AMERICAN UNIVERSITY OF ARMENIA
College of Science and Engineering
CS 260 Image Processing

FINAL PROJECT – Color-based Face Detection

Deadline: Tuesday, December 20 2022, no later than 22:00 **SHARP**
Submission format: Electronic
Submit to: skhachat@aua.am, grigor_aghajanyan@edu.aua.am
Subject line: CS260/371 Project, your AUA ID#

Project Objective

Face detection has been a subject of active studies for several decades due to its high practical value. A survey on skin color-based methods in face detection is conducted in [1]. The easiest and fastest skin classifiers are implemented by explicit relationships between color space coordinates. Such rules are normally found empirically. It is also shown that they can be established by machine learning algorithms. The project aims at formulation, study and testing of iterative face detection and recognition algorithms based on the concept of a Face-Relevant Operational Gamut (FROG) – a layered region in RGB color space significant amount of facial pixels belong to [2].

Face-relevant Colors

Let us consider in RGB color space face images of different people taken in different places by different cameras under usual lighting conditions – daylight, camera flash, bulb light, etc. The representative set of face-relevant colors S is the set of unique (R, G, B) triplets from more than 130000 collected pixels. The frequencies of occurrence of the selected colors on face images are discarded in S, which makes it practically independent from the training set provided an adequate size of the latter. It is convenient to represent the set S as a union of subsets $S(G)$ with constant value of the green component:

$$S = \bigcup_{G=0}^{255} S(G)$$

The following statistical parameters for $(R + B) / 2$ are computed within each such subset:

$s_0(G) = \min\{(R + B) / 2, (R, G, B) \in S(G)\}$,
$s_4(G) = \max\{(R + B) / 2, (R, G, B) \in S(G)\}$,
$s_2(G) = \text{mean}\{(R + B) / 2, (R, G, B) \in S(G)\}$,
$s_1(G) = s_2(G) - \sigma\{(R + B) / 2, (R, G, B) \in S(G)\}$,
$s_3(G) = s_2(G) + \sigma\{(R + B) / 2, (R, G, B) \in S(G)\}$,

where σ is the standard deviation.

Figure 1(a) depicts in succession these series, with $s_0(G)$ being at the bottom. As shown in Table 1, quadratic regression perfectly fits all of them. Therefore, in a 2D space of $(G, (R + B) / 2)$ we get the Face-Relevant Operational Gamut – an analytically defined region of face-relevant colors depicted in Figure 1(b). With the help of the derived curves a set of pixel selection rules is defined as

$$f_{ij}: B < G \;\&\; G < R \;\&\; \text{bottom}(G) \le (R + B) / 2 \le \text{top}(G), \qquad (1)$$

where $\text{bottom}(G) = s_i(G)$ and $\text{top}(G) = s_j(G)$ $(0 \le i < j \le 4)$.

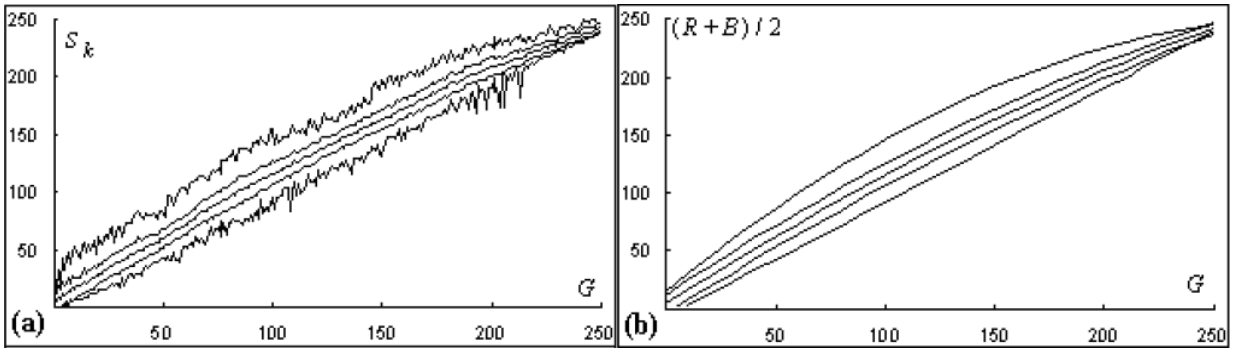


Figure 1 (a) $s_k(G)$ series, $0 \le k \le 4$. The curves are ordered from $k = 0$ at the bottom to $k = 4$ at the top; (b) the Face-Relevant Operational Gamut in 2D space of (G, $(R + B) / 2$).

Table 1 Quadratic approximation of $s_k(G)$ series

	Trend line	**R^2**
s_0	0.9848 G - 6.7474	99.63%
s_1	-0.0009 G^2 + 1.1917 G – 4.0146	99.98%
s_2	-0.0011 G^2 + 1.2262 G + 4.0264	99.96%
s_3	-0.0013 G^2 + 1.2608 G + 12.067	99.91%
s_4	-0.0026 G^2 + 1.5713 G + 14.8	98.86%

The code of ImageJ plugin for f_{01} (**Binary_Layer_0.java**) is given below. Obviously, the codes of ImageJ plugins for f_{12} (**Binary_Layer_1.java**), f_{23} (**Binary_Layer_2.java**) and f_{34} (**Binary_Layer_3.java**) differ only in the return statements of the **bottom()** and **top()** methods.

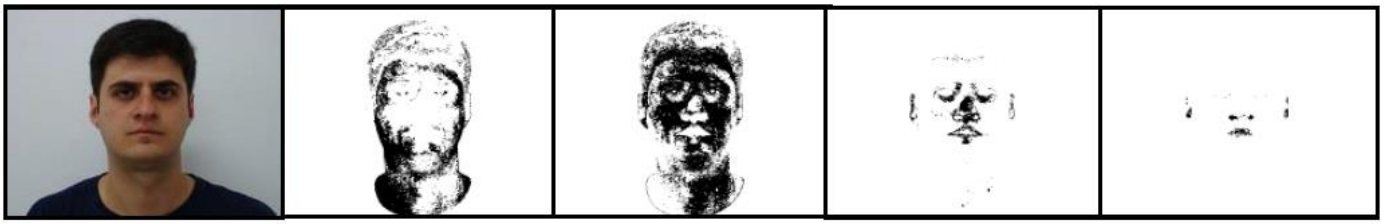
```
// Binary_Layer_0.java

import ij.ImagePlus;
import ij.process.ImageProcessor;
import ij.plugin.filter.PlugInFilter;
import java.awt.Color;

public class Binary_Layer_0 implements PlugInFilter {

	public double bottom(int x) {
		return 0.9848 * x - 6.7474;
	}

	public double top(int x) {
		return -0.0009 * x * x + 1.1917 * x - 4.0146;
	}

	public int setup(String args, ImagePlus im) {
		return DOES_RGB;
	}

	public void run(ImageProcessor ip) {
		int width = ip.getWidth(), height = ip.getHeight(), pixel, r, g, b;
		double rb;
		Color color;

		for (int row = 0; row < height; row++)
			for (int col = 0; col < width; col++) {
				color = new Color(ip.getPixel(col, row));
				r = color.getRed();
				g = color.getGreen();
```

```
                        b = color.getBlue();
                        rb = (r + b) / 2.;
                        if (b < g && g < r && rb >= bottom(g) && rb <= top(g))
                              ip.putPixel(col, row, 0); //BLACK
                        else
                              ip.putPixel(col, row, 16777215); //WHITE
                  }
            }
}

// in Binary_Layer_1.java

      public double bottom(int x) {
            return -0.0009 * x * x + 1.1917 * x - 4.0146;
      }

      public double top(int x) {
            return -0.0011 * x * x + 1.2262 * x + 4.0264;
      }

// in Binary_Layer_2.java

      public double bottom(int x) {
            return -0.0011 * x * x + 1.2262 * x + 4.0264;
      }

      public double top(int x) {
            return -0.0013 * x * x + 1.2608 * x + 12.067;
      }

// in Binary_Layer_3.java

      public double bottom(int x) {
            return -0.0013 * x * x + 1.2608 * x + 12.067;
      }

      public double top(int x) {
            return -0.0026 * x * x + 1.5713 * x + 14.8;
      }
```

For your convenience, ImageJ plugins **All_Layers.java** and **Binalry_Layers.java** are uploaded in the \Home Works folder of the course Moodle site. The former colors the pixels filtered by f_{01}, f_{12}, f_{23} and f_{34} with green, yellow, read and pin respectively, and the latter generates four separate black-and-white images for each rule.

Task 0: Create a repository and submit its URL. All project deliverables should be uploaded there. An individual dataset will be assigned. Download the image files from the **FEI Face Database** (https://fei.edu.br/~cet/facedatabase.html). The files with names starting from **1** to **50** inclusive should be extracted from **originalimages_part1.zip** archive, with names from **51** to **100** inclusive – from **originalimages_part2.zip**, with names from **101** to **150**– from **originalimages_part3.zip**, and from **151** to **200** – from **originalimages_part4.zip**.

Task 1: Many frontal images from the FEI database (with **x_11.jpg** file name format) exhibit the following behavior: f_{01} selects pixels mainly concentrated along the face bounds, f_{12} – the majority of the skin pixels, f_{23} – pixels from the central region around the nose, eyebrows and lips, and f_{34} – pixels from the lips and, possibly, ears. Let's call it Standard Behavior. An example is shown below.



Apply different filters (Mean, Median, Gaussian Blur, etc.) before and / or after the extraction of the Binary Layers as to produce connected and smoothened regions and minimize or remove insignificant details. Try filters of different radii, and study the influence of the image / face size on them. Use commands from ImageJ **Process ▷ Filters** sub-menu.

Task 2: In many applications it is important to locate some *central* region of a face image. Consider a simple algorithm based on the Standard Behavior:

1. Extract pixels by f_{12}, f_{23} and f_{34} – the images **layer_1_**, **layer_2_** and **layer_3_** after running the **Binary_Layers.java** plugin. Make them binary images using ImageJ command **Process ▷ Binary ▷ Make Binary**.
2. Fill holes using ImageJ command **Process ▷ Binary ▷ Fill Holes**.
3. Leave only the common pixels using **AND** operation in ImageJ command **Process ▷ Image Calculator**.
4. Compute the centroid of the detected region using ImageJ commands **Analyze ▷ Set Measurements** and **Analyze ▷ Analyze Particles**. Save in the repository the coordinates of the centroid and the resulting image of the detected region under **Task2_<original filename>** name.

Test this algorithm on frontal images **xxx-11.jpg**, **xxx-12.jpg**, **xxx-13.jpg** from your dataset. For the convenience, you may record a macro with the mentioned above commands using ImageJ command **Plugins ▷ Macros ▷ Record**.

Task 3: Consider an alternative approach and apply it to the frontal images:

1. Apply the result of **Task 1** to the original image to reduce the noise in subsequent steps.
2. Extract pixels by f_{12}, f_{23} and f_{34} – ensure each of the images **layer_1_**, **layer_2_** and **layer_3_** contain only prominent connected regions.
3. Analyze particles in all three images and show the ellipses in the **Analyze ▷ Analyze Particles** command.
4. Collect the ellipses in one image using **OR** operation in ImageJ command **Process ▷ Image Calculator**. Save the result in the repository under **Task3_<original filename>** name.
5. Select nested ellipses, where the outer ellipse from the **layer_1_** image contains a smaller ellipse from the **layer_2_** image, and the latter contains an inner ellipse from the **layer_3_** image. Compare the centroid of the inner ellipse with the result of **Task 2**.

Task 4: Apply the algorithm from **Task 3** to rotated images from your dataset and study a relationship between the layout of the nested ellipses and the rotation angle. Assume the faces are rotated by the same 20° angle on the images from **xxx-01.jpg** to **xxx-10.jpg**.

Task 5: Consider below two methods to estimate the face boundaries and compare the obtained results:

Method 1:

1. Apply the result of **Task 1** to the original image.
2. Extract pixels by f_{01}.
3. Skeletonize the binary regions in the binary image **layer_0_**, and save the result in the repository under **Task5M1_<original filename>** name.

Method 2:

1. Apply the result of **Task 1** to the original image.
2. Extract pixels by f_{01} and f_{12}.
3. Dilate the binary regions in the binary images **layer_0_** and **layer_1_**.
4. Leave only the common pixels using the **AND** operation, and save the result in the repository under **Task5M2_<original filename>** name.

Task 6: Relatively low precision of the color-based algorithms can be justified by their application to low-resolution images, where the number of pixels is small and, therefore, the processing time is much shorter.

1. Write an ImageJ plugin that generates a low-resolution copy of an opened original image from your dataset. The width and the height of the generated copy are 5 times smaller than those of the original one. The original image is divided into 5×5 boxes, and the arithmetic mean of 25 pixel values from each box is assigned to the corresponding pixel of the copy image.
2. Apply the algorithms from the previous tasks to the generated low-resolution image.
3. Process additionally the binary regions as needed to ensure comparable accuracy with the high-resolution case.
4. Save in the repository all reproduced results under **Task6_<high-resolution task result filename>** name.

Submission Conditions:

1. This is an individual assignment. Identical or similar submissions / files / results / reports / diagrams etc. will be disqualified – both the source(s) and receiver(s) will collect 0 point.
2. Group work will be accepted only if all group members are explicitly indicated in the submission. The individual contribution of each group member must also be explicitly stated, including all reasons of forming the group.
3. The submission deadline is rigidly strict. Submit even an unfinished work to get points and feedback. Late submissions will be disqualified and collect 0 point.
4. Not only precise solutions, but also free-format descriptions of ideas, difficulties, algorithms, simplifications, assumptions, etc. may be submitted.
5. All used external sources must be explicitly acknowledged.
6. **ATTENTION**: Project Support Sessions (PSS) will run until the submission deadline – on Tuesdays December 13 and 20 at 19:00, on Thursday December 15 at 19:00 and on Saturday December 17 at 12:00.

References

1. V. Vezhnevets, V. Sazonov, A. Andreeva. A Survey on Pixel-Based Skin Color Detection Techniques, *Proc. of the 13th GraphiCon*, Moscow, 2003, pp.85-92.
2. S. Khachatryan,. FLID Color Based Iterative Face Detection (see Face_Detection.pdf in the \Home Works folder of the course Moodle site).