB}\) is processed to separate its red, green, and blue components \((p(n,m)_R, \ p(n,m)_G, \ p(n,m)_B)\). The FCE extracts a cluster of colors based on a defined color pattern (seed) denoted by \(\text{seed}_{RGB}, \ \text{seed}_{RGB}\) and is either directly defined by its RGB components (seed_\(R\), seed_\(G\), seed_\(B\)) or determined by a pixel in the image. The color components’ differences between a pixel \(p(n,m)_{RGB}\) and seed_\(RGB\) are calculated as follows:
\[
\begin{align*}
\text{dif} (n,m)_R &= p(n,m)_R - \text{seed}_R, \\
\text{dif} (n,m)_G &= p(n,m)_G - \text{seed}_G, \\
\text{dif} (n,m)_B &= p(n,m)_B - \text{seed}_B,
\end{align*}
\]

\(0 \leq m < M, \ 0 \leq n < N.\)

\(M\) and \(N\) represent the size of an array which holds an image. The corresponding fuzzy rules can be obtained as follows:

- If \(\text{dif}(n,m)_R\) and \(\text{dif}(n,m)_G\) and \(\text{dif}(n,m)_B\) are Zero, Then \(p(n,m)\) is Matched.
- If \(\text{dif}(n,m)_R\) or \(\text{dif}(n,m)_G\) or \(\text{dif}(n,m)_B\) is Negative or Positive, Then \(p(n,m)\) is Unmatched.

Both the rules indicate that \(p(n,m)\) belongs to the object to be extracted, if the Euclidean distances between \(p(n,m)_{RGB}\) and \(\text{seed}_{RGB}\) along the three axes in RGB coordinate system are small enough; otherwise, \(p(n,m)\) does not belong to the object.

Figure 2(a) shows the membership functions \((\mu_N(x), \ \mu_Z(x), \ \mu_P(x))\) for the input fuzzy variables (Negative, Zero, Positive) defined by
\[
\begin{align*}
\mu_N (x) &= \begin{cases} 
1 & -255 \leq x < -\alpha_2 \\
\frac{x + \alpha_1}{\alpha_1 - \alpha_2} & -\alpha_2 \leq x < -\alpha_1 \\
0 & -\alpha_1 \leq x \leq 255,
\end{cases} \\
\mu_Z (x) &= \begin{cases} 
0 & -255 \leq x < -\alpha_2 \\
\frac{x + \alpha_2}{\alpha_2 - \alpha_1} & -\alpha_2 \leq x < -\alpha_1 \\
1 & -\alpha_1 \leq x \leq 255,
\end{cases} \\
\mu_P (x) &= \begin{cases} 
0 & -255 \leq x < -\alpha_1 \\
\frac{x + \alpha_2}{\alpha_2 - \alpha_1} & -\alpha_1 \leq x < 255, \\
1 & 255 \leq x \leq 255,
\end{cases}
\end{align*}
\]

and Figure 2(b) shows the membership functions \((\mu_M(x), \ \mu_U(x))\) for the output fuzzy variables (Matched, Unmatched) defined by
\[
\begin{align*}
\mu_M (x) &= \begin{cases} 
\frac{\rho_M - x}{\rho_M} & 0 \leq x < \rho_M \\
0 & \rho_M \leq x \leq 255,
\end{cases} \\
\mu_U (x) &= \begin{cases} 
0 & 0 \leq x < \rho_U \\
\frac{x - \rho_U}{255 - \rho_U} & \rho_U \leq x \leq 255,
\end{cases}
\end{align*}
\]

This document is protected by copyright. All rights reserved.

Journal of Electrical and Computer Engineering
where $\rho_M + \rho_U = 255$. Based on $\text{dif}(n,m)_R$, $\text{dif}(n,m)_G$, and $\text{dif}(n,m)_B$, the fuzzy rules produce the weight $\omega_m$ for Matched and the weight $\omega_u$ for Unmatched by

$$\omega_m = \min \{\mu_Z(R), \mu_Z(G), \mu_Z(B)\},$$

$$\omega_u = \max \{\mu_N(R), \mu_N(G), \mu_N(B), \mu_P(R), \mu_P(G)\}. \tag{6}$$

Figure 2(b) shows the produced areas in the output domain, while $\omega_m$ and $\omega_u$ cut $\mu_M(x)$ and $\mu_U(x)$. A crisp output value, $\Delta \rho_F$, is calculated by the centroid defuzzification method:

$$\Delta \rho_F = \frac{\int \mu_{\text{out}}(x) \, x \, dx}{\int \mu_{\text{out}}(x) \, dx}, \tag{7}$$

where $\mu_{\text{out}}(x)$ represents the envelope function of the areas cut by $\omega_m$ and $\omega_u$ in fuzzy output domain. If $\Delta \rho_F < \sigma$, $p(n,m)$ is extracted, $p(n,m)$ is not extracted, where $\sigma$ is a threshold. The FCE can be understood as a mapping operator between Euclidean distances $\{\text{dif}(n,m)_R, \text{dif}(n,m)_G, \text{dif}(n,m)_B\}$ in the RGB space and a difference $\Delta \rho_F$ in the intensity space under a fuzzy metric.

The example in Figure 3 shows how to create some subimages with interesting colors from the source image $I_{\text{FLOWER}}$ by the FCE. First, we define three seeds: seed$_{\text{RED}} = (255, 0, 0)$, seed$_{\text{GREEN}} = (0, 255, 0)$, and seed$_{\text{BLUE}} = (0, 0, 255)$ to create three subimages $\{I_{\text{RED}}, I_{\text{GREEN}}, I_{\text{BLUE}}\}$ that hold the words “red,” “green,” and “blue” from the source image $I_{\text{FLOWER}}$, as shown in Figure 3(b). Also, we select a seed $p(0,0)_{\text{RGB}} = (6, 6, 8)$ at the origin to remove the background from the source image $I_{\text{FLOWER}}$ as shown in Figure 3(c). Figure 3(d) shows $I_{\text{PETAL}} = I_{\text{FLOWER}} - (I_{\text{RED}} \cup I_{\text{GREEN}} \cup I_{\text{BLUE}} \cup I_{\text{BACKGROUND}})$ that holds the color components of the flower petal.
Table 1: Definitions of the eight subspaces shown in Figure 1(a).

<table>
<thead>
<tr>
<th>Vertex $i$</th>
<th>Subspace</th>
<th>Red-axis</th>
<th>Green-axis</th>
<th>Blue-axis</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_0^{RGB}$ (0, 0, 0)</td>
<td>$v_{black}$</td>
<td>[0, 127]</td>
<td>[0, 127]</td>
<td>[0, 127]</td>
</tr>
<tr>
<td>$v_0^{RGB}$ (0, 0, 255)</td>
<td>$v_{blue}$</td>
<td>[0, 127]</td>
<td>[0, 127]</td>
<td>[128, 255]</td>
</tr>
<tr>
<td>$v_0^{RGB}$ (0, 255, 0)</td>
<td>$v_{green}$</td>
<td>[0, 127]</td>
<td>[128, 255]</td>
<td>[0, 127]</td>
</tr>
<tr>
<td>$v_0^{RGB}$ (0, 255, 255)</td>
<td>$v_{cyan}$</td>
<td>[0, 127]</td>
<td>[128, 255]</td>
<td>[128, 255]</td>
</tr>
<tr>
<td>$v_0^{RGB}$ (255, 0, 0)</td>
<td>$v_{red}$</td>
<td>[128, 255]</td>
<td>[0, 127]</td>
<td>[0, 127]</td>
</tr>
<tr>
<td>$v_0^{RGB}$ (255, 0, 255)</td>
<td>$v_{pink}$</td>
<td>[128, 255]</td>
<td>[0, 127]</td>
<td>[128, 255]</td>
</tr>
<tr>
<td>$v_0^{RGB}$ (255, 255, 0)</td>
<td>$v_{yellow}$</td>
<td>[128, 255]</td>
<td>[128, 255]</td>
<td>[0, 127]</td>
</tr>
<tr>
<td>$v_0^{RGB}$ (255, 255, 255)</td>
<td>$v_{white}$</td>
<td>[128, 255]</td>
<td>[128, 255]</td>
<td>[128, 255]</td>
</tr>
</tbody>
</table>

Pseudocode 1

```
IniSeedList (I_{IMG})
for i ← 1 to 8 do
    pixel_num_i ← 0;
for n ← 1 to N do
    for m ← 1 to M do
        Calculate $d(n, m)$ in (8);
        $d_i(n, m) ← \min_k [d_k(n, m)];$
        pixel_num_i ← pixel_num_i + 1;
        Find seed candidate in each subspace;
        // with the shortest distance to its vertex $i$.
        Push seed candidate into SeedList;
        Sort seed candidates;
        // according to pixel number classified in the subspaces.
return SeedList;
```

4. Creating Subimages Using FCE

This section discusses the CreateSubImage algorithm for creating a set of subimages from a color image using the FCE in the RGB space. A critical step in this algorithm is to select seeds for the FCE. For this purpose, we evenly divide the RGB space into the eight subspaces $\{v_{black}, v_{red}, v_{green}, v_{blue}, v_{yellow}, v_{pink}, v_{cyan}, v_{white}\}$, which are defined by the well-defined colors: black, red, green, blue, yellow, pink, cyan, white, at their corresponding vertices of the RGB space, as shown in Figure 1(a). Table 1 lists their definitions; for example, $v_{black}$ is defined by a cubic $[0, 127] \times [0, 127] \times [0, 127]$ on the red-axis, green-axis, and blue-axis.

We calculate the distances of each pixel's RGB components to the eight well-defined colors by

$$d_i(n, m) = \sum [p(n, m)_{RGB} - v_i^{RGB}]^2 \quad i = 1, 2, \ldots, 8, \quad (8)$$

where $n$ and $m$ are the index of a pixel in the image, $p(n, m)_{RGB}$ is the color components of the pixel located at $(n, m)$, $v_i^{RGB}$ is the color components of vertex $i$ in the RGB space, and $d_i(n, m)$ is the distance between the pixel's color and vertex $i$.

We developed two procedures, IniSeedList and UpdateSeedList, to initialize a seed list and to update it. For an input image, IniSeedList classifies each pixel into one of the eight subspaces according to its minimum distance to vertex $i$ that is a well-defined color in the RGB space. In each subspace, IniSeedList selects such a pixel as a seed candidate, seed$_{cand}^{(i)}$, that has the shortest distance to its vertex $i$. Then, IniSeedList inserts seed$_{cand}^{(i)}$ into the seed list in order of the number of pixels classified in the subspaces and returns it. UpdateSeedList checks if seed$_{cand}^{(i)}$ is popped out or is extracted by the FCE. If so, UpdateSeedList updates seed$_{cand}^{(i)}$ in its corresponding subspace. The pseudocode of the IniSeedList procedures is shown in Pseudocode 1 and the pseudocode of the Update-SeedList is shown in Pseudocode 2.

Table 2 shows an example of a seed list generated by IniSeedList from the color image, $I_{BIRD}$. Columns 1 and 2 in Table 2 indicate the defined subspaces and the number of the classified pixels in the subspaces, column 3 shows the order of seed$_{cand}^{(i)}$ in the list, and columns 4 and 5 list the RGB color components and locations of seed$_{cand}^{(i)}$ in $I_{IMG}$, respectively. The top seed in Table 2 is seed$_{cand}^{(1)} = (240, 251, 243)$ with location $(269, 271)$ from the subspace $v_{white}$ within 100,369 pixels. After the algorithm pops out this seed for creating $I_M^{(1)}$, UpdateSeedList produces a new seed candidate in the subspace $v_{white}$ from $I_U^{(1)}$ and inserts it on the top of the list. The new seed$_{cand}^{(1)} = (240, 251, 247)$ is located at $(283, 267)$ in $I_U^{(1)}$ and is also from the subspace $v_{white}$ with 105,045 pixels. Please note that the
### Table 2: The seed list generated from the color image, $I_{BIRD}$.

<table>
<thead>
<tr>
<th>Subspace</th>
<th>Number of pixels</th>
<th>$seed_cand^{(0)}$ order</th>
<th>$seed_cand^{(0)}$ colors</th>
<th>$seed_cand^{(0)}$ locations $(n,m)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_white</td>
<td>100369</td>
<td>1</td>
<td>(244, 251, 243)</td>
<td>(269, 271)</td>
</tr>
<tr>
<td>v_black</td>
<td>19393</td>
<td>2</td>
<td>(0, 6, 0)</td>
<td>(193, 233)</td>
</tr>
<tr>
<td>v_cyan</td>
<td>70</td>
<td>3</td>
<td>(151, 100, 57)</td>
<td>(216, 190)</td>
</tr>
<tr>
<td>v_red</td>
<td>26</td>
<td>4</td>
<td>(206, 166, 107)</td>
<td>(225, 204)</td>
</tr>
<tr>
<td>v_blue</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>v_yellow</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>v_green</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>v_pink</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Pseudocode 2**

```latex
\textbf{UpdateSeedList} (\text{img}, \text{SeedList})
\text{subspace}\_\text{index}(i) \leftarrow 0;
\text{for } i \leftarrow 1 \text{ to } 8 \text{ do}
\quad \text{if } (\text{seed}\_\text{cand}^{(0)} \text{eimg})
\quad \quad \text{subspace}\_\text{index}(i) \leftarrow 1;
\quad \text{else}
\quad \quad \text{Remove seed}\_\text{cand}^{(0)} \text{ from SeedList;}
\quad \text{for } i \leftarrow 1 \text{ to } 8 \text{ do}
\quad \quad \text{if subspace}\_\text{index}(i) = 0
\quad \quad \quad \text{Find seed}\_\text{cand}^{(0)} \text{ in subspace\_}i;
\quad \quad \quad \quad \text{// with the shortest distance to its vertex } i.
\quad \quad \quad \text{Push seed}\_\text{cand}^{(0)} \text{ into SeedList;}
\quad \quad \text{Sort seed}\_\text{cand}^{(0)};
\quad \text{// according to pixel number classified in the subspaces.}
\text{return SeedList;}
```

input image to InisSeedList is $I_{\text{img}} = I_{\text{BIRD}} - I_{\text{CENTER}}$, instead of $I_{\text{BIRD}}$ itself. $I_{\text{CENTER}}$ is an image extracted by seed $\text{CENTER} = (127, 127, 127)$ that is the center of the RGB space. It is difficult to determine in which subspace the center color is classified, so any seed close to the center may not be able to differentiate colors very well, as shown in Figure 4(a). In order to guarantee to get a “good” seed, our approach does not select any seed falling in $I_{\text{CENTER}}$. It is also possible that there is no pixel in a subspace; that is, the seed candidates are less than eight. For example, the seed candidates for $I_{\text{BIRD}}$ are not available in the subspaces $v_{\text{blue}}, v_{\text{yellow}}, v_{\text{green}}, \text{and } v_{\text{pink}}$.

The pseudocode of the CreateSubImage procedures is listed as shown in Pseudocode 3.

CreateSubImage creates a set of subimages from a color source image. CreateSubImage invokes mainly three procedures: InisSeedList, UpdateSeedList, and FuzExtractor. The pseudocode of the FuzExtractor is shown in Pseudocode 4.

CreateSubImage has two arguments: $I_{\text{SRC}}$ and center, where $I_{\text{SRC}}$ is a source image and center is a Boolean variable. If center = true, $I_{\text{CENTER}}$ is created by seed $\text{CENTER}$, otherwise $I_{\text{CENTER}}$ is assigned by the empty image ø. FuzExtractor is mainly implemented based on (3) and extracts a subimage $I_{\text{TEMP}}$. The FuzExtractor procedure starts with seed $\text{RGB}$ to generate $I_{\text{MATCH}}$ with region growing. Because seed $\text{RGB}$ belongs to its subimage $I_{\text{MATCH}}$. FuzExtractor pushes its four neighbor pixels into a queue and checks the first-in element, pixel $\text{FIRST}$, in the queue to see if it belongs to the subimage $I_{\text{MATCH}}$. If pixel $\text{FIRST}$ belongs to $I_{\text{MATCH}}$, its new four neighbors are pushed into the queue. The FuzExtractor procedure repeats this process until the queue is empty, which indicates that there are no more pixels in the source image $I_{\text{SRC}}$ matching the seed $\text{RGB}$ color. After FuzExtractor returns $I_{\text{MATCH}}$, CreateSubImage assigns $I_{\text{MATCH}}$ to $I_{\text{TEMP}}$ as described.

CreateSubImage uses two conditions to check if $I_{\text{TEMP}}$ is significant or not. The first condition is to check $I_{\text{TEMP}}/I_{\text{SRC}}$ that it is a ratio of $I_{\text{TEMP}}$ area over $I_{\text{SRC}}$ one. If this ratio is too small, CreateSubImage ignores $I_{\text{TEMP}}$ as $I_{\text{TEMP}}$ holds too few pixels extracted from the source image $I_{\text{SRC}}$. $I_{\text{TEMP}}$ is most likely affected by noise and is not significant for any object in the source image. CreateSubImage also uses the condition, $(I_{\text{TEMP}}/I_{\text{SRC}}) \& \text{var}(I_{\text{TEMP}})$, to check $I_{\text{TEMP}}$, where var($I_{\text{TEMP}}$) is a variance defined by

$$\text{var}(I_{\text{TEMP}}) = E \left( \left( I_{\text{TEMP}} - \mu \right)^2 \right),$$

where \(\mu = E \left( \left( I_{\text{TEMP}} \right)^2 \right)\). A big $I_{\text{TEMP}}/I_{\text{SRC}}$ indicates a large area size of $I_{\text{TEMP}}$ and a big var($I_{\text{TEMP}}$) implies that $I_{\text{TEMP}}$ contains the color components
CreateSubImage \( (I_{\text{SRC}}, \text{center}) \)
\[
\begin{align*}
i &\leftarrow 0; I_M^{(i+1)} \leftarrow \emptyset; I_U^{(i)} \leftarrow I_{\text{SRC}}; \\
\text{If (center = true)} &\quad I_{\text{CENTER}} \leftarrow \text{FuzExtractor} (I_{\text{SRC}}, \text{seed}_{\text{CENTER}}); \\
\text{else} &\quad I_{\text{CENTER}} \leftarrow \emptyset; \\
I_{\text{IMG}} &\leftarrow I_U^{(i)} - I_{\text{CENTER}}; \\
\text{SeedList} &\leftarrow \text{IniSeedList} (I_{\text{IMG}}); \\
\text{while} (\text{SeedList} \neq \emptyset) \text{ do} \\
\quad \text{seed}_{\text{RGB}} &\leftarrow \text{The first element in SeedList}; \\
\quad I_{\text{TEMP}} &\leftarrow \text{FuzExtractor} (I_{\text{SRC}}, \text{seed}_{\text{RGB}}); \\
\quad \text{if} &\quad I_{\text{TEMP}}/I_{\text{SRC}} > \epsilon \\
\quad &\quad \text{if} (I_{\text{TEMP}}/I_{\text{SRC}} < \gamma_1 \& \text{ var} (I_{\text{TEMP}}) < \gamma_2) \quad I_M^{(i+1)} \leftarrow I_{\text{TEMP}}; \\
\quad &\quad I_U^{(i+1)} \leftarrow I_{\text{SRC}} - (I_M^{(i+1)} \cup I_M^{(i)}) \cdots I_M^{(1)}); \\
\quad &\quad I_{\text{IMG}} \leftarrow I_U^{(i+1)} - I_{\text{CENTER}}; \\
\quad \text{SeedList} &\leftarrow \text{IniSeedList} (I_{\text{IMG}}); \\
\quad i &\leftarrow i + 1; \\
\text{end} \\
\text{return} &\quad \{I_M^{(i+1)}\};
\end{align*}
\]

Pseudocode 3

with a wide range distribution, as shown in Figure 4(b). Such a subimage is also considered as insignificant. If \( I_{\text{TEMP}} \) is significant, CreateSubImage assigns \( I_{\text{TEMP}} \) to a Matched subimage \( I_M^{(i+1)} \) and creates an Unmatched \( I_U^{(i+1)} \), where \( I_M^{(i+1)} \) holds the color components matching the color of seed_{RGB} and \( I_U^{(i+1)} \) holds unextracted pixels in the source image. CreateSubImage iteratively processes \( I_U^{(i)} \) until \( I_U^{(i)} = \emptyset \); that is, each pixel in the image is processed.

There may exist tiny holes in the subimage \( I_M^{(i)} \). Because these tiny fragments may be insignificant for object pattern
FuzExtractor ($I_{SRC}$, seed$_{RGB}$)

Move seed$_{RGB}$ into $I_{MATCH}$;
Push four neighbor pixels of seed$_{RGB}$ into queue;
while (queue ≠ ø) do
  pixel$_{FIRST}$ ← the top element in queue;
  Calculate $\Delta \rho_F$ using (3);
  if ($\Delta \rho_F < \sigma$) then
    Move pixel$_{FIRST}$ into $I_{MATCH}$;
    Push unduplicated neighbors of pixel$_{FIRST}$ into queue;
  end
return $I_{MATCH}$;

Pseudocode 4

MergeSubImage ($\{I_M^{(i)}\}$)

$K ← I$;
for $i ← 1$ to $I$ do
  for $l ← i - 1$ to $I$ do
    if ($I_M^{(i)} \cap I_M^{(l)} > \tau$; $I_M^{(i)} ← I_M^{(i)} \cup I_M^{(l)}$;
      Remove $I_M^{(l)}$ from the list;
      $K ← K - 1$;
    break;
for $k ← 1$ to $K$ do
  $I_{OBJ}^{(k)} ← I_M^{(k)}$;
  $I_{SEG} ← I_{OBJ}^{(k)} \cup I_{OBJ}^{(k-1)} \cdots I_{OBJ}^{(1)}$;
return $I_{SEG}$;

Pseudocode 5

5. Merging Subimages

The proposed algorithm merges significant subimages into a set of meaningful segmented objects by the procedure MergeSubImage. The pseudocode of the MergeSubImage procedure is shown in Pseudocode 5.

The inputs to the procedure MergeSubImage are the subimages $\{I_M^{(i)}\}$, and the output from MergeSubImage is the segmented image $I_{SEG}$. Figures 5(a)–5(b) show a color image, $I_{BIRD}$, and the segmented image in the RGB space, $I_{SEG}$, which is a union of 6 objects, $I_{OBJ}^{(i)}$, $i = 1, \ldots, 6$, shown in Figures 5(f)–5(k). Figure 5(e) shows the manually made benchmark, $I_{BECHMARK}$.

MergeSubImage merges the subimages according to the overlap sizes of their color components to construct some objects in the source image. In the process of the MergeSubImage, the FCE usually classifies an individual pixel into multiple subimages. It causes these subimages to share a common color area in the RGB space, and even probably a
subimage may fully cover another subimage. The common area of two subimages implies two features: spatial proximity and color similarity, because the two subimages connect together and have a partition with the same color if they share the common area. Therefore, our algorithm uses this unique characteristic to merge subimages. MergeSubImage uses the union \((I_M^{(i)} \cap I_M^{(l)})/I_M^{(i)})\) to check the common area size of two subimages \(I_M^{(i)}\) and \(I_M^{(l)}\) in the RGB space. A bigger value of \((I_M^{(i)} \cap I_M^{(l)})/I_M^{(i)})\) means that \(I_M^{(i)}\) and \(I_M^{(l)}\) share a larger common area; that is, their color components are more similar. A data structure for maintaining the subimages \(\{I_M^{(i)}\}\) is a list that sorts \(\{I_M^{(i)}\}\) in a sending order of their sizes. Technical implementation of merging \(\{I_M^{(i)}\}\) is shown in Figure 6. The merging process starts with the last element \(I_M^{(i)}\) with \(i = I\), located at the list tail, which is the subimage with the smallest area size, and checks \(I_M^{(i)}\) with each subimage \(\{I_M^{(l)}\}, l = i - 1, \ldots, 1\) via \((I_M^{(i)} \cap I_M^{(l)})/I_M^{(i)})\). If \((I_M^{(i)} \cap I_M^{(l)})/I_M^{(i)}) > \tau\), the algorithm merges \(I_M^{(i)}\) and \(I_M^{(l)}\) together, removes \(I_M^{(i)}\) from the list, and replaces \(I_M^{(i)}\) with \(I_M^{(i)} \cup I_M^{(l)}\), where \(I_M^{(l)}\) size is bigger than \(I_M^{(i)}\) one. After processing all the subimages in the list, the list holds the partitioned objects \(\{I_{OBJ}^{(k)}\}\) from the input source image \(I_{SRC}\). MergeSubImage returns the partitioned objects \(\{I_{OBJ}^{(k)}\}\) as well as its union, \(I_{OBJ}^{(k)} \cup I_{OBJ}^{(k-1)} \cdots I_{OBJ}^{(1)}\), as the segmented image \(I_{SEG}\).

Table 4 lists the extracted subimages from the source image \(I_{BIRD}\) in Figure 5(a). MergeSubImage merges these
Table 4: $I_{\text{OBJ}}^{(i)}$ merged by subimages created from $I_{\text{BIRD}}$.

<table>
<thead>
<tr>
<th>Segmented objects $I_{\text{OBJ}}^{(i)}$</th>
<th>Sub-images</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{\text{OBJ}}^{(1)}$</td>
<td>$I_M^{(1)}$</td>
</tr>
<tr>
<td>$I_{\text{OBJ}}^{(2)}$</td>
<td>$I_M^{(2)}, I_M^{(9)}, I_M^{(12)}$</td>
</tr>
<tr>
<td>$I_{\text{OBJ}}^{(3)}$</td>
<td>$I_M^{(12)}$</td>
</tr>
<tr>
<td>$I_{\text{OBJ}}^{(4)}$</td>
<td>$I_M^{(22)}$</td>
</tr>
<tr>
<td>$I_{\text{OBJ}}^{(5)}$</td>
<td>$I_M^{(22)}, I_M^{(22)}, I_M^{(22)}, I_M^{(22)}, I_M^{(22)}, I_M^{(22)}, I_M^{(22)}$</td>
</tr>
<tr>
<td>$I_{\text{OBJ}}^{(6)}$</td>
<td>$I_M^{(22)}, I_M^{(22)}, I_M^{(22)}, I_M^{(22)}, I_M^{(22)}, I_M^{(22)}, I_M^{(22)}$</td>
</tr>
</tbody>
</table>

20 subimages into 6 partitioned objects $\{I_{\text{OBJ}}^{(i)}\}$ in Figures 5(f)–5(k). Table 4 also shows information on which and how many subimages are merged int0 which object. For example, the partitioned object $I_{\text{OBJ}}^{(3)}$ shown in Figure 5(f) comes from a single subimage $I_M^{(11)}$, while the partitioned object $I_{\text{OBJ}}^{(5)}$ in Figure 5(j) is produced by merging 11 subimages $I_M^{(9)}, I_M^{(22)}, I_M^{(23)}, I_M^{(27)}, I_M^{(29)}, I_M^{(32)}, I_M^{(34)}, I_M^{(36)}, I_M^{(38)}, I_M^{(41)}, I_M^{(43)}$. Figure 5(b) shows the segmented image $I_{\text{SEG}}$ by a union operation of $I_{\text{OBJ}}^{(6)} \cup I_{\text{OBJ}}^{(5)} \ldots I_{\text{OBJ}}^{(1)}$.

6. Discussions and Conclusions

In this section, we present our selection of the membership functions’ parameters of the FCE defined in (4)–(7) and settings of the CreateSubImage and MergeSubImage procedures used for our tests. We empirically selected the parameters of the membership functions as $\alpha_1 = 60, \alpha_2 = 200, \rho_M = 20, \rho_M + \rho_U = 255$, and $\sigma = 90$. A smaller $\alpha_1$ may more precisely segment color components in an input image but produces more extracted subimages. A slight change of $\alpha_2$ does not affect the segmentation result, so it is not sensitive. $\rho_M$ should be at least less than 127.5. The threshold, $\sigma$, is important for classifying pixels into a Matched subimage $I_M^{(i)}$ or an Unmatched subimage $I_U^{(i)}$. We select the parameters in CreateSubImage and MergeSubImage as $\varepsilon = 0.01, \gamma_1 = 0.25, \gamma_2 = 500$, and $\tau = 0.6$. The function of $\varepsilon$ is to remove insignificant subimages with a “too small” area that may be caused by noise. Both parameters $\gamma_1$ and $\gamma_2$ are defined to remove another type of insignificant subimages, for example, $I_{\text{TEMP}}$ shown in Figure 4(b). This subimage is not extracted very well due to a “bad seed,” seed$_{RGB} = (64,28,28)$. Such a subimage is featured by its large area size and a large variance of its color components. The large variance indicates that the subimage holds a wide range of color distribution in the RGB space, so this subimage should be removed. $\tau$ is the threshold to check the overlap area over the subimage with a smaller area. In this paper, $\tau = 0.6$ herein means that the overlap area is 60% of the subimage with the smaller area. The bigger $\tau$ is, the more similar the color components of the two subimages are. Changing $\tau$ affects the number of partitioned objects generated by MergeSubImage. As an extreme example, $\tau = 1.0$ means that only if the larger subimage fully overlaps the smaller subimage, the two subimages will be merged together.

In our implementation, the algorithm invokes the CreateSubImage procedure twice. For the first invocation, an input image, $I_{\text{IMG}}$, to IniSeedList and UpdateSeedList, is $I_{\text{SRC}} - I_{\text{CENTER}}$, where $I_{\text{CENTER}}$ is extracted by seed$_{CENTER} = (127,127,127)$. Because the first call is to process a source image $I_{\text{SRC}}$, the algorithm does not select a seed close to the RGB space center. As discussed in Section 4, the seed $I_{\text{CENTER}}$ cannot differentiate color components which are far from the well-defined eight colors. For example, the extracted image $I_{\text{CENTER}}$ shown in Figure 4(a) holds many color components of six corn cobs. Because the algorithm does not select any seed from $I_{\text{CENTER}}$, it is very possible that some of the pixels in the source image remain unclassified. In
order to process the unclassified pixels, we create an image $I_{\text{UNCLASSIFIED}} = I_{\text{SRC}} - \cup I_{\text{OBJ}}^{(i)}$, as shown in Figure 4(c). The algorithm calls the CreateSubImage a second time to process $I_{\text{UNCLASSIFIED}}$, but the input image to IniSeedList and UpdateSeedList is $I_{\text{UNCLASSIFIED}}$ itself. The variance of $I_{\text{UNCLASSIFIED}}$ is smaller than that of its original source image $I_{\text{SRC}}$, as most pixels with a large range of color distribution are classified after first calling CreateSubImage. For this reason, we halve the parameter $\alpha_1 = 30$ of the membership functions to process $I_{\text{UNCLASSIFIED}}$.

The proposed approach segments the color image $I_{\text{CORN}}$ in Figure 7(a) into 12 objects. Figure 7(b) shows the segmented image. Figures 7(f)–7(q) display the first 12 partitioned objects. $I_{\text{OBJ}}^{(3)}$ and $I_{\text{OBJ}}^{(6)}$ in Figures 7(h) and 7(k) are extracted from $I_{\text{UNCLASSIFIED}}$ in Figure 4(c).

In order to test the robustness of the proposed algorithm, we applied the algorithm with the same settings defined in the first paragraph of this section to process the five color images $\{I_{\text{BIRD}}, I_{\text{CORN}}, I_{\text{LAKE}}, I_{\text{MAN}}, I_{\text{BABOON}}\}$, in the RGB color space as shown in Figures 5(a), 7(a), 8(a), 9(a), and 10(a), respectively. Their corresponding segmented images are displayed in Figures 5(b), 7(b), 8(b), 9(b), and 10(b), respectively. The first four color images are from the database at UC Berkeley [23]. These selected color images represent different features and diverse scenes that might be taken under different light conditions. The source image $I_{\text{BIRD}}$ in Figure 5(a) holds a bird on a tree trunk under the sky. The source image $I_{\text{CORN}}$ in Figure 7(a) holds six corns with many small multicolor kernels and its segmented image holding the six colorful corns. The source image $I_{\text{LAKE}}$ in Figure 8(a) shows a lake and a mountain under the sky at evening. Figures 9(a) and 10(a) show the color images $I_{\text{MAN}}$ and $I_{\text{BABOON}}$. Table 5 lists the subimages, and the partitioned objects from the 5 color images. 85 subimages are extracted from $I_{\text{BABOON}}$, but the partitioned objects are only 6.

There is a lack of a strong mathematical model for precisely expressing the human’s understanding of colors due to psychophysical perception. For a given color, different observers may perceive its color definition differently. That is why the FCE-based approach classifying an individual pixel into several subimages is reasonable. The proposed approach to color image segmentation is easy to use, and it does not apply any procedure where one needs to preprocess source images. Removing subimages with “too small” areas is filtering short edges and noise in a color image. The approach automatically partitions a color image into an appropriate number of meaningful objects or regions. The manually made benchmarks are downloaded from the database at UC Berkeley [23], as shown in Figures 5(e), 7(e), 8(e), 9(e), 11(e), and 12(e), respectively. These benchmarks display desired segmentation results represented by the images’ boundaries.

Although the proposed approach does not reach the benchmarks’ results, it is able to partition the color images into meaningful objects, for example, the $I_{\text{CORN}}$ segmentation results as shown in Figure 7.

In this study, we also compared the performance of the FCE-based algorithm by segmenting the same test images $\{I_{\text{BIRD}}, I_{\text{CORN}}, I_{\text{LAKE}}, I_{\text{MAN}}, I_{\text{BABOON}}\}$ in the RGB, HSV, and YUV spaces, respectively. In this case, we only transformed the color image representation from the RGB space into the HSV and YUV spaces according to (1) and (2). The test images are selected randomly from the database at UC Berkeley [23], as shown in Figures 5(a), 7(a), 8(a), 9(a), 11(a), and 12(a). Their corresponding segmented images in the RGB, HSV, and YUV spaces are displayed in Figures 5(b)–5(d), 7(b)–7(d), 8(b)–8(d), 9(b)–9(d), 11(b)–11(d), and 12(b)–12(d). We find that the segmented images in the HSV space $I_{\text{SEG}}$ are the most different from $I_{\text{BENCHMARK}}$ by comparison. So the FCE-based algorithm defined in the HSV space is not adopted. In some cases, the segmentation results in the YUV space are superior to the RGB spaces, such as the “face” of $I_{\text{MAN}}$ in Figure 9 and the “boxes” in $I_{\text{VEGGI}}$ in Figure 12. But in most cases, the segmented images in the RGB space $I_{\text{SEG}}$ are the most similar to $I_{\text{BENCHMARK}}$. It is proven that the FCE-based algorithm defined in the RGB space is the most robust. The possible reason for the results is the transformation of the color image representation from the RGB space into the HSV and YUV spaces may cause color errors so that the segmentation is influenced.

In order to show the advantages of the new approach to color image segmentation, we applied Canny edge detection and Log edge detection algorithms to segment the test images $\{I_{\text{LAKE}}, I_{\text{MAN}}, I_{\text{HORSE}}, I_{\text{VEGGI}}\}$, respectively, and the segmented images were shown in Figures 8(f)–8(g), 9(f)–9(g), 11(f)–11(g), and 12(f)–12(g). Then we compared $I_{\text{CANNY}}$ and $I_{\text{LOG}}$ with the segmented images through the FCE-based approach, namely, $I_{\text{SEG}}$, $I_{\text{HSE}}$, and $I_{\text{HSE}}$, and meanwhile, $I_{\text{BENCHMARK}}$ can be used as a metric to evaluate the segmentations. As shown in Figures 8 and 9, the edges in $I_{\text{CANNY}}$ and $I_{\text{LOG}}$ are “too many” or “too few” compared with $I_{\text{BENCHMARK}}$, which proves that the segmentation effect with Canny edge detection and Log edge detection is inferior to the FCE-based approach. Although the segmentation effect with Log edge detection is superior to the FCE-based approach applied in the HSV and YUV spaces, it performs not well as the proposed approach applied in the RGB color space, as shown in Figures 11 and 12. Overall, compared with Canny edge detection and Log edge detection, the new approach based on FCE defined in the RGB color space presents the best performance in the color image segmentation.

The approach is effective in tracking or following a certain color because the FCE-based approach does not need a precise color pattern to extract a set of colors. We apply this algorithm to extract chemical plumes in images taken underwater at very poor illumination conditions [26]. In this paper, we start selecting seeds near the eight well-defined colors, that is, at the vertices of the RGB space, and move their selection from the vertices to the RGB center. Our further research will investigate more effective methods for selecting seeds because seed selection plays an important role in this algorithm.

### Table 5: Statistics of subimages and partitioned objects of five color images.

<table>
<thead>
<tr>
<th>ID</th>
<th>$I_{\text{BIRD}}$</th>
<th>$I_{\text{CORN}}$</th>
<th>$I_{\text{LAKE}}$</th>
<th>$I_{\text{MAN}}$</th>
<th>$I_{\text{BABOON}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{\text{M}}^{(i)}$</td>
<td>20</td>
<td>41</td>
<td>16</td>
<td>22</td>
<td>85</td>
</tr>
<tr>
<td>$I_{\text{OBJ}}^{(i)}$</td>
<td>6</td>
<td>13</td>
<td>6</td>
<td>54</td>
<td>6</td>
</tr>
</tbody>
</table>
Figure 7: Source image, $I_{\text{CORN}}$; segmented images in the RGB, HSV, and YUV space, $I_{\text{SEG}}$, $I'_{\text{SEG}}$, and $I''_{\text{SEG}}$; manually made benchmark, $I_{\text{BENCHMARK}}$; and partitioned objects in the RGB space, $I_{\text{OBJ}}$. 
Figure 8: Source image, $I_{LAKE}$; segmented images in the RGB, HSV, and YUV space, $I_{SEG}$, $I'_{SEG}$, and $I''_{SEG}$; manually made benchmark, $I_{BENCHMARK}$; segmented images with Canny edge detection and Log edge detection, $I_{Canny}$, $I_{Log}$.

Figure 9: Source image, $I_{MAN}$; segmented images in the RGB, HSV, and YUV space, $I_{SEG}$, $I'_{SEG}$, and $I''_{SEG}$; manually made benchmark, $I_{BENCHMARK}$; segmented images with Canny edge detection and Log edge detection, $I_{Canny}$, $I_{Log}$.

Figure 10: Source image $I_{BABOON}$ and its segmented image $I_{SEG}$. 
Figure 11: Source image, $I_{\text{HORSE}}$; segmented images in the RGB, HSV, and YUV space, $I_{\text{SEG}}$, $I'_{\text{SEG}}$, and $I''_{\text{SEG}}$; manually made benchmark, $I_{\text{BENCHMARK}}$; segmented images with Canny edge detection and Log edge detection, $I_{\text{Canny}}$, $I_{\text{Log}}$.

Figure 12: Source image, $I_{\text{VEGGI}}$; segmented images in the RGB, HSV, and YUV space $I_{\text{SEG}}$, $I'_{\text{SEG}}$, and $I''_{\text{SEG}}$; manually made benchmark, $I_{\text{BENCHMARK}}$; segmented images with Canny edge detection and Log edge detection, $I_{\text{Canny}}$, $I_{\text{Log}}$. 
role in achieving a good segmentation performance. Another issue we will address in our study is how to automatically determine the parameters of fuzzy sets based on an input color image.

Conflicts of Interest

The authors declare no conflicts of interest.

Authors’ Contributions

All the authors contributed equally to this work.

Acknowledgments

This research work was funded by the National Natural Science Foundation of China under Grant no. 61473207.

References


