Reliable Color Recognition in Images by Using a Modified HSV Algorithm

Samuel Kosolapov
Department of Electronics
Braude 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 a natural choice for executing color recognition 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. Practically, for dark regions, nearly gray regions, and for overbleached areas, the calculation of the Hue must be considered as non-reliable. Inherent to digital camera noise, makes the calculations of Hue in the above situations 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. To make the code more clear, names of the fields in this structure were changed. The specific pixel now can be flagged as "IS VALID", "IS GRAY", "HAS DARK COLOR COMPONENT", "HAS OVERBLEACHED COLOR COMPONENT", "IS NEARLY GRAY". This flag may be instrumental for reliable color recognition and for reliable modification of the color of the pixels in accordance with the selected rules. By selecting values of the "sValidityParameters", the algorithm user can specify situations when the color of the specific pixel is set to the pre-defined value, marking problematic situations. The examples provided 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; reliable color recognition; reliable color manipulations;*.**

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 day's vision color receptors. However, for applications used in machine vision, this presentation is not always practical. An alternative "traditional" 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 [2]-[4]. Functions converting pixels in the RGB space to HSV and HSL spaces and back are well known and can be found in any programming language, including C-language. The value of H actually describes the pixel's color and thus can be used to recognize the pixel's color in a simple and convenient way.

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 [5]. A number of approaches to recognize fruits [6] and specifically apples [7] were tested. However, algorithms of that type use additional information about objects to be recognized and are in most cases too heavy for real-life applications.

Unfortunately, "traditional" 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, "traditional" presentation HSV and HSL become 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, in a practical situation, 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 Hue cannot be calculated in this situation. 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. Even though 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.

Known properties of digital cameras' noise' require a rethinking of the way of Hue calculations in a number of additional problematic situations. For the synthetic image (an image created by software), an exemplary pixel {R=100, G=0, B=0} can be described as a pixel having pure RED color. However, for the digital camera that has 5 units of noise, those zero values are electronically problematic. The same is valid for the value equal to 255: this value, in most cases, means that the object is overbleached and that its color is distorted by clipping. Those and some other situations make the "traditional" HSV/HSL approach at least problematic for real-life applications.

The upper image in Figure 1 demonstrates a real-life photo of the mandarins. This image was taken with a typical smartphone camera. Nowadays, those cameras have powerful internal automatic image processing algorithms. In most cases, this results in an image optimized for a human observer. However, this, in many situations, results in color

distortion, which is "bearable" and even "preferable" for a human observer but is obviously a "no-go" for color recognition algorithms designed for a machine vision application. Callout P1a points to the pixel of the mandarin, the color of which is clearly different from other pixels of the same mandarin. Analyzing the profile of the line containing this pixel, one can see that the colors of the pixels in this region are clearly distorted: overbleached. Specific pixel marked as P1a on the image has a color (R=255, G=254, B=150) (see callout P1B). Value 255 for a real-life image must be considered as an "electronically clipped value" and must be treated as a "distorted value." Analyzing the pixel marked as P2a of the image, one can see that this pixel has a value {R=105, G=30, B=0} (see callout P2B on the lower profile). Value 0 for a real-life image must be considered as an "electronically clipped value" and treated as a "distorted value." Even though human observer probably recognize those pixels as "pixels of orange", machine vision must reject Hue's calculations of those pixels as non-reliable.

In order to better specify the problem with "traditional" Hue calculations, a special synthetic image was created. This synthetic Image (presented in Figure 2 - upper left image) has 6 strips {R, G, B} values of each strip are specified in the callouts. In order to demonstrate the challenge of reliable Hue calculations, strips have a slight green tint: the green component is "amplified" by adding values like 1 (in strip #1), 2 (in strip #2), etc. Additionally, low-level pseudo-random noise was added. Profile (presented in Figure 2 in the lower right image) and scaled-up fragment of strip #2 with saturation set to maximum (see the lower right image in Figure 2) demonstrate the effect of noise on the color of the specific pixel: when the color of the pixel is close to GRAY, the effect of noise can be significant (see strips #1, #2, and #3). It is clearly seen that while the original colors are a clear mixture of GREEN and GRAY, the colors in the lower right fragment in Figure 2 are actually pseudo-random. This effect of noise can be seen numerically in Figure 3. The upper image represents the Hue map of the original image after the addition of the noise. For strip #6 (the right part of the synthetic image), Hue value is about 85 – that is – mostly green (see an explanation of the Hue values in the Hue map later in the text). However, for strips #1, #2, and #3 (left parts of the synthetic image), the Hue value jumps unpredictably from 0 to 250, which means that the color in those strips is changed unpredictably, making the resulting values of Hue in this regions at least unreliable.

Again, even if the human observer probably recognizes pixels of this synthetic image as GREEN (because the human brain definitely can effectively eliminate noise in many practically important situations, machine vision algorithms must reject Hue's calculations of those noised pixels as non-reliable and mark them as "problematic".

Earlier attempts to improve the HSV/HSL algorithm were described in [8] and [9]. The last published version is described in [1]. This article describes a more elaborate approach based on a number of additional modifications of the previously published versions in an attempt to provide a simpler algorithm that enables the implementation of reliable color recognition and reliable color manipulation.

As in the previous versions, an extended structure, "sHSVF," is used, in which F stands for a "validityFlag".

Based on the previous experiments, the structure "sValidityParameters" was modified. Values of the fields in this structure make it possible to classify the specific pixel as "IS VALID", "IS GRAY", "HAS DARK COLOR COMPONENT", "HAS OVERBLEACHED COLOR COMPONENT", "IS NEARLY GRAY", and properly set the "validityFlag" for each pixel of the image. By selecting values of the "sValidityParameters", the user of the modified HSV algorithm can specify situations when the gray level of the specific pixel in the H-map is changed to the pre-defined color value, clearly marking problematic situations.

Section II describes the definitions of "sHSVF" structure (subsection 'A'), "sValidityParameter" (subsection 'B'), and flags used in specific situations (subsection 'C').

Section III presents changes in the "traditional" 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 "sHSVf" 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 colorComponentMinValue;
        double colorComponentMaxValue;
        double saturationMinValue;
};
```

Usage of this structure will be described later.

C. Definitions of FLAGS

In order to properly mark different situations, a number of FLAGS were defined.

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

This flag is set when the value of HUE can be calculated. Traditionally, the Hue value is in the range {0..359}. However, in order to present Hue values as a gray image on the PC monitor, a scaling factor of 255/360 is applied. Then, the pixel in the Hue map is presented as a gray pixel having a value in the range {0..255}. For example, for the GREEN Hue, the "traditional" value is 120. However, in the Hue map (after applying the factor 255/360), the resulting value is 85. For the saturation map, value 0 means that the pixel is gray, whereas value 255 means that this pixel has a pure color (maximal saturation).

The following flags are used to mark a problematic situation in which Hue cannot be calculated, or this calculation can be considered as non-reliable. All problematic pixels are marked as COLOR (and not gray) pixels so that, as a human operator, as a machine vision algorithm can easily discard problematic pixels from a recognition process.

```
#define IS_GRAY (1)
```

In the classical HSV algorithm, DELTA of the specific pixel is calculated as a difference between the maximal and minimal values of the {R, G, B} values of the specific pixel. If DELTA is ZERO, this means that R=G=B and that this pixel is gray. In this case, Hue value cannot be calculated. Hence, this flag is set when DELTA of the specific pixel is ZERO. In this situation, the problematic pixel in the Hue map and in the Saturation map has MAGENTA color.

```
#define HAS_DARK_COLOR_COMPONENT (2)
```

This flag is set when at least one of the {R, G, B} values of the specific pixel is lower than the value specified in the parameter "colorComponentMinValue" in the structure "sValidityParameter". This situation is electronically problematic, hence those pixels of the Hue and Saturation maps are marked by a RED color.

```
#define
HAS_OVERBLEACHED_COLOR_COMPONENT (3)
```

This flag is set when at least one of the {R, G, B} values of the specific pixel is higher than the value specified in the parameter "colorComponentMaxValue" in the structure "sValidityParameter". This situation is electronically problematic hence those pixels of the HUE and Saturation maps are marked by a YELLOW color.

```
#define IS_NEARLY_GRAY (4)
```

This flag is set if the calculated saturation value is lower than the value of "saturationMinValue" in the structure "sValidityParameter." A GREEN color marks those pixels on the Hue and Saturation maps.

Again, if none of the above flags were set, "sValidityParameter" would be set to the IS_VALID value (defined as zero). In this case, 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 to the goal of this research.

III. CHANGES IN THE CLASSICAL HSV ALGORITHM

Classical function RGBtoHSV which is described at [2]-[4], and C-code of which is available in the public domain was modified by adding flags defined before. A complete code of the reworked function "ConvertRGBtoHSVf" is presented in Figure 4. 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; values of the Hue, Saturation, Value, and Validity Flag of the above pixel to be calculated as defined in the "sHSVF" structure; "param" - specifying parameters used for the processing of this pixel; and flag "useLimits", which can be set to FALSE or TRUE. The value of the "V" can be calculated in any situation. When this flag is set to FALSE, values of Hue and Saturation are calculated in the "traditional way" without taking into account validity parameters (see lines 9-62 in Figure 4). Then, in the situation when Hue value cannot be calculated, values of Hue and Saturation are set to 0, which creates some well-known ambiguity: Hue=0 can mean RED color and/or GRAY pixel. When this flag is set to TRUE, {R, G, B} values of the pixel in the test are compared with the values specified in the "param" values (see lines 66-100 in Figure 4). A relevant color marks problematic pixels (see lines 76, in Figure 4), which enables to exclude them later from the following image analysis.

The reverse function ConvertHSVfToRGB is defined as:

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

This function has no significant differences from the "traditional" code.

As a reasonable self-test, a sequence of functions "ConvertRGBtoHSVf" - "ConvertHSVfToRGB" was run for a different sets of {R,G,B} values. No errors were found during those tests.

Additionally, 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 were defined.

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. However, this approach is adequate only for synthetic images. The noise of the cameras "widens" the values of the selected Hue value. Hence, the additional parameter "hueHalfRange" is added to this function. 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 problematic pixels from processing. 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 image in Figure 5 presents a synthetic image specially prepared for the tests of a modified approach and its HSVF maps. This image contains 12 strips. The first strips are nearly gray (see callouts), whereas the last six have colors close to the six primary colors. The lower image presents profiles. Figure 6 demonstrates a color map of the upper image created by using a "traditional" algorithm (flag "useLimits" was set to "FALSE"). In this situation, three strips were marked as problematic: Hue cannot be calculated in those cases. Then, a low-level pseudorandom noise was applied to the image presented in Figure 6. The upper image in Figure 7 demonstrates profiles after the addition of the noise, whereas the lower image represents the Hue map. Hue recognition for most strips (except strips #4 and #5) is problematic – Hue calculations in those specially created problematic situations are non-reliable.

The upper left image in Figure 8 represents GREEN RAMP mixed with low-intensity RED RAMP. The lower left image represents the profile of this image. The upper right image represents the Hue map, and the lower right image is the profile of the Hue Map. It can be seen that Hue's calculations are not reliable for the "dark" regions. Hue values calculated for other parts of the ramp are noisy but reliable.

The upper left image in Figure 9 is a slightly problematic real-life photo of lemons and mandarins. The upper right image represents a Hue map calculated with the flag "useLimits" set to TRUE. Lower images represent corresponding profiles. Callout P1 points to the pixel having overbleached values for all color components; they are seen as pure white. They are marked by MAGENTA color, which is reserved for GRAY color (white in the HSV concept is gray because the saturation in this case is ZERO). Callout P2 points to the Hue value of a "typical" lemon. Callout P3 points to the pixel having a Hue value of the "typical" mandarin. The left image in Figure 10 represents parts of lemons as they were recognized by the modified algorithm. It can be seen that nearly all parts of lemons are reliably recognized and "recolored' to a strong blue color (except small overbleached regions marked by callout {253, 255, 254} and a small part of "too dark" lemons, which was not recolored). The right image demonstrates reliable recognition of mandarins.

Figure 11 represents another problematic image of mandarins. It can be seen that overbleached pixels are excluded from recognition.

V. Summary and Conclusions

The presented recognition and recoloring algorithm can be fine-tuned by setting dedicated for that goal parameters and flags. Examples given in Section IV demonstrated that the described approach could be used to analyze and process real-life photos. It is planned to rewrite the algorithm by using pointers and Q-numbers to improve the speed of the algorithm.

References

[1] S. Kosolapov, "*Color Manipulation in Images by Using a Modified HSV Algorithm*", Think Mind, CONTENT 2023: The Fifteenth International Conference on Creative Content Technologies, Think Mind, pp. 1-6, 2023.

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

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

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

[5] 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.

[6] W.C. Seng and S.H. Mirisaee, "*A new method for fruits recognition system*", Electrical Engineering and Informatics, 2009. ICEEI '09. International Conference, Volume 01, 2009.

[7] W.Ji, D. Zhoo et al,, "*Automatic recognition vision system guided for apple harvesting robot*", Computers & Electrical Engineering, Volume 38, Issue 5, pp. 1186 – 1195, September 2012.

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

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

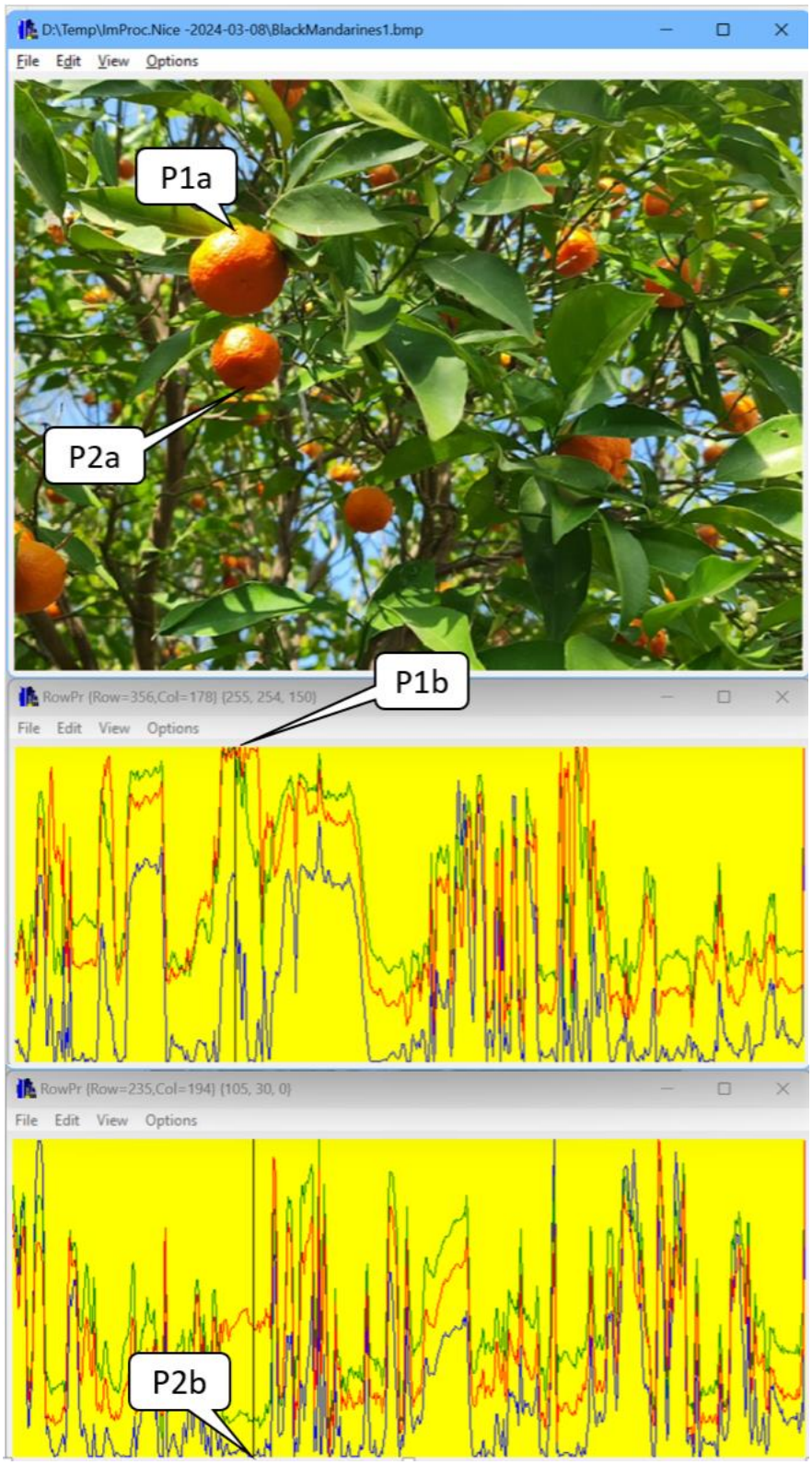

Figure 1. Upper image: real-life image of mandarines obtained by using typical smartphone camera (Samsung Galaxy Note 20). Image in the middle: profile for the line marked as P1a. The color of the pixel P1a is {R=255, G=245, B=150} (marked as P1b). It is clearly seen, that color of this pixel is distorted (overbleached). The Lower image is a profile of the line marked as P2a. The color of the pixel is {R=105, G=30, B=0} (marked as P2b). It can be stated that the color of this pixel is distorted (B component has a zero value).

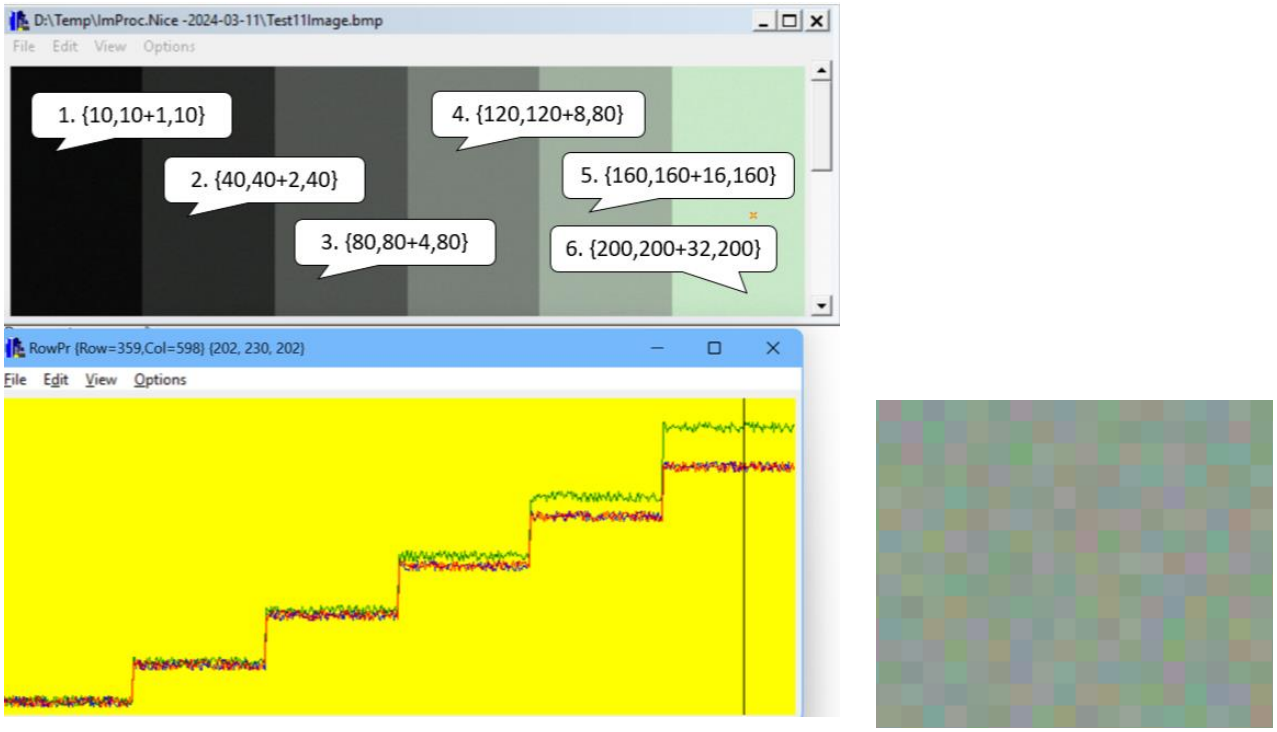

Figure 2. Upper Left: Synthetic image – 6 strips, having mix of GRAY + GREEN colors. (see callouts). Pseudo-random noise is added. Lower left: Profile of a row = 359. Specific values of the pixel of cursor are {R=202, G= 230, B = 202}, and because of noise values are slightly differ from declared values {R=200, G=232, B=200}. Lower right: scaled UP fragment from the strip #2. Saturation and brightness were set to maximum to emphasize that because of noise, nearly gray pixels (with some green tint) now have pseudorandom colors (red, green, blue, etc.).

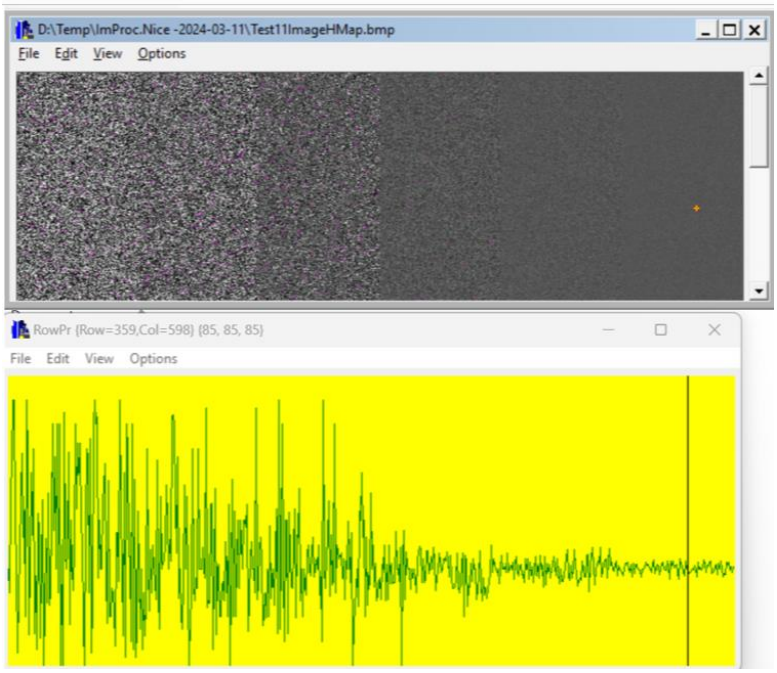

Figure 3. Upper image: Hue map calculated by a “traditional” algorithm. Lower image: profile of the upper image. It can be seen that the Hue value in strip #6 is about 85 (which corresponds to GREEN). Additionally, it can be seen that for lower intensities, Hue value is nearly pseudorandom – so that colors of the pixels are nearly pseudorandom – as it can seen in the lower right image in Figure 2.

Figure 4. Full code of the modified function ConvertRGBtoHSVf. Code between lines 3-62 is a classical well-known code. Lines 54-99 contains proposed modifications. See additional explanations in the text.

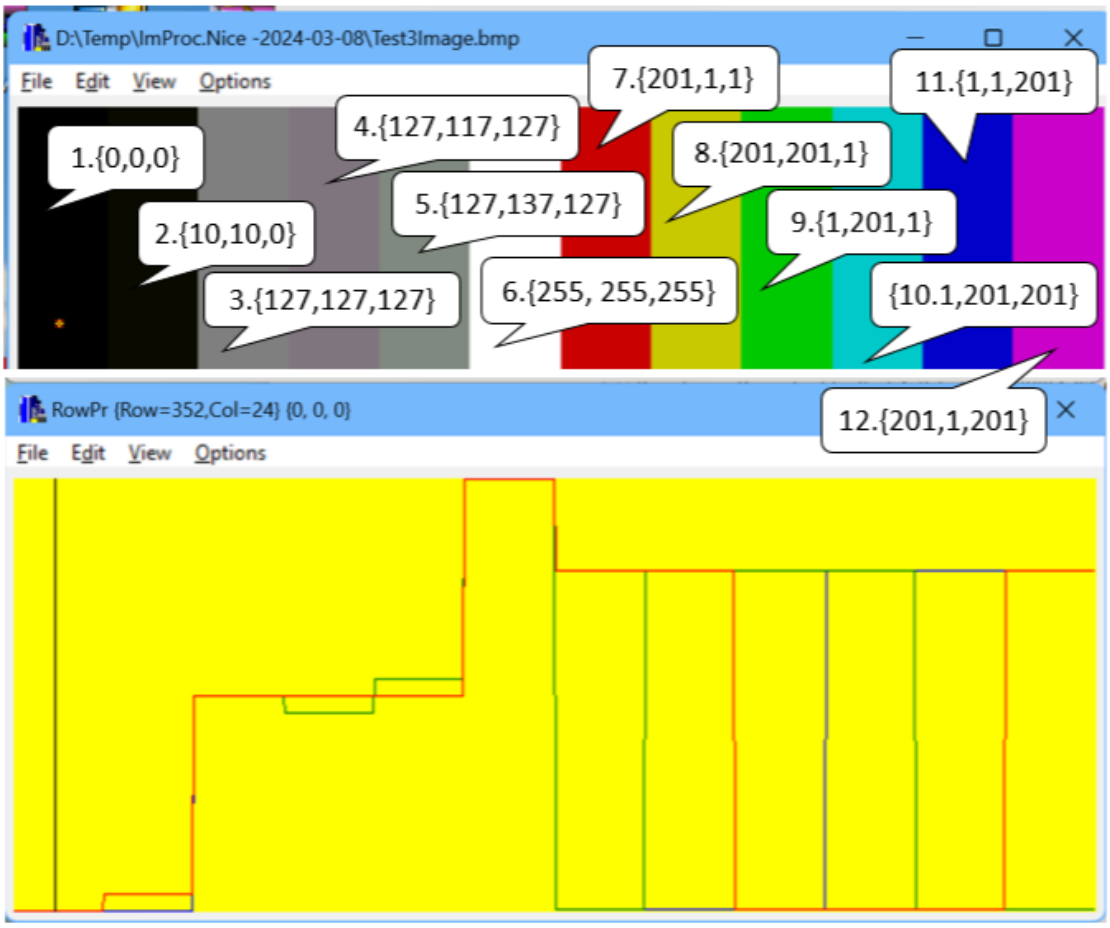

Figure 5. Upper Image: Synthetic Test Image containing 12 strips having selected {R, G, B} values. Callouts specified those values for every strip. Lower image: R, G and B profiles of the upper image. The strip #1 is totally black : {0,0,0}. Considering future tests, some 0 values were shifted to value 1, or to another value (see values in callouts). See additional explanations in the text.

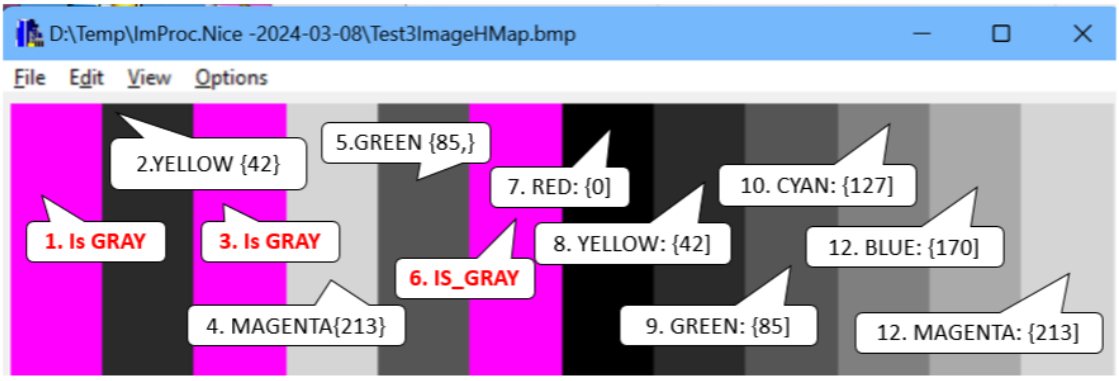

Figure 6. H-Map of the Test Image presented in Figure 5. Flag “useLimits” was set to “FALSE”, hence the processing was provided by a “traditional” part of the function “ConvertRGBtoHSVf”, without modifications. The strips #1, #3, and #6 were marked by the “traditional” algorithm by MAGENTA color because their original pixels are exact gray pixels (see callouts in Figure 5. In this case, pair of strips #2 and #8 has the same Hue =42 (corresponding to a YELLOW color), despite the fact that most human observers will not be able to detect color of the very dark strip #2. The same is true for strips #5 and #9 (Hue = 85, which corresponds to the GREEN color of the strips). In this case, the human observer can validate that strip #5 has a GREEN tone. Original {R,G,B} values {127, 137, 127} (see callout 5 in Figure 5) confirmed that this color is a mix of gray {127, 127, 127} with dark GREEN {0,10,10}.

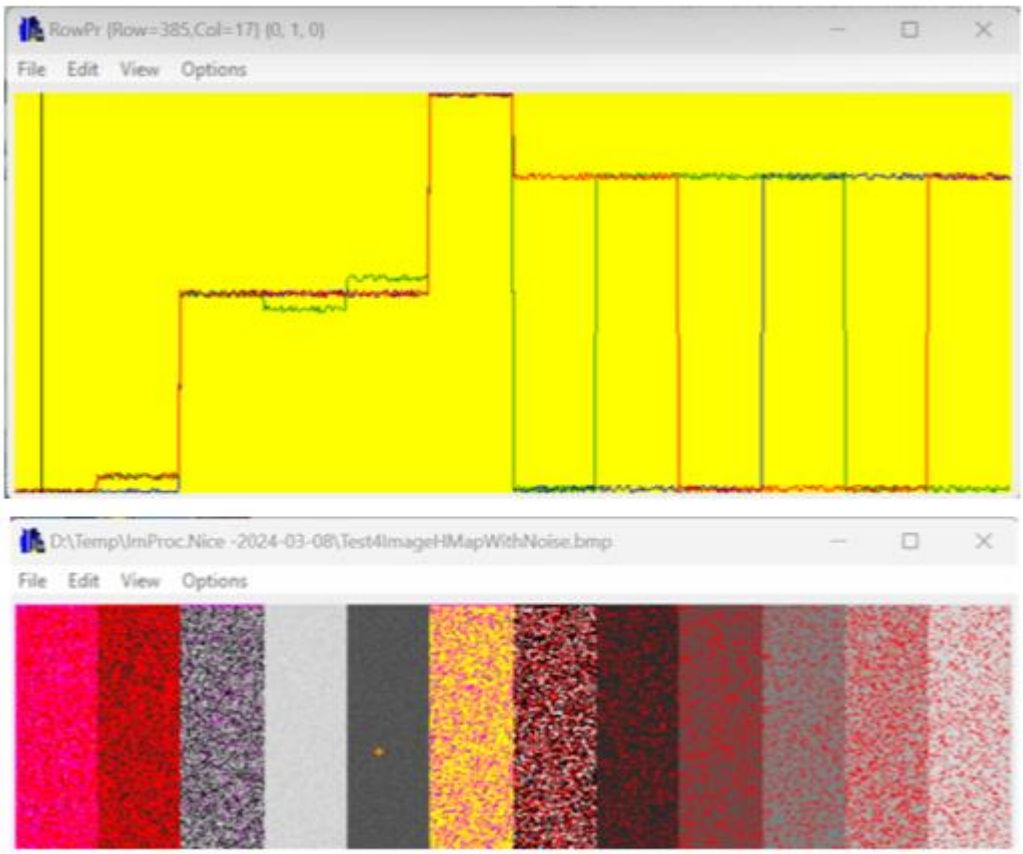

Figure 7. Pseudo-random noise was added to the Test Image from the Figure 5. This pseudo-random noise can be clearly seen in the Upper Image – profiler of row 385. Lower image: Flag "useLimits" was set to "TRUE"; hence the processing was provided with a "modified" part of the function ConvertRGBtoHSVf. It is clear that even small level of noise makes color recognition for the selected values problematic.

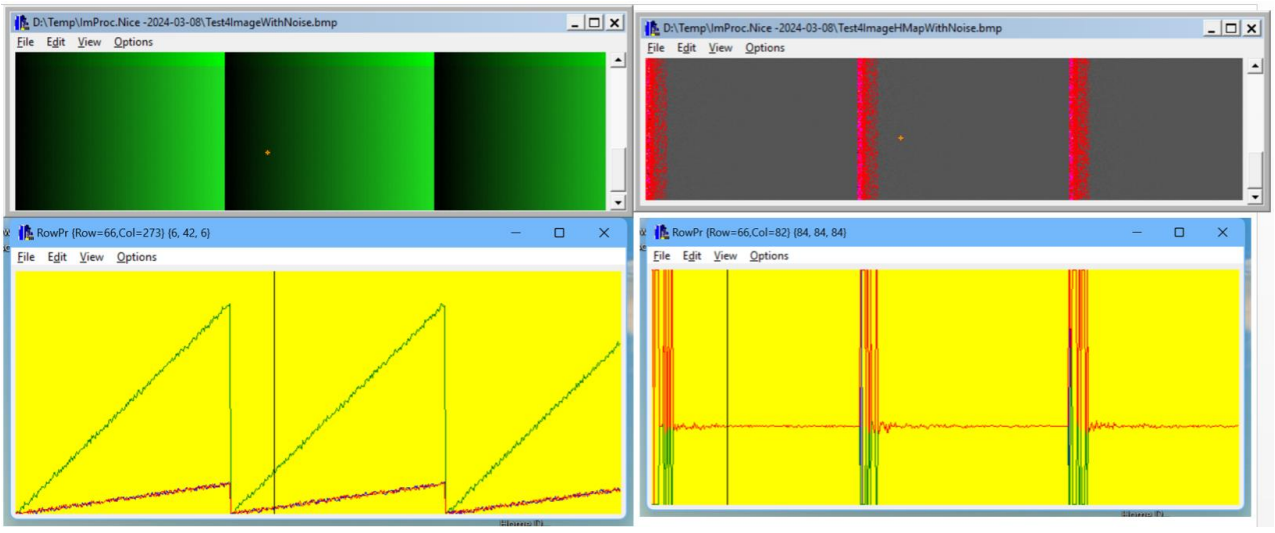

Figure 8. Upper left: Synthetic image GREEN RAMP mixed with low-ntensity RED RAMP. A low-level pseudo-random noise was added. This noise can be seen in the Lower Left image: profile of the Upper left image. Upper right image: Hue map of the Upper Left image. It can be seen that for most pixels reliable color recognition is possible (Hue value is about 85 which corresponds to GREEN – see Lower Right profile). However, values of Hue for dark" regions of this ramp are marked by RED color (signalling that at least some values of some color components are too low).

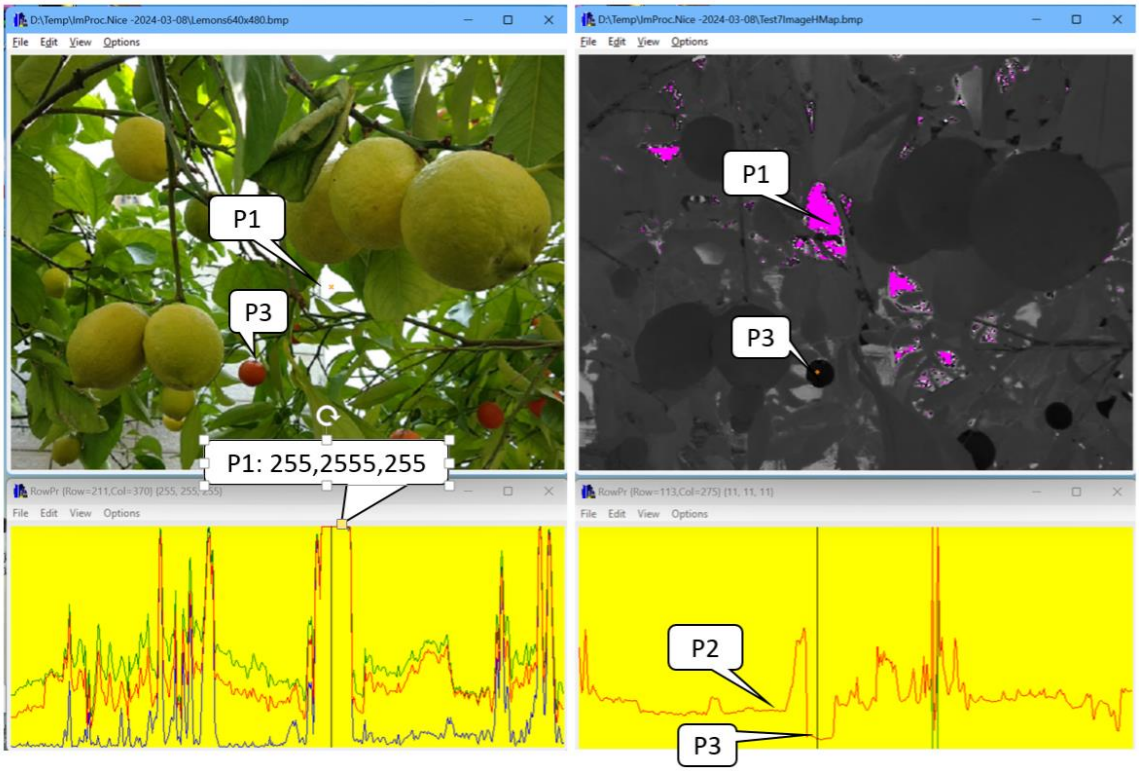

Figure 9. Upper left: Real-life image of lemons and mandarines. Upper right: H-Map of the Upper left image. Lower right: profile of the line of pixel marked as P1. Lower right: profile of row marked as P3. See the discussion in the text.

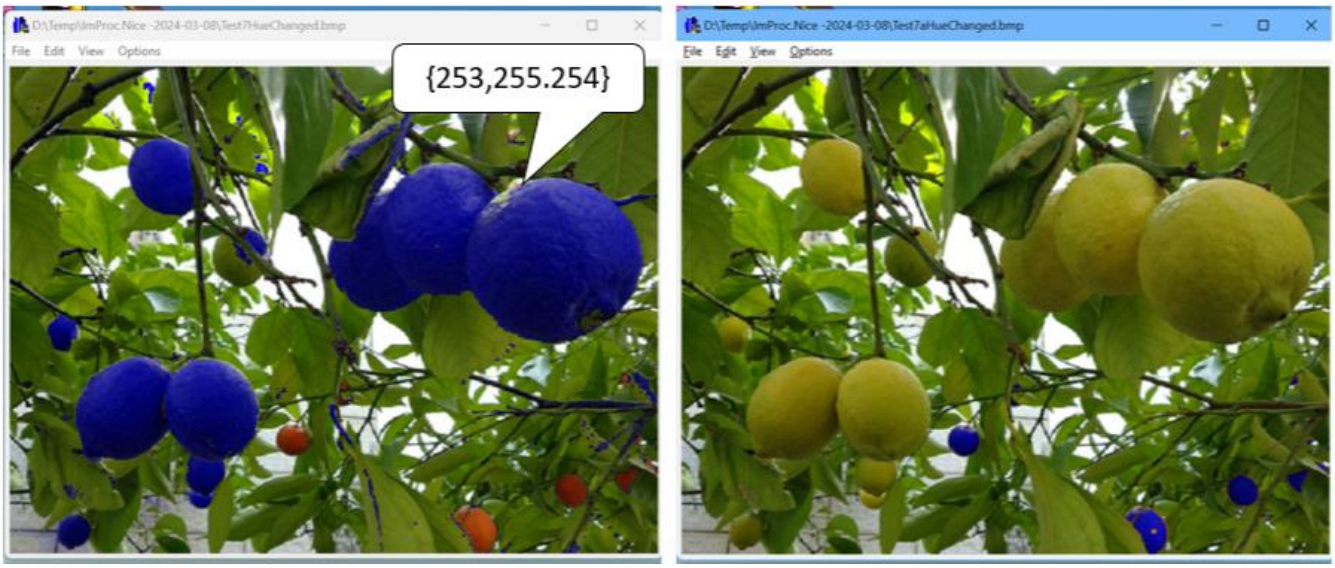

Figure 10. Left image: Recognition of lemons from the upper left image in Figure 9. Recognized parts of lemons are recolored to the bright BLUE color. Non-recognized pixels retain their original colors. Right Image: Recognition of mandarins from the upper left image in Figure 9. Recognized parts of mandarines are recolored to the bright BLUE color. Non-recognized pixels retain their original color. See additonal details in the text.

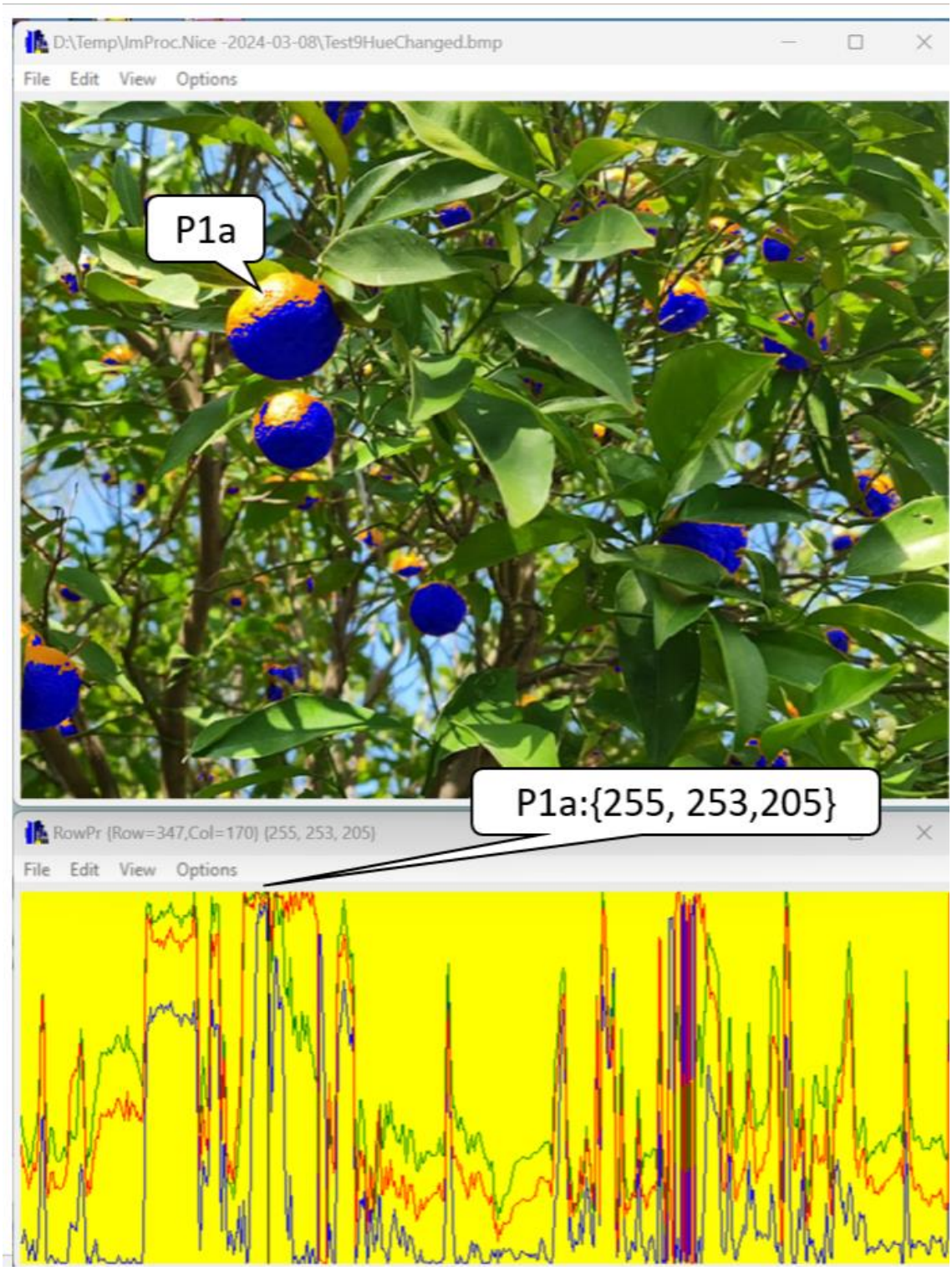

Figure 11. Upper image: results of recognition of the mandarines from the image presented in Figure 1. Lower image: profile of the line marked as P1a. Pixels near P1a have an obvousely distorted color {255, 253. 205}. Experienced human operator without hesitations would mark those pixels as “part of the mandarine”; however, in the frames of the discussed approach, they must not be recognized as having reliable Hue value, and, thus, those pixels were not recolored.