Color Manipulation in Images by Using a Modified HSV Algorithm

Samuel Kosolapov
Department of Electronics
Braude Academic College of Engineering
Karmiel, Israel
e-mail: ksamuel@braude.ac.il

Abstract— **There are several situations in which the HSV (Hue, Saturation, Value) algorithm is preferred to execute color manipulations in a specific image. Unfortunately, whereas Value and Saturation can be calculated in any situation, Hue cannot be calculated when the specific pixel is gray. Even more, for dark regions, nearly gray regions, and for overbleached regions, the calculation of the Hue is non-reliable. Inherent to digital image noise makes the calculations of Hue in the above situations even more problematic. In an attempt to provide more control during color manipulations, an extended structure "sHSVF" was defined, in which F is a "Validity Flag". Additionally, the structure "sValidityParameters" was defined. Values of the fields in this structure make it possible to classify the specific pixel as "IS_VALID", "HAS_ZERO", "IS_TOO_GRAY", IS_OVERBLEACHED", "IS_TOO_DARK", "IS_TOO_LIGHT", and properly set the "Validity Flag" for each pixel of the image. This flag may be instrumental in color recognition and in reliably modifying the color of the pixels in accordance with the selected rules. By selecting values of the "sValidityParameters", the user of the algorithm can specify situations when the color of the specific pixel is set to the pre-defined value, marking problematic situations. Provided examples demonstrate that this approach can be used for reliable color recognition and advanced color manipulations for synthetic and real-life images.**.

Keywords-Image Processing; HSV; color manipulations; reliable color recognition.

I. INTRODUCTION

The standard inexpensive color digital camera on its output produces a sequence of bytes. In order to apply to this sequence basic imaging processing algorithms, this sequence is organized as a two-dimensional matrix of picture elements (pixels). Each pixel is a vector in the {R (red), G (green), B (blue)} space. In the inexpensive cameras values of color components are in the range {0..255}. Presentation of the color as an {R, G, B} vector is quite natural for a human observer, having three types of color receptors. However, for applications used in machine vision, this presentation is not always convenient. An alternative presentation uses {H (hue), S (saturation) V (value)} space, or {H, S, L (lightness)} space. Presentations of the pixels in the RGB, HSV, and HSL spaces are described in a number of classical image processing books [1]-[3]. Functions converting pixels in the RGB space to HSV and HSL spaces and back are well known. The value of H actually describes the color of the pixel and thus can be used to recognize the color of the pixel in a simple and convenient way for machine vision.

There is a number of alternative approaches – for example, a sophisticated approach based on a sequence of different image processing algorithms designed for the specific goal [4]. However, algorithms of that type are in most cases too heavy for real-life applications.

Unfortunately, plain and simple HSV and HSL algorithms have an inherent problem: Hue cannot be calculated if R, G, and B values are equal. When the presentation of pixel values in the range of byte {0..255] is used, classic presentation HSV and HSL became problematic. One solution is to use the zero value of S (saturation) as a marker, pointing out that the value of H cannot be calculated in that case. However, practically when an image has a noise (and images of digital cameras always have significant noise), the situation becomes even more problematic. It is clear that pixel {100,100,100} is gray, and in this situation, Hue cannot be calculated. But if a digital camera produces a noise of, say 5 units, then pixel {98, 101, 104} must be treated as problematic for the reliable Hue calculation. Despite the fact that the value of Hue can be calculated in that case, it is clear that using this value for color recognition may lead to unreliable results.

Detailed analysis of the properties of digital cameras reveals a number of additional problematic situations. For the synthetic image (an image created by software), pixel {100, 0, 0} is a legal description of a RED pixel. But for the digital camera having noise 5 units this zero value is electronically problematic. The same is valid for the value equal to 255 – this value, in most cases, means that the object is overbleached. Those and some other situations make the classic HSV/HSL approach at least problematic for real-life applications.

The upper image in Figure 1 demonstrates a real-life photo of the mandarins. Vertical and horizontal orange arrows point to the marker (cross) in the clearly overbleached region of this selected mandarin. The lower image presents the profile of the row of the marker in the upper image. The blue arrow point to the values that arrived at the maximal value of 255. Despite the human observer probably recognizing those pixels as "pixels of orange", machine vision must reject Hue calculations of those pixels as non-reliable. The green arrow is pointing to the regions in which the blue value is very low and even arrives at the minimal zero value. In the upper image, those pixels are green leaves in the shadow. In this situation, the color of those pixels cannot be reliably evaluated as by a human

 ISBN: 978-1-68558-048-3

observer, as by hue calculation by using a computer algorithm.

Earlier attempts to improve HSV/HSL algorithm were described in [5] and [6]. This article described a more elaborated approach in an attempt to provide a more reliable algorithm of color recognition and color manipulation based on a modification of the classic HSV algorithm.

To cope with the above problems, in an attempt to provide more control during color recognition and color manipulations, an extended structure “sHSVF” was defined, in which F is a "Validity Flag".

Additionally, the structure "sValidityParameters" was defined. Values of the fields in this structure make it possible to classify the specific pixel as "IS_VALID", “HAS_ZERO”, "IS_TOO_GRAY", IS_OVERBLEACHED", "IS_TOO_DARK", "IS_TOO_LIGHT", and properly set the "Validity Flag" for each pixel of the image. By selecting values of the “sValidityParameters”, the user of the modified HSV algorithm can specify situations when the color of the specific pixel is set to the pre-defined value, marking problematic situations.

Section II describes the definition of sHSVF structure (subsection ‘A’), sValidityParameter (subsection ‘B’), and flags that are used in specific situations (subsection ‘C’).

Section III presents changes in the classical HSV algorithm.

Section IV presents exemplary analyses and processing of synthetic and real-life images demonstrating the properties of a modified algorithm.

Section V shortly summarizes the results obtained.

II. STRUCTURES SHSVF, SVALIDITYPARAMETER AND FLAGS

To store {R, G, B} values of the pixel, standard sRGB structure was used without changes:

```
struct sRGB
{
        unsigned char r;
        unsigned char g;
        unsigned char b;
};
```

Standard sHSV structure was modified by using the “double” type and by adding the integer “ValifityFlag”.

A. *Structure sHSVF*

The resulting sHSV structure was defined as:

```
struct sHSVf
{
        double H;   // Hue
        double S;   // Saturation
        double V;   // Value
        int    validityFlag; // Validity flag
}; .
```

B. *Structure sValidityParameter*

Structure sValidityParameter was designed to set numerical values needed to mark problematic pixels. It was defined as:

```
struct sValidityParameter
{
        double rgbMeanVmin;
        double rgbMeanMax;

        double SaturationMin;
};
```

Usage of this structure will be described later.

C. *Definitions of FLAGS*

In order to properly mark problematic situations, the following FLAGS were defined:

```
#define IS_VALID (0)
```

This flag is set when the value of HUE can be calculated. In this situation value in the HUE map and in the Saturation map is gray in the range {0..255}.

```
#define HAS_ZERO (1)
```

This flag was set when at least one of the {R, G, B} values were zero. This situation is electronically problematic, hence those pixels on the HUE and Saturation maps were marked by a MAGENTA color.

```
#define IS_TOO_DARK    (2)
```

This flag was set if the mean value calculated as (R+G+B)/3 was lower than the value set in the rgbMeanMin parameter of the structure sValidityParameter. Those pixels in the HUE and Saturation maps were marked by a RED color.

```
#define IS_TOO_LIGHT   (3)
```

This flag was set if the mean value calculated as (R+G+B)/3 was higher than the value set in the rgbMeanMax parameter of the structure sValidityParameter. Those pixels on the HUE and Saturation maps were marked by a YELLOW color.

```
#define IS_OVERBLEACHED   (4)
```

This flag was set if at least one value in {R,G,B} is 255. Those pixels on the HUE and Saturation maps were marked by a GREEN color.

```
#define IS_TOO_GRAY    (5)
```

This flag was set if the calculated saturation value was lower than the value of SaturationMin in the structure sValidityParameterat. Those pixels on the HUE and Saturation maps were marked by a BLUE color.

Again, if none of the above flags was set, sValidityParameter is set to the IS_VALID value defined as zero. Then pixels in the Hue and Saturation maps are gray pixels, whereas the level of gray mapping Hue and Saturation values to the range of [0..255}. It must be noted that historically, in the Windows OS, values of Hue and Saturation were mapped in the {0..239} range. Some authors mapped values of Hue in the {0..360} range, however, this range cannot be presented in the standard displays designed for humans. Hence, the range [0..255} is better suited the goal of this research.

 ISBN: 978-1-68558-048-3

III. CHANGES IN THE CLASSICAL HSV ALGORITHM

Classical function RGBtoHSV which is described at [1]-[3], and C-code of which is available in the public domain was modified by adding flags defined before. A number of exemplary code fragments of the reworked function ConvertRGBtoHSVf are presented in Figure 2. This function is defined as:

```
void ConvertRGBtoHSVf(
        sRGB rgb,  sHSVf & hsvf,
        sValidityParameter param,
        int useLimits);
```

Arguments of the function are "rgb" values of the current pixel as defined in the sRGB structure; calculated values of the Hue, Saturation, Value, and Validity flag of the above pixel as defined in the sHSVF structure; "param" specifying parameters used as limits for the processing of this pixel; and flag useLimits, which can be set to FALSE or TRUE. When this flag is set to FALSE, this values of Hue and Saturation are calculated in the "classical way". The value of the "V" can be calculated in any situation. The code and comments of the code fragments presented in Figure 2 are self-explanatory.

Obviously, the reverse function ConvertHSVfToRGB is defined as:

```
void ConvertHSVfToRGB(
    sHSVf hsvf,
    sRGB& rgb);
```

Additionally, were defined functions converting source image to the Hue, Saturation, and Value maps, and to the image presenting validityFlag values as a human-readable gray map, in which different values are encoded by using different levels of gray.

To demonstrate this approach to the well-known procedure of "recoloring", the exemplary function ChangeHue was defined as:

```
void ChangeHue(
        unsigned char trueColorSource[][..][..],
        unsigned char trueColorDestination[][..][..],
        double oldHue,  double newHue,
        double hueHalfRange,
        sValidityParameter param, int useLimits);
```

It must be noted that most recoloring algorithms replace the specified value of Hue with a new one. But this approach is adequate only for synthetic images. Hence, in this function, the additional parameter "hueHalfRange" is added. Then, all pixels having valid values of Hue in the range from (oldHue – hueHalfRange) to the (oldHue + hueHalfRange) will be replaced with the "newHue" value. Naturally, this function uses validity parameters to exclude from the processing problematic pixels. By setting values of those parameters different image processing and color manipulation effects can be achieved.

IV. EXAMPLES OF ANALYSIS AND PROCESSING OF SYNTHETIC AND REAL-LIFE IMAGES

The upper left image in Figure 3 presents a synthetic image specially prepared for the tests of a modified approach and its HSVF maps. The upper part of this image is strips having different gray levels and 6 basic colors Pseudo random noise with a magnitude of 5 units was added. The effect of this noise is clearly seen in the top right map of the Validity flag. The central part of the synthetic image is a pure green ramp, whereas the lower part of this image is a sum of a gray ramp with a green ramp. The lower left image in Figure 3 represents the Hue map, whereas the lower right image represents a Saturation map. Detailed analysis of images presented in Figure 3 validated that the HSVF algorithm works as expected.

Figure 4 represents an example of color manipulation with a real-life photo. Yellow Lemons in the left image were marked in the right image by using non-natural Blue color. In this image, only one lemon was not properly illuminated by the Sun, so all lemons (except the darker part of one lemon) were successfully recognized and recolored.

The left top image in Figure 5 presents somehow problematic real-life photo. In this photo, some mandarins and leaves are in the deep shadow, whereas some mandarins are clearly overbleached. Parameters of the algorithms were selected in such a way, that only reliably validated parts of the mandarins will be recolored to the blue color (see top right image). The lower left image presents a fragment of the Hue map, whereas the lower right image represents a fragment of the Saturation map of the above photo. Problematic pixels are marked as green (overbleached) and red (too dark).

V. SUMMARY AND CONCLUSIONS

An exemplary recoloring algorithm can be fine-tuned by setting relevant parameters and flags. Examples presented in section IV demonstrated that the described approach could be used to analyze and process real-life photos. It is planned to add more parameters and flags in future research.

REFERENCES

[1] J. Russ, and F.Neal, "*The Image Processing Handbook*", CRC Press, 2017.

[2] R. Jain, R. Kasturi, and D. Schunck, "*Machine Vision*", MIT Press and McGraw Hill, Inc. 1995.

[3] J. Foley, A. van Dam, S. Feiner, and J. Hughes , "*Computer Graphics Principles and Practice*". Second Edition in C, ADDISON-WESLEY, 1997.

[4] J. Siyi and C. Heng, "*Color Recognition Algorithm Based on Color Mapping Knowledge for wooden Building Image", Scientific Programming*, Volume 2022, pp 1-15, 2022.

[5] S. Kosolapov, "*Comparison of Robust Color Recognition Algorithms*", Journal of International Scientific Publications, Materials, Methods and Technologies, Volume 15, pp 274 – 283, 2021.

[6] S. Kosolapov, "*Evaluation of Robust Color Recognition Algorithms*", Journal of International Scientific Publications, Materials, Methods & Technologies, Volume 16, pp 83-93, 2022.

 ISBN: 978-1-68558-048-3

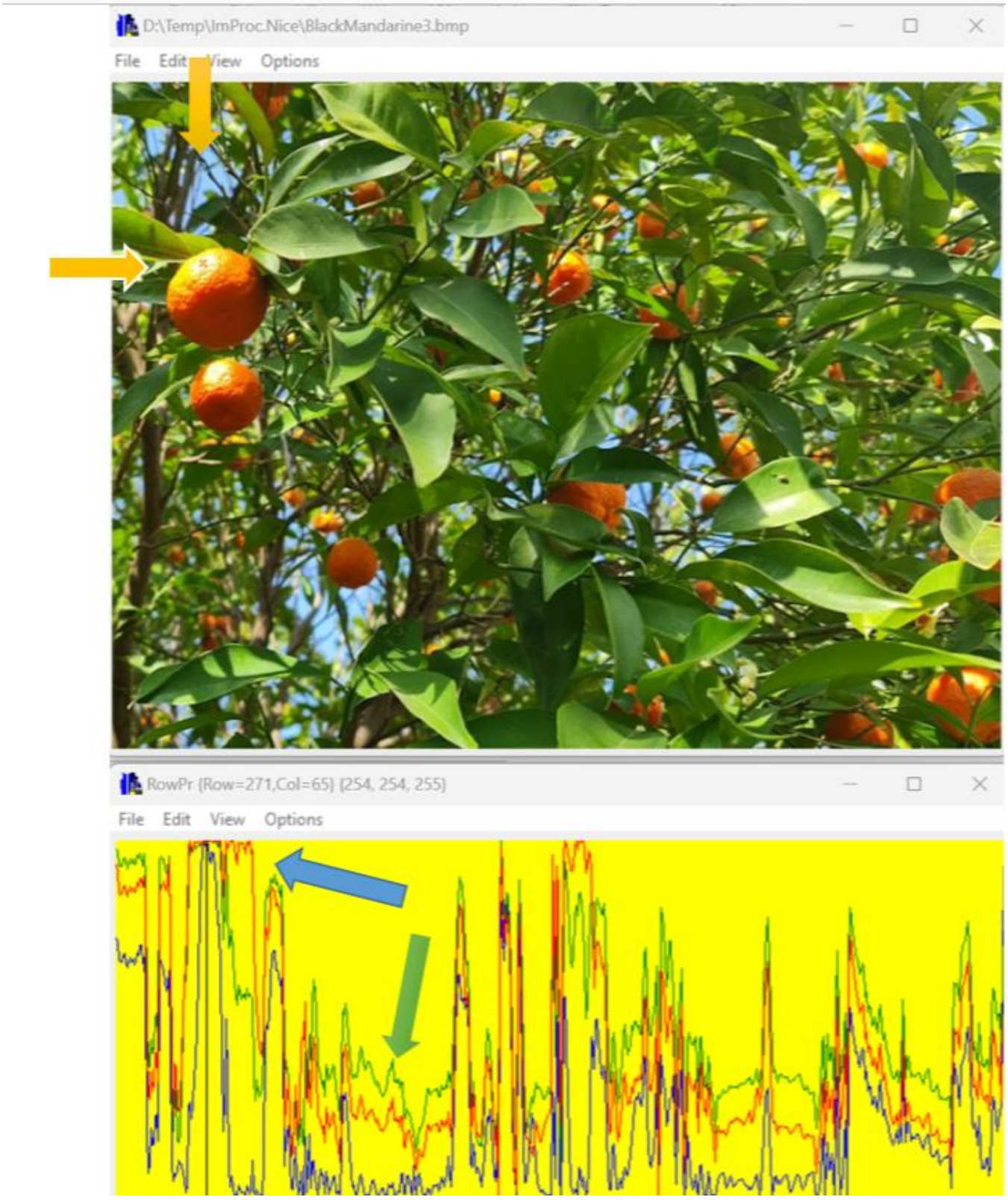

Figure 1. Real-life image of mandarines and profile of the row marked by a cross and by orange arrows. Blue arrow points to the overbleached region. Gren arrow points to the unrealistically low blue values.

```
if (useLimits == TRUE)
{
    if ((rgb.r == 255) || (rgb.g == 255) || (rgb.b == 255))
    {
        hsvf.validityFlag = IS_OVERBLEACHED;
        hsvf.V = 255;//  Can always be set
        hsvf.H = 0;  //  Non reliable: flag is set
        hsvf.S = 0;  //  Non reliable: flag is set
        return;
    }
```

```
double mean = 1.0 * (rgb.r + rgb.g + rgb.b) / 3.0;

if (mean < param.rgbMeanMin)
{
    hsvf.validityFlag = IS_TOO_DARK;
    hsvf.V = 0;  //  Can always be set
    hsvf.H = 0;  //  Non reliable: flag is set
    hsvf.S = 0;  //  Non reliable: flag is set
    return;
}
```

```
if (mean > param.rgbMeanMax)
{
    hsvf.validityFlag = IS_TOO_LIGHT;
    hsvf.V = 255;//  Can always be set
    hsvf.H = 0;  //  Non reliable: flag is set
    hsvf.S = 0;  //  Non reliable: flag is set
    return;
}
```

```
double delta = max - min;

if (delta == 0)  // Pixel is Gray
{
    hsvf.validityFlag = IS_TOO_GRAY;
    //  V was already calculated
    hsvf.S = 0;  //  Saturation is 0
    hsvf.H = 0;  // Hue cannot be calculated, flag is set
    return;
}
```

```
if (useLimits == TRUE)
{
    if (hsvf.S < param.SaturationMin)
    {
        //  V was already calculated
        hsvf.S = 0; // S calculation is not reliable, hence is set to 0
        hsvf.H = 0; // Hue calculation is not reliable, hence is set to 0
        hsvf.validityFlag = IS_TOO_GRAY;
        return;
    }
}
```

Figure 2. Extracts of the code from the file HSV.cpp demonstrating some situations when validity flags are set to the relevant values.

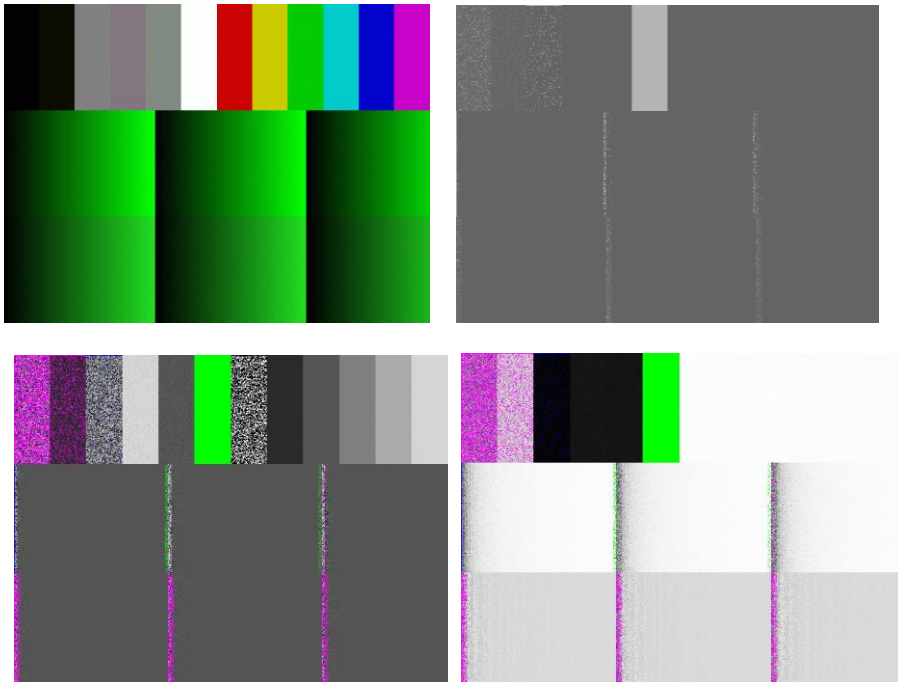

Figure 3. Upper left: Synthetic image with simulated noise. Upper right: Map of validity flags. Lower left: Hue Map. Lower right: Saturation map.

Figure 4. Left: Real-life image of lemons. Right: Example of color manipulation: Objects having Hue in the selected range of values are recolored to the blue color.

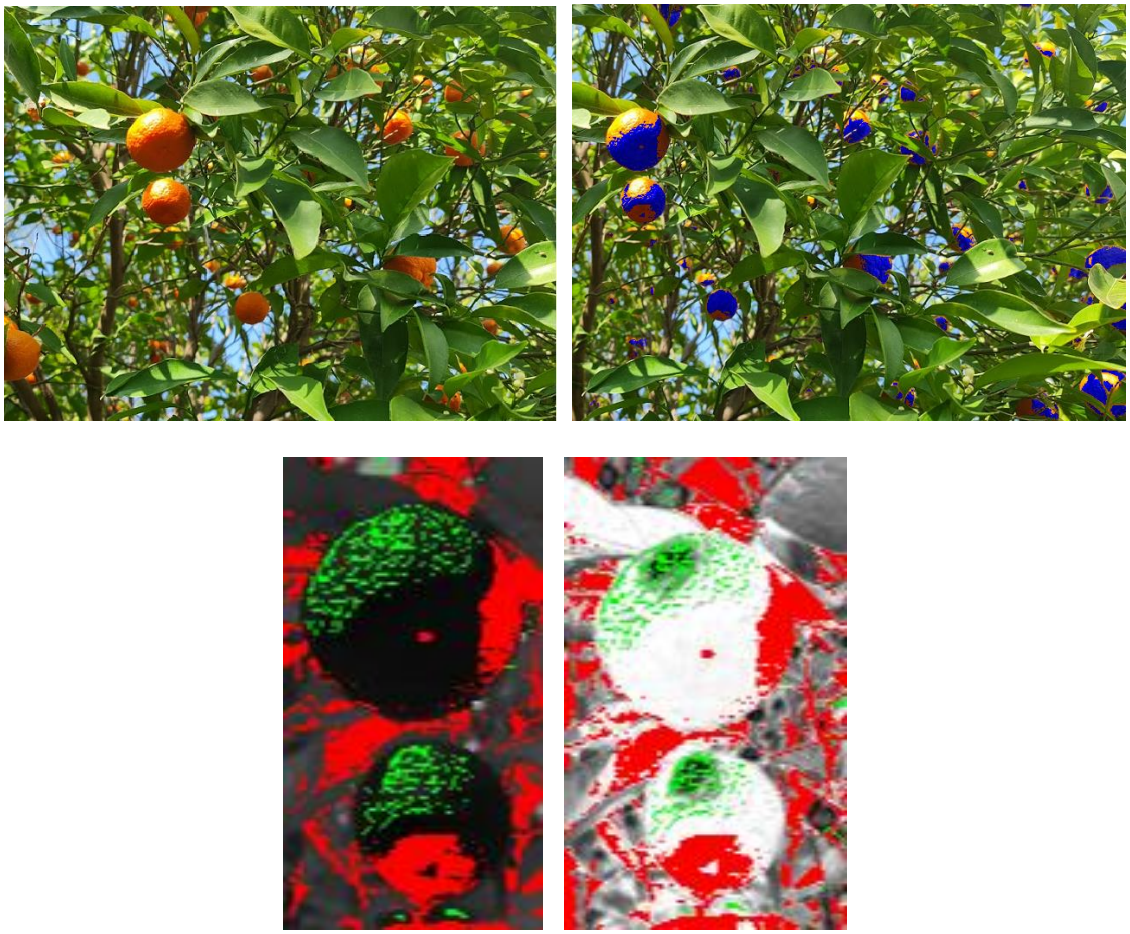

Figure 5. Top Left: Real-life image of mandarins. Top Right: Right: Example of color manipulation: Objects having reliably calculated Hue values in the selected region of the Hue values are recolored to the blue color. Bottom left: Extract from the Hue Map. Bottom right: Extract from the Saturation map. Problematic regions are marked as green (overbleached) and red (too dark)

 ISBN: 978-1-68558-048-3