Kyungpook National University

Artificial Brain Research - ABR

BEP

Lesson and Assignment 1

Color Space

Gwenaelle Cunha Sergio

Winter 2014

1 Summary of Lecture Material

1.1 Programming Environment

For this lecture, we'll be using the *Microsoft Visual Studio*, with the Microsoft Foundation Class Library (MFC), and the programming language will be C++. The advantages of using this library is that it the programming is object-oriented and event-driven. Furthermore, there's a toolbox with a drag and drop functionality to help build the User Interface, and the only moment when you have to write code is when an event occurs.

1.2 Color Space

A pixel is the minimal unit that constitutes an image. A color space is an abstract mathematical model that describes the colors. There are a few types of color space: RGB, CMYK, HSV and YIQ.

1.2.1 RGB

The RGB Color Space, **Fig. 1** is the most used one of them all. The R stands for Red, the B stands for Blue and the G stands for Green. Those are the three primary colors, from which one can create any other color by combining them in an addictive manner, since making this an addictive color space.

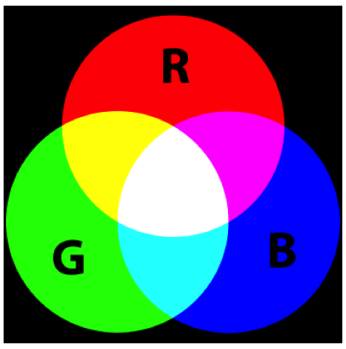

Figure 1: RGB Color Space

The values for R, G and B range from 0 to 255, but they can be normalized to range between 0 and 1; 0 meaning black and 255 meaning white. Gray images have the same value for R, G and B.

1.2.2 CMYK

The CMYK Color Space, **Fig. 2** is a subtractive color space, usually used in printing machines. The C stands for Cyan, the M stands for Magenta, the Y stands for Yellow and the K stands for blacK.

Figure 2: CMYK Color Space

Each CMYK component can be obtained from the values of the RGB components (r, g and b, normalized to range between 0 and 1) by using the following calculations [1]:

$$K = (1 - max(r,g,b))$$
$$C = \frac{1-r-k}{1-k}$$
$$M = \frac{1-g-k}{1-k}$$
$$Y = \frac{1-b-k}{1-k}$$

1.2.3 HSV

The HSV Color Space, **Fig. 3** is one of the most common "cylindrical-coordinate representations of points in an RGB color model" [2]. This color space exists for the sole reason that it is more intuitive and perceptually relevant for humans than the traditional representation. The H stands for Hue, the S stands for Saturation and the V stands for Value.

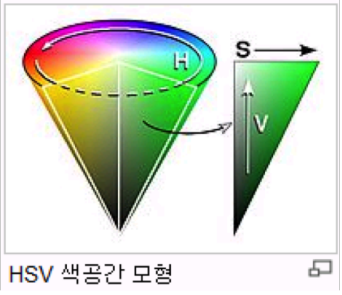

Figure 3: HSV Color Space

[1] $http://www.rapidtables.com/convert/color/rgb-to-cmyk.htm$
[2] $http://en.wikipedia.org/wiki/HSL\_and\_HSV$

1.2.4 YIQ

The YIQ Color Space, **Fig. 4** is mostly used by the NTSC color TV system and it "takes advantage of human color-response characteristics" [3], seeing that the human eye is more sensitive to changes in the orange-blue range (I) than in the purple-green range (Q). The Y component represents the luma information, and is the used by black and white television receivers, whilst the I and Q represent the chrominance information. The I stands for In-phase and the Q stands for Quadrature (it uses a quadrature amplitude modulation).

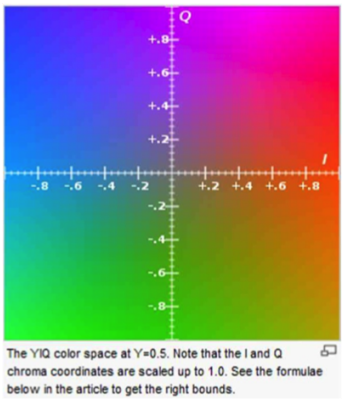

Figure 4: YIQ Color Space

1.3 OpenCV

In order to handle image processing, we had to add and configure the OpenCV library to the project. OpenCV stands for Open Computer Vision and is an open source cross-platform computer vision C library. This library makes it easier to read and display images, and it has numerous image processing functions.

For example, to show an image declared as *IplImage*, the following code applies:

```
int main(void) {
    IplImage *img = cvLoadImage(\img.bmp");   // allocation
    cvNamedWindow(\image", 1);   // creating window
    cvShowImage(\image", img);   // image displaying
    cvWaitKey(0);   // *waiting time for key input
    cvReleaseImage(&img);   // image release
    cvDestroyWindow(\image");   // window release
}
```

[3] *http://en.wikipedia.org/wiki/YIQ*

2 Assignment

This first assignment consists of creating buttons using MFC and when a button is clicked, showing each component of the RGB, CMYK and HSV color space; and tracking an eye in an image using the tools and knowledge acquired so far.

2.1 User Interface

The interface for this assignment is very simple, consisting of four buttons (**Fig. 5**). When the user clicks on the button RGB, CMYK or HSV it analyzes the image indicated on the code and shows the user the image's RGB, CMYK and HSV components in their own windows. When the fourth button is clicked, it detects the eye in an image.

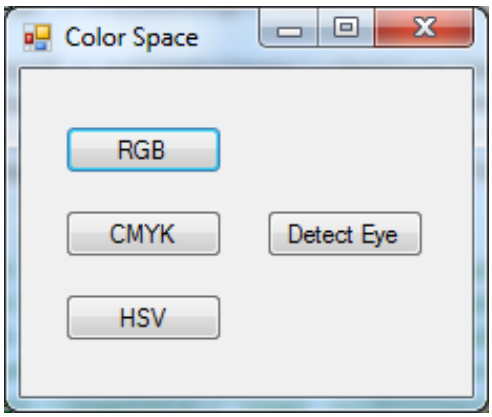

Figure 5: Microsoft Visual Interface

2.2 Color Space

Take the original image on **Fig. 6**.

Figure 6: Original image

We can obtain the three RGB components from this image by using the *cvSplit* function from the OpenCV library, as show in the following code:

```
cvSplit(image, imageB, imageG, imageR, NULL);
```

The result of that code is shown in **Fig. 7**, where it's shown, from left to right and top to bottom, the original image and the R, G and B components.

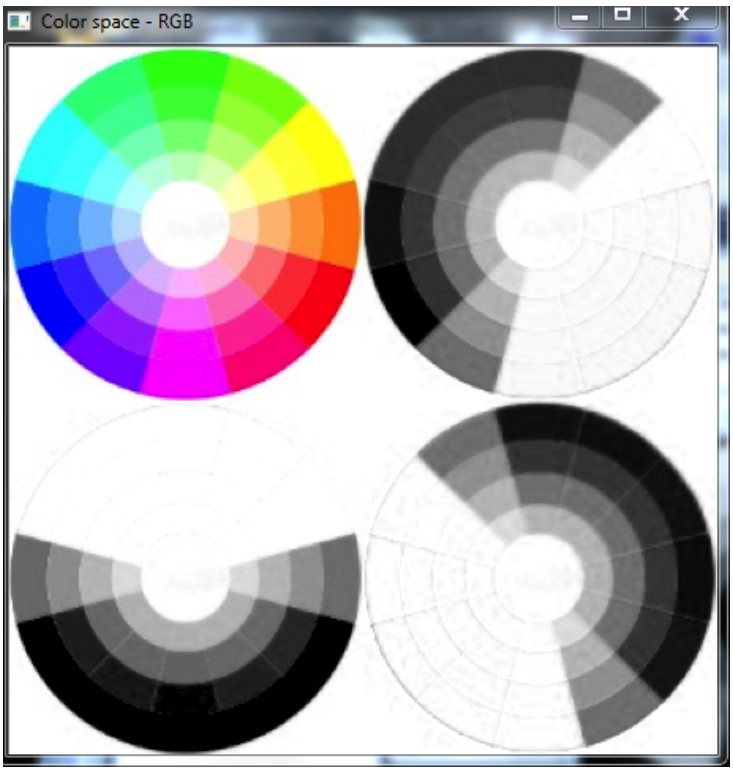

Figure 7: RGB image

Even when using OpenCV, there's no direct method to obtain the CMYK components, so that it's necessary to code (shown below) a conversion with the equations shown in the previous section, in the CMYK subsection.

```
cv::Mat mat = cvLoadImage("colorFile.png");//1);//Mat(image, false);
cv::vector<cv::Mat> cmyk;
for (int i = 0; i < 4; i++) {
     cmyk.push_back(cv::Mat(mat.size(), CV_32F));
}
for (int i = 0; i < mat.rows; ++i){
     for (int j = 0; j < mat.cols; ++j){
          cv::Vec3b p = mat.at<cv::Vec3b>(i,j);
          float r = p[2]/255.;
          float g = p[1]/255.;
```

```
            float b = p[0]/255.;
            float k = (1 - max(max(r,g),b));
            cmyk[0].at<float>(i,j) = (1 - r - k) / (1 - k);  //C
            cmyk[1].at<float>(i,j) = (1 - g - k) / (1 - k);  //M
            cmyk[2].at<float>(i,j) = (1 - b - k) / (1 - k);  //Y
            cmyk[3].at<float>(i,j) = k;  //K
        }
}
```

The CMYK analyzes results in **Fig. 8**, where it's shown, from left to right and top to bottom, the C, M, Y and K components, respectively.

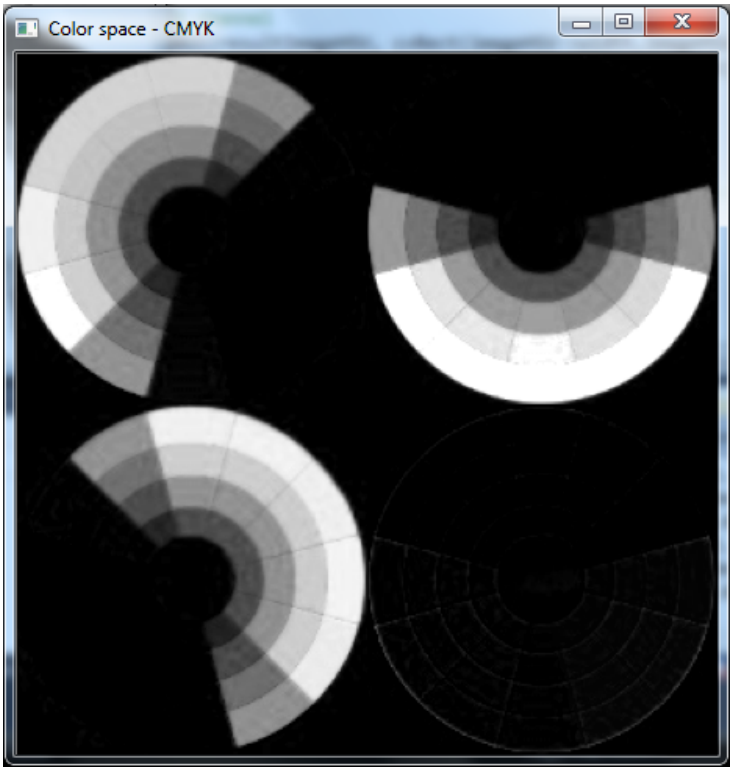

Figure 8: CMYK image

The HSV analyzes results in **Fig. 9**, where it's shown, from left to right and top to bottom, the original image in the HSV color space, and its components H, S and V. In order to obtain those results, it's necessary to convert the color space of the image from RGB to HSV, using the *cvCvtColor* function with *CV_BGR2HSV* as parameter, and then use the same function used when analyzing the RGB color space (*cvSplit*).

```
cvCvtColor(image, imageHSV, CV_BGR2HSV);
cvSplit(imageHSV, imageH, imageS, imageV, NULL);
```

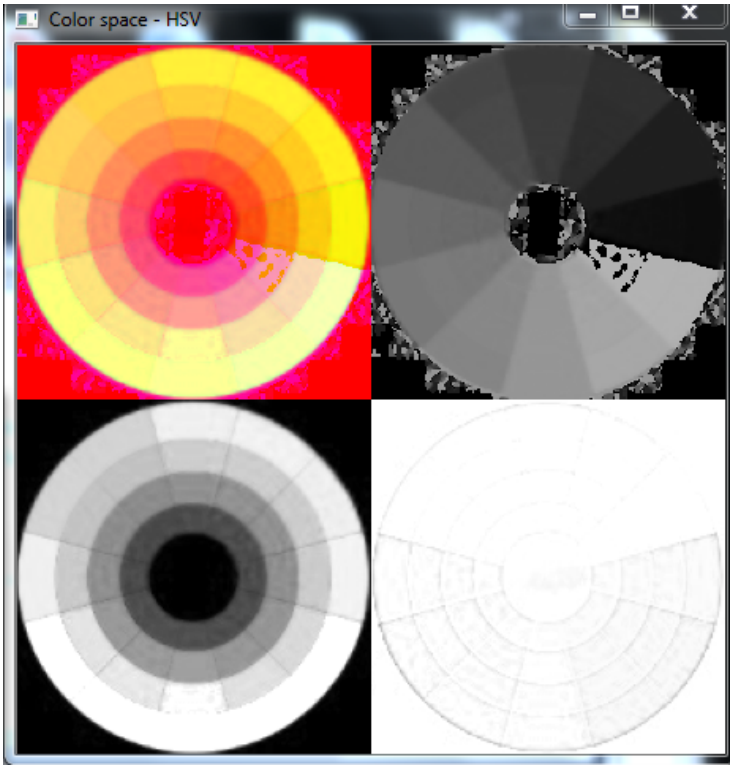

Figure 9: HSV image

2.3 Eye Detection

In this part of the assignment, we'll have to detect the eye given the image in **Fig. 10**.

Figure 10: Original image

The way this was achieved was by first detecting the darkest colors, shown in **Fig. 11**, since the eye should be the darkest part of the image:

```
Mat eyeInRange;
Scalar minColor = Scalar(0,0,0);
Scalar maxColor = Scalar(25,25,25);
inRange(eye, minColor, maxColor, eyeInRange);
```

Figure 11: Color detection

Then, calculating the contours from the image obtained previously with the color detection technique:

```
vector<vector<cv::Point>> contours;
findContours(eyeInRange.clone(), contours, CV_RETR_EXTERNAL,
    CV_CHAIN_APPROX_NONE);
```

As can be seen in **Fig. 11**, the eye is represented by the second largest contour, obtained with the code below and shown in **Fig. 12**:

```
int largestContour = 0;
int largestIndex = 0;
int secondLargestContour = 0;
int secondLargestIndex = 0;

for (int i = 0; i < contours.size(); i++) {
    if (contours[i].size() > largestContour) {
        secondLargestContour = largestContour;
        secondLargestIndex = largestIndex;
        largestContour = contours[i].size();
        largestIndex = i;
    } else if (contours[i].size() > secondLargestContour) {
        secondLargestContour = contours[i].size();
        secondLargestIndex = i;
    }
}

Scalar color = Scalar(255,255,255);
drawContours(aim, contours, secondLargestIndex, color, CV_FILLED, 8);
```

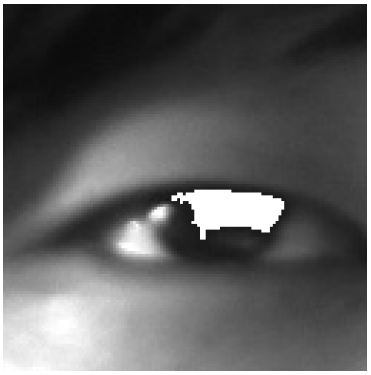

Figure 12: Second largest contour in image

However, we don't want to see the contour, but an aim in the eye, **Fig. 13**:

```
Point2f center = contours.size();
float radius = contours.size();
minEnclosingCircle((Mat) contours[secondLargestIndex], center, radius);
circle(eyeDetected, center, (int) radius, color, 2);
cv::line(eyeDetected, cv::Point(center.x, center.y-(int) radius),
    cv::Point(center.x, center.y+(int) radius), color, 2);
cv::line(eyeDetected, cv::Point(center.x-(int) radius, center.y),
    cv::Point(center.x+(int) radius, center.y), color, 2);
```

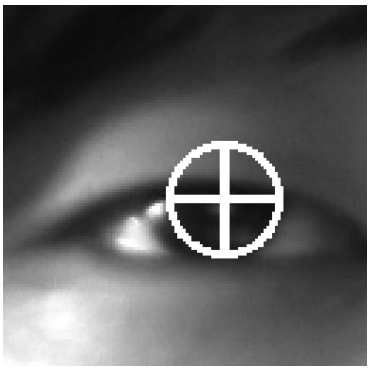

Figure 13: Eye detection