Introduction to computer vision

In general, computer vision covers very wide area of issues concerning understanding of images by computers. It may be considered as a part of artificial intelligence and aims at designing processes of visual analysis which would make it possible to extract relevant information from images, similarly as it is done by human. This task should be performed fully automatically with high speed and effectiveness comparable or higher than achieved by people.

Image Color Conversion

CIE Chromaticity Diagram and Color Gamut

Figure 1 presents a diagram of all visible colors. It is called a chromaticity diagram and was developed as a result of the experimental investigations performed by CIE (International Commission on Illumination), see http://members.eunet.at/cie/. The diagram presents visible colors as a function of *x* (red) and *y* (green) components called *chromaticity coordinates*. Positions of various spectrum colors (from violet to red) are indicated as the points of tongue-shaped curve called *spectrum locus*. The straight line connecting the ends of the curve is called the *purple line*. The point of equal energy represents the CIE standard for white light. Any point within the diagram represents some mixture of spectrum colors. The pure or fully saturated colors lie on the spectrum locus. Straight-line segment joining any two points in the diagram defines all color variations than can be obtained by additively combining these two colors. A triangle with vertices at any three points determine the gamut of colors that can be obtained by combining corresponding three colors. The structure of the human eye that distinguishes three different stimuli, establishes the three-dimensional nature of color. The color may be described with a set of three parameters called tristimulus values, or components. These values may, for example, be dominant wavelength, purity, and luminance, or so-called primary colors: red, green, and blue. The chromaticity diagram exhibits that the gamut of any three fixed colors can not enclose all visible colors. For example, Figure 1 shows schematically the gamut of reproducible colors for the *RGB* primaries of a typical color CRT monitor, *CMYK* color printing, and for the *NTSC* television.

Color Models

The purpose of a color model is to facilitate the specification of colors in some standard generally accepted way. In essence, a color model is a specification of a 3-D coordinate system and a subspace within that system where each color is represented by a single point. Each industry that uses color employs the most suitable color model. For example, *RGB* color model is used in computer graphics, and *YUV* or YC_bC_r are used in video systems, *PhotoYCC** is used in PhotoCD* production and so on. Transferring color information from one industry to another requires transformation from one set of values to another. The Intel IPP provides a wide number of functions to convert different color spaces to *RGB* and vice versa.

The RGB Color Model

In the *RGB* model, each color appears as a combination of red, green, and blue. This model is called additive, and the colors are called primary colors. The primary colors can be added to produce the secondary colors of light (see Figure 2) - magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). The combination of red, green, and blue at full intensities makes white. The color subspace of interest is a cube shown in Figure 2 (*RGB* values are normalized to 0..1), in which *RGB* values are at three corners; cyan, magenta, and yellow are the three other corners, black is at their origin; and white is at the corner farthest from the origin.

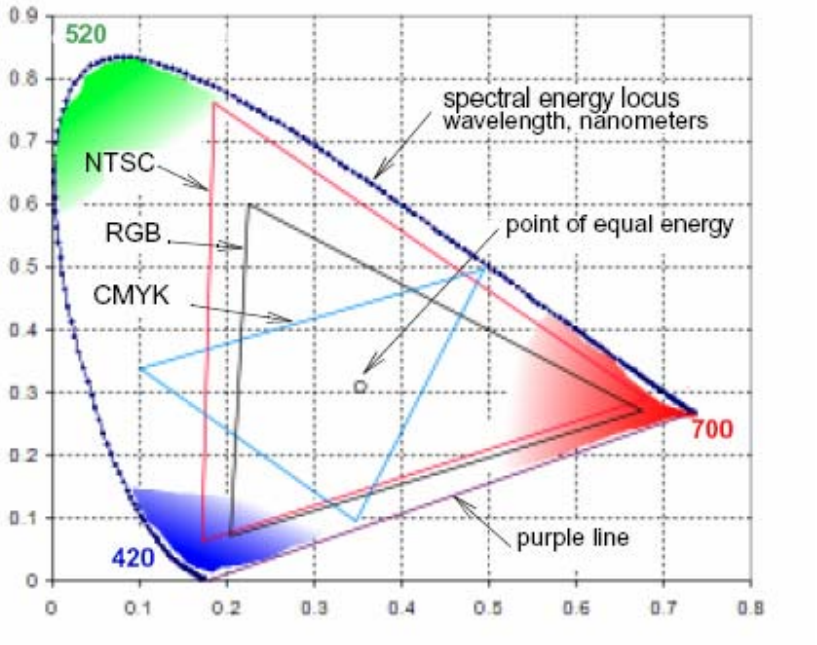

Figure 1. CIE xyY *Chromaticity Diagram and Color Gamut*

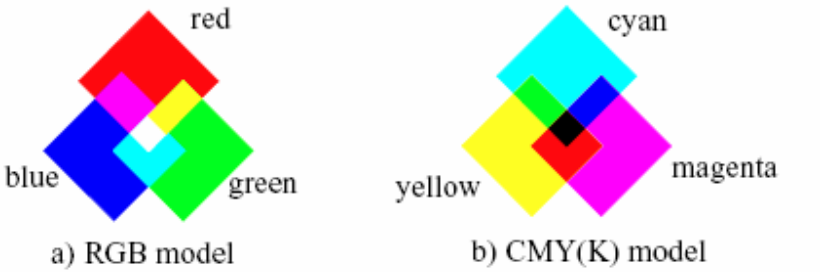

Figure 2. Primary and Secondary Colors for RGB and CMYK Models

The gray scale extends from black to white along the diagonal joining these two points. The colors are the points on or inside the cube, defined by vectors extending from the origin. Thus, images in the *RGB* color model consist of three independent image planes, one for each primary

color. As a rule, Intel IPP color conversion functions operate with non-linear gamma-corrected images *R'G'B'*. The importance of the *RGB* color model is that it relates very closely to the way that the human eye perceives color. *RGB* is a basic color model for computer graphics because color displays use red, green, and blue to create the desired color. Therefore, the choice of the *RGB* color space simplifies the architecture and design of the system. Also, a system that is designed using the *RGB* color space can take advantage of a large number of existing software routines, since this color space has been around for a number of years.

However, *RGB* is not very efficient when dealing with real-world images. To generate any color within the *RGB* color cube, all three *RGB* components need to be of equal pixel depth and display resolution. Also, any modification of the image requires modification of all three planes.

The CMYK Color Model

The *CMYK* color model is a subset of the *RGB* model and is primarily used in color print production. *CMYK* is an acronym for cyan, magenta, and yellow along with black (noted as K). The *CMYK* color space is subtractive, meaning that cyan, magenta yellow, and black pigments or inks are applied to a white surface to subtract some color from white surface to create the final color. For example (see Figure 2), cyan is white minus red, magenta is white minus green, and yellow is white minus blue. Subtracting all colors by combining the *CMY* at full saturation should, in theory, render black. However, impurities in the existing *CMY* inks make full and equal saturation impossible, and some *RGB* light does filter through, rendering a muddy brown color. Hence, the addition of black ink to *CMY*. The *CMY* cube is shown in Figure 3, in which *CMY* values are at three corners; red, green, and blue are the three other corners, white is at the origin; and black is at the corner farthest from the origin.

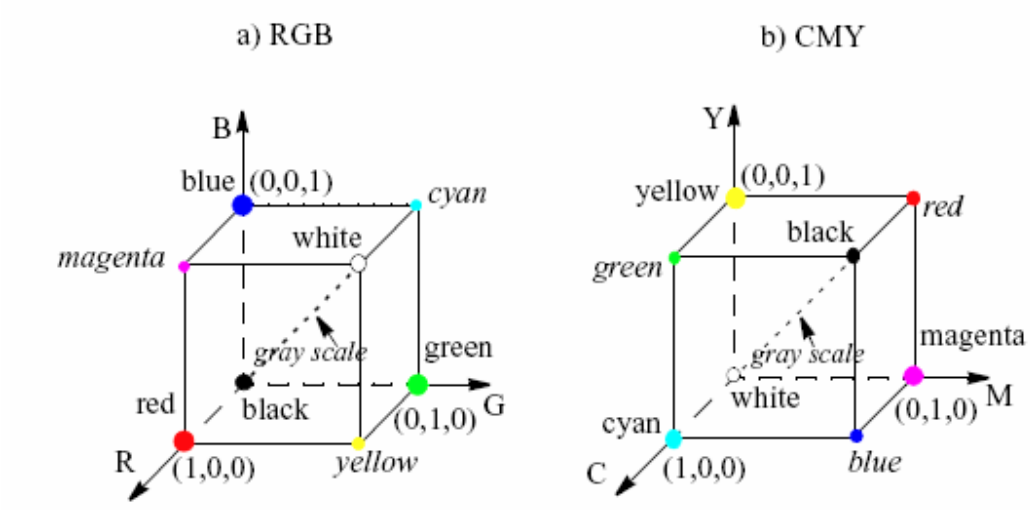

Figure 3. RGB and CMY Color Models

The YC_bC_r Color Model

The YC_bC_r color space is used for component digital video and was developed as part of the ITU-R BT.601 Recommendation. *RGB* colors cube in the YC_bC_r space is presented in Figure 4. The

Intel IPP functions use the following basic equations to convert between R'G'B' in the range 0-255 and $Y'C'_bC'_r$ (this notation means that all components are derived from gamma-corrected R'G'B'):

```
Y' = 0.257*R' + 0.504*G' + 0.098*B' + 16
Cb' = -0.148*R' - 0.291*G' + 0.439*B' + 128
Cr' = 0.439*R' - 0.368*G' - 0.071*B' + 128
R' = 1.164*(Y'-16) + 1.596*(Cr'-128)
G' = 1.164*(Y'-16) - 0.813*(Cr'-128) - 0.392*(Cb'-128)
B' = 1.164*(Y'-16) + 2.017*(Cb'-128)
```

Intel IPP color conversion functions specific for JPEG codec used different equations:

```
Y = 0.299*R + 0.5587*G + 0.114*B
Cb = -0.116874*R - 0.33126*G + 0.5*B + 128
Cr = 0.5*R - 0.41869*G - 0.08131*B + 128
R = Y + 1.402*Cr - 179,456)
G = Y - 0.34414*Cb - 0.71414*Cr + 135.45984)
B = Y + 1.772*Cb - 226.816
```

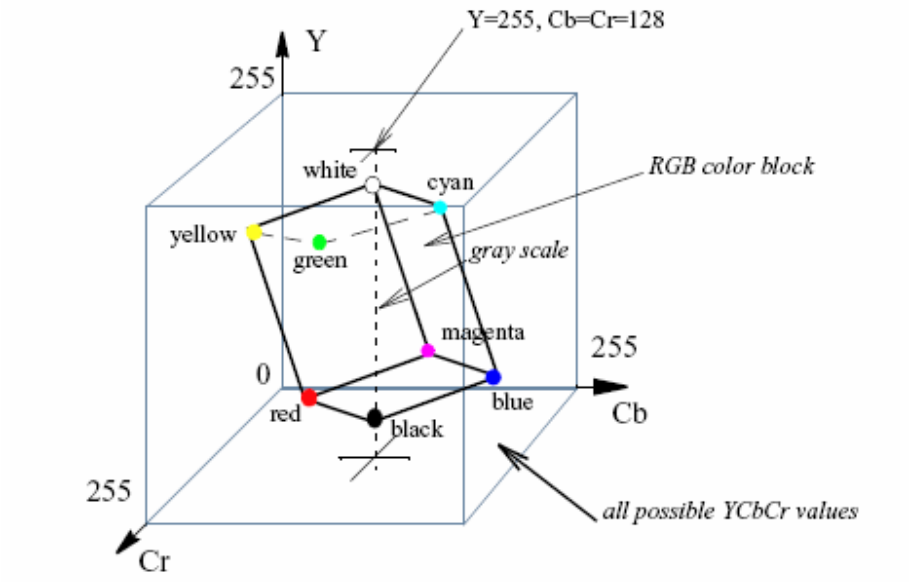

Figure 4. RGB Colors Cube in the YC_bC_r Space

HSV and HLS Color Models

The *HLS* (hue, lightness, saturation) and *HSV* (hue, saturation, value) color models were developed to be more "intuitive" in manipulating with color and were designed to approximate the way humans perceive and interpret color.

Hue defines the color itself. The values for the hue axis vary from 0 to 360 beginning and ending with red and running through green, blue and all intermediary colors.

Saturation indicates the degree to which the hue differs from a neutral gray. The values run from 0, which means no color saturation, to 1, which is the fullest saturation of a given hue at a given illumination.

Intensity component - *lightness* (*HLS*) or *value* (*HSV*), indicates the illumination level. Both vary from 0 (black, no light) to 1 (white, full illumination). The difference between the two is that maximum saturation of hue is at S=1 and full illumination (V=1) in the *HSV* color model, whereas in the *HLS* color model maximum saturation is at lightness L=0.5. The *HSV* color space is essentially a cylinder, but usually it is represented as a cone or hexagonal cone (hexcone) as shown in the Figure 5, because the hexcone defines the subset of the *HSV* space with valid *RGB* values. The *value* V is the vertical axis, and the vertex V=0 corresponds to black color. Similarly, a color solid, or 3D-representation, of the *HLS* model is a double hexcone with *lightness* as the axis, and the vertex of the second hexcone corresponding to white.

Both color models have intensity component decoupled from the color information. The *HSV* color space yields a greater dynamic range of saturation.Conversions from *RGB* to *HSV*/*HLS* and vice-versa in Intel IPP are performed in accordance with the respective pseudocode algorithms, given in the descriptions of corresponding conversion functions.

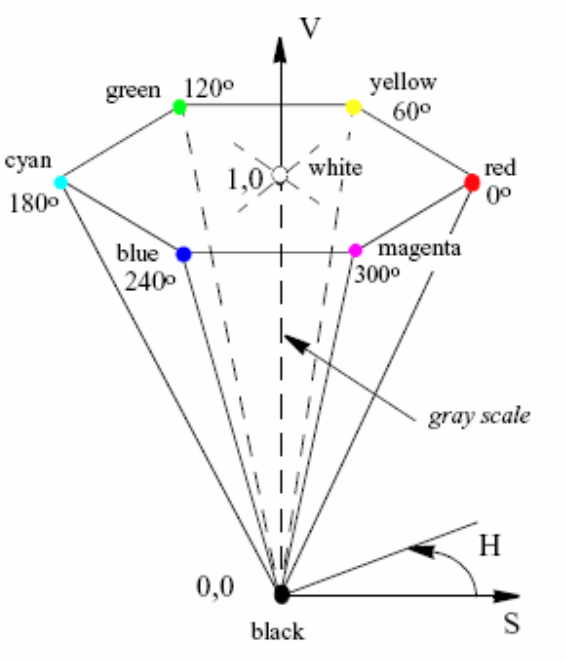

Figure 5. HSV Solid

Morphological operations

Generally, the erosion and dilation smooth the boundaries of objects without significantly changing their area. Both operations use either a symmetric 3x3 mask, a user-defined rectangular mask, or a structuring element. In a more general sense, morphological operations involve an image *A* called the *object of interest* and a kernel element *B* called the *structuring element*. The image and structuring element could be in any number of dimensions, but the most common use is with a 2D binary image, or with a 3D gray scale image. The element *B* is most often a square or a circle, but it could be any shape. Just like in convolution, *B* is a kernel or template with an

anchor point. Figure 6 shows dilation and erosion of object *A* by *B*. In the figure, *B* is rectangular with an anchor point at upper left shown as a dark square.

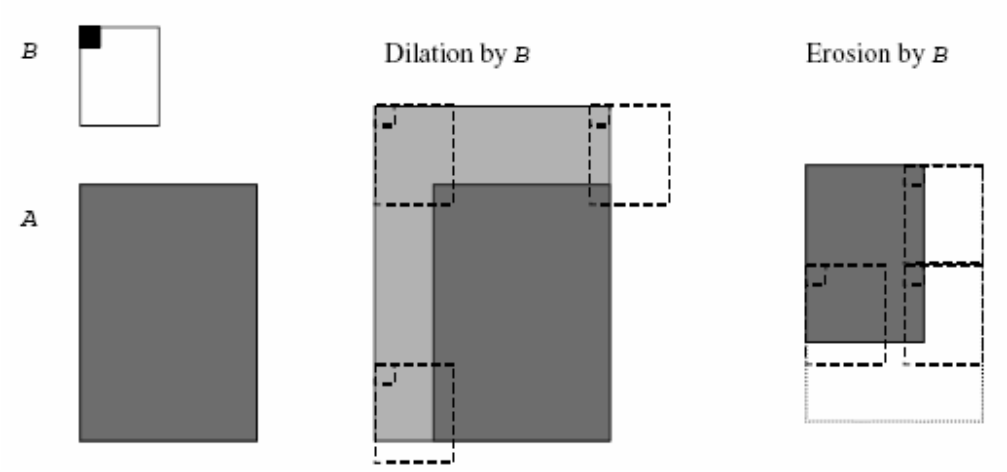

Figure 6. Dilation and Erosion of A *by* B

In case of dilation, a pixel under the anchor point of *B* is marked "on", if at least one pixel of *B* is inside of *A*. In case of erosion, a pixel under the anchor of *B* is marked "on", if *B* is entirely within *A*.

Face detection

Automatic human face recognition can be considered as one of the areas of computer vision. The first step of face recognition is face detection. An image must be analyzed in order to decide whether it contains faces or not. In case of a positive answer face or faces must be precisely located in the input image, so that it is possible to process them further.

One of the face detection methods is based on color images in the YC_bC_r color space. It may be noticed that eyes are characterized by high blue and low red values, as well as by many dark and bright pixels (Figure 7). These facts may be utilized for designing the eye map (see Equation 1 – 3). EM_C is a chrominance eye map, EM_L is a luminance eye map and EM is the final eye map created by multiplying the EM_C and EM_L maps. C_b, C_r and Y are color channels in the YC_bC_r color space, $\overline{C}_r$ is negation of the C_r channel (255 - C_r), Y_{dil} and Y_{er} are Y channels dilated and eroded, respectively. The calculated maps are after that eroded, dilated and normalized in order to eliminate false noise information, which is usually weaker than the response of eye and mouth regions. Examples of face images in various channels and corresponding eye maps are presented in Figure 7.

$$EM_C = \frac{1}{3}(C_b^{\,2} + \overline{C}_r^{\,2} + \frac{C_b}{C_r}) \qquad (1)$$

$$EM_L = \frac{Y_{dil}(p,q)}{Y_{er}(p,q)+1} \quad (2)$$

$$EM = (EM_C) \cdot (EM_L) \quad (3)$$

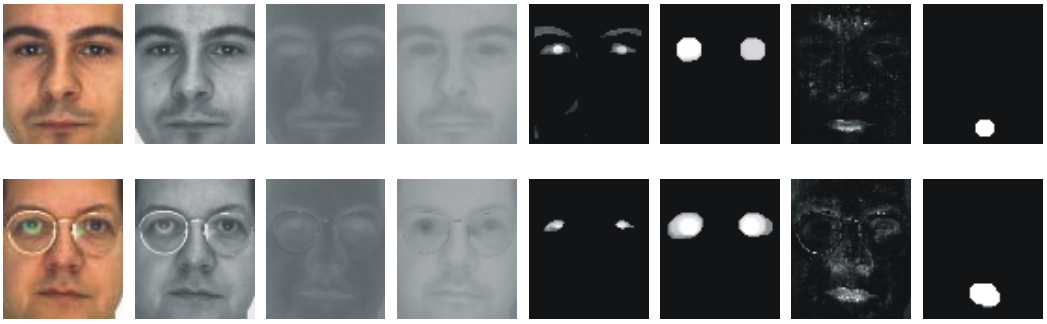

Figure 7. Examples of face images in various channels and maps which have been obtained (from the left: original image, Y *channel,* C_b *channel,* C_r *channel, two images of eye maps and two images of mouth maps). The maps are presented in two modes: achieved directly from equations.1 – 5 (left) as well as eroded and dilated versions (right). Erosion and dilation eliminates noise, which is present especially in the case of the mouth maps and extracts relevant information concerning the features location.*

A similar approach can be applied to generate a mouth map. Mouth pixels contain higher C_r values and lower C_b values comparing to other face regions. Basing on this observation, a mouth map has been constructed, which may be calculated as defined in Equation 4, where *MM* is a mouth map, η is a ratio of the average C_r^2 value ($avg(C_r)$) to the average C_r/C_b value ($avg(C_r/C_b)$). An example of a mouth map is shown in Figure 7.

$$MM = C_r^2 \cdot (C_r^2 - \eta \cdot \frac{C_r}{C_b})^2 \quad (4)$$

$$\eta = 0.95 \cdot avg(C_r^2) / avg(\frac{C_r}{C_b}) \quad (5)$$

By analyzing the eye and mouth maps, an exact location of feature points may be calculated, which means that using color information only, it is possible to detect faces and facial feature points in images.

Exercise

The aim of this exercise is to implement face detection in color images.

The exercise is based on *Vision* programming platform. The platform consists of an executable program (*vision.exe*) which utilizes *lib.dll* library file. This file is generated by compiling a

VS 7.0 C++ project (*Detection*). The functions of this project will be modified during this exercise.

When *vision.exe* is launched, a dialog box will be displayed, in which output folders should be selected. The results (face images maps and text files containing features coordinates) will be stored to these folders. By default, the paths are set to a folder, in which *vision.exe* is placed. After setting the paths a user should press *Process Files* and select files to be processed.

Functions which must be modified during the exercise are: *MouthMap*, *EyeMap*, *Detect* in *Detection* component.

MouthMap and *EyeMap* functions are called with following arguments:

- *pC_b* – a pointer to Cb channel of input image.
- *pC_r* – a pointer to Cr channel of input image.
- *nW*, *nH* – input image width and height.
- *pDst* – a pointer to output image (its size should be equal to the input image size; memory is allocated).

The functions are called for every image in the selected set of images. The results are saved in output folders and also passed to *Detect* function.

Detect function is called with following arguments:

- *pEyeMap* – eye map generated by *EyeMap* function (data written to *pDst*).
- *pMouthMap* – mouth map generated by *MouthMap* function (data written to *pDst*).
- *nW*, *nH* – image width and height.
- *nLE_X*, *nLE_Y* – left eye coordinates (to be calculated).
- *nRE_X*, *nRE_Y* – right eye coordinates (to be calculated).
- *nM_X*, *nM_Y* – mouth coordinates (to be calculated).

The function is called for each image after maps generation (when *MouthMap* and *EyeMap* return results).

There are two functions already implemented (*Dilate* and *Erode*) which can be used to perform image dilation and erosion respectively. These operations are useful for maps generation.

Images which should be processed during the exercise can be found in *img* folder. The images size is constant (64x75 pixels) and eyes are placed in fixed positions (15, 24) and (49, 24). It is therefore possible to asses eyes detection precision basing on this information.

Tasks to be accomplished:

- Fill *MouthMap* function, so that it generates mouth maps described in this instruction.
- Fill *EyeMap* function, so that it generates eye maps described in this instruction.
- Modify *Detect* function, so that it calculates eyes and mouth coordinates basing on maps generated by *MouthMap* and *EyeMap* functions.