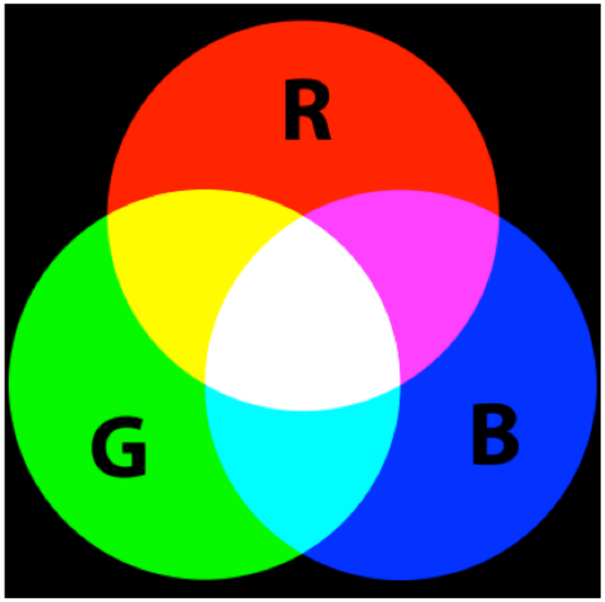
Color Space
Assignment 1

Gwenaelle Cunha Sergio
ABR Lab – BEP
KNU 2014.1

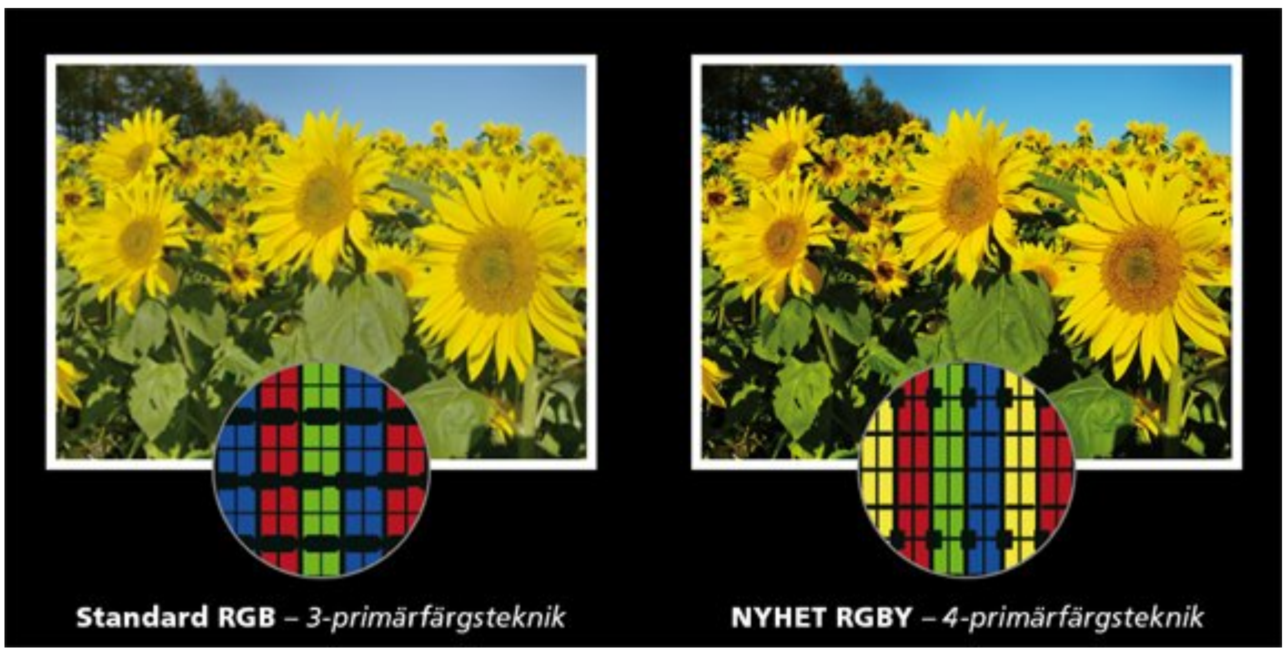
RGB

- Most used
- Three primary colors
- Addictive color space
- Range: 0 – 255



RGBY

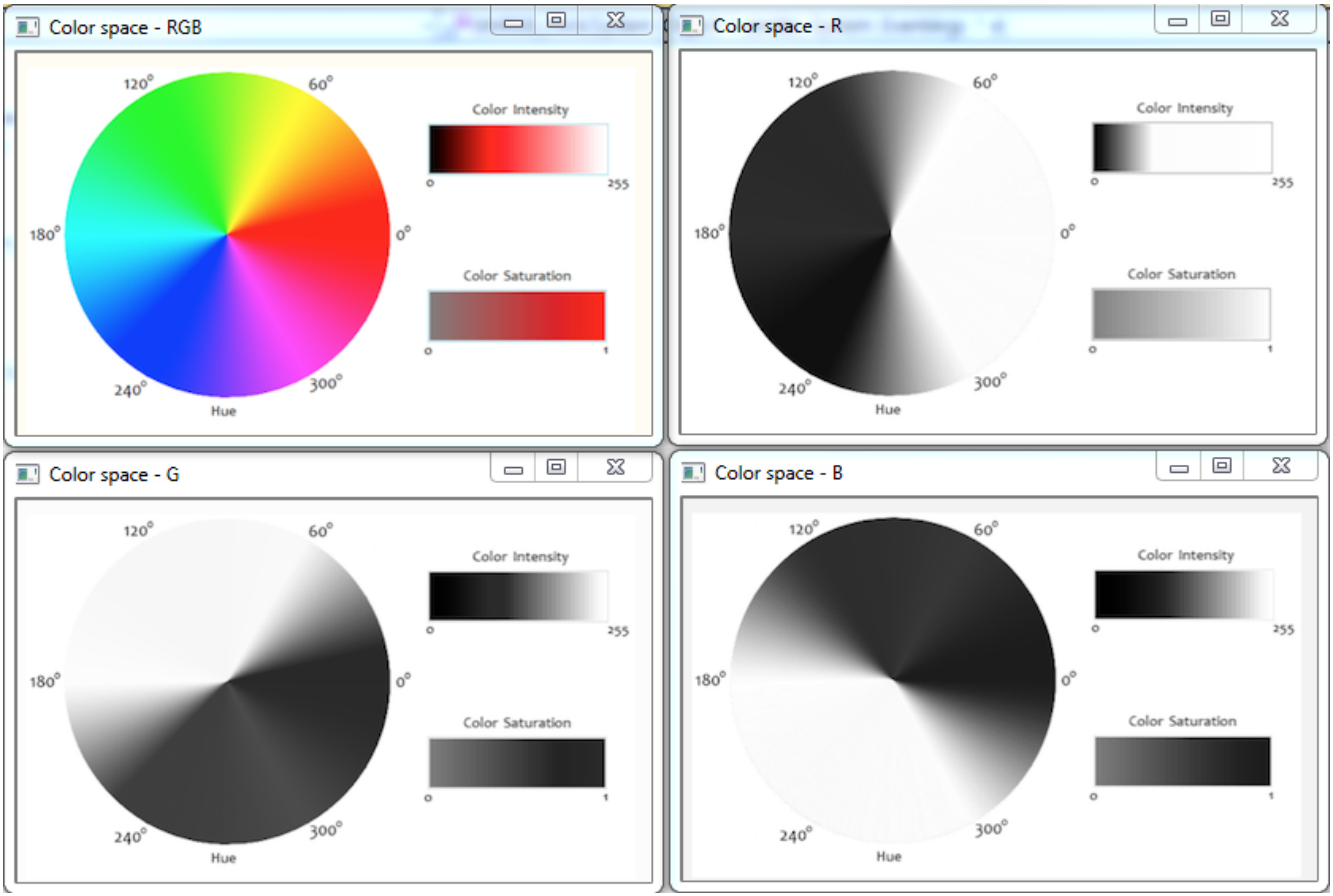
- Real RGB
- Adds yellow pixel: more colors available



RGB - Code

```
cv::Mat mat = cvLoadImage("colorFile2.png");
cv::vector<cv::Mat> rgb;
for (int i = 0; i < 4; i++) {
  rgb.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);
    rgb[0].at<float>(i,j) = p[2]/255.;        //R
    rgb[1].at<float>(i,j) = p[1]/255.;        //G
    rgb[2].at<float>(i,j) = p[0]/255.;        //B
  }
}
```

RGB - Result



Color space - RGB
Color space - R
Color space - G
Color space - B
120°
60°
180°
0°
240°
300°
Hue
Color Intensity
0
255
Color Saturation
0
1

CMYK

- Used in printers
- Subtractive color space
- RGB to CMYK

$$K = (1 - max(r, g, b))$$

$$C = \frac{1 - r - k}{1 - k}$$

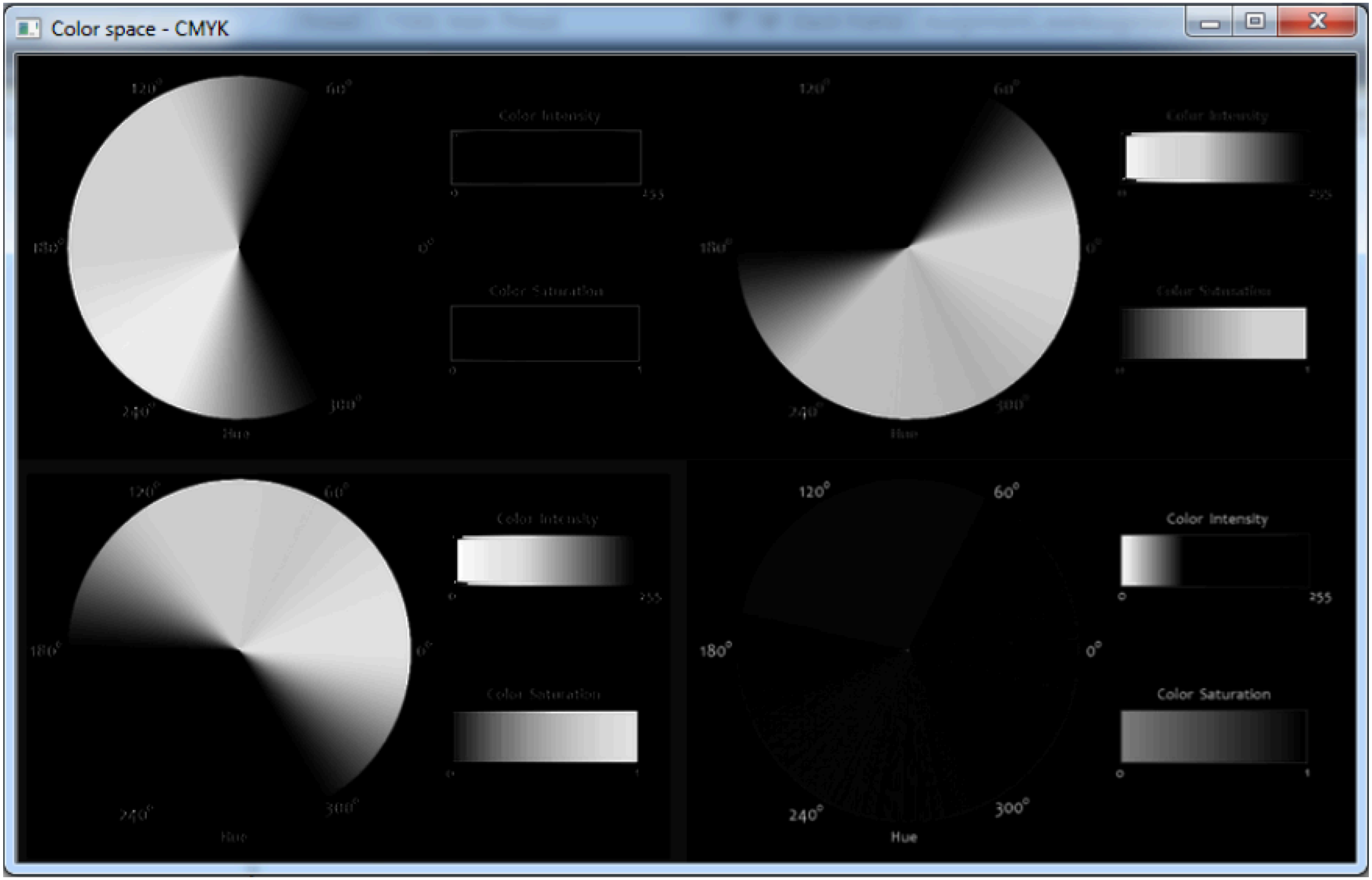
$$M = \frac{1 - g - k}{1 - k}$$

$$Y = \frac{1 - b - k}{1 - k}$$

CMYK - Code

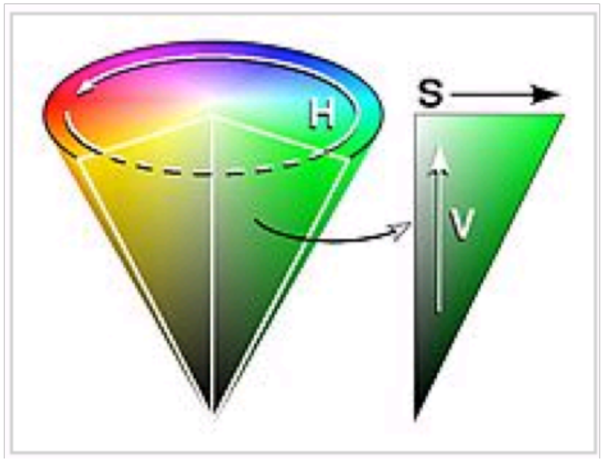
```
cv::Mat mat = cvLoadImage("colorFile.png");
cv::vector<cv::Mat> cmyk;
for (int i = 0; i < 4; i++) {
    cmyk.push_back(cv::Mat(mat.size(), CV_32F));
}
for (int i = 0; i < mat.rows; ++i){
    for (int j = 0; j < mat.cols; ++j){
        cv::Vec3b p = mat.at<cv::Vec3b>(i,j);
        float r = p[2]/255.;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

CMYK - Result


Color space - CMYK
120°
60°
180°
0°
240°
300°
Hue
Color Intensity
0
255
Color Saturation
0
1

HSV

- One of the most common "cylindrical-coordinate represen-tations of points in an RGB color model".
- More intuitive and perceptually relevant for humans than the traditional representation.
- Conversion from RGB



$$
\begin{aligned}
M &= max(R,G,B) \\
m &= min(R,G,B) \\
C &= M - m
\end{aligned}
$$

$$
H' = \begin{cases} 0 & C = 0 \\ \frac{G-B}{C} \bmod 6 & M = R \\ \frac{B-R}{C} + 2 & M = G \\ \frac{R-G}{C} + 4 & M = B \end{cases}
$$

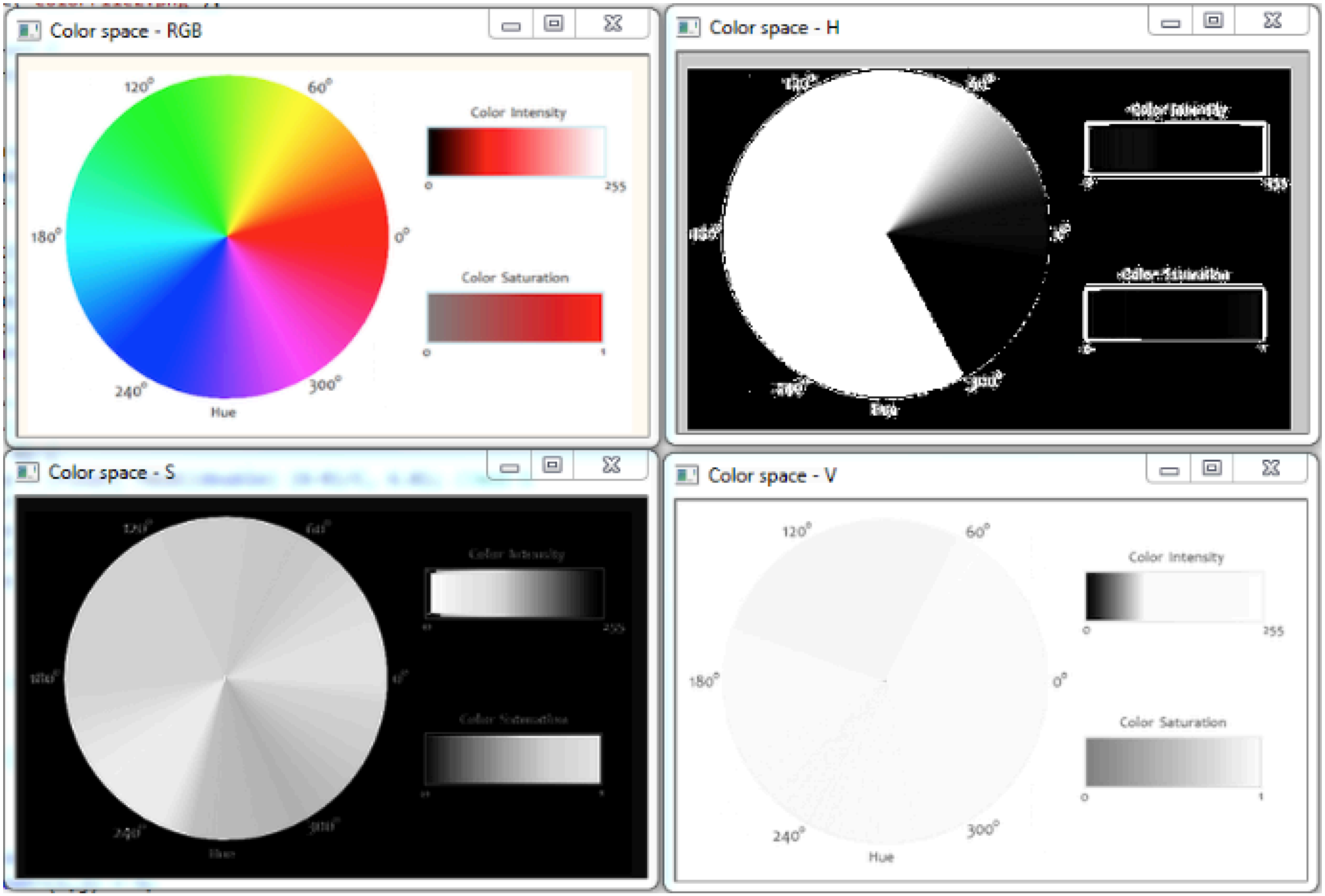
$$
\begin{aligned}
H &= 60^o H' \\
S &= \begin{cases} 0 & C = 0 \\ \frac{C}{V} & \text{otherwise} \end{cases} \\
V &= M
\end{aligned}
$$

HSV - Code

For every index in the image, do:

```
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 M = max(max(R,G),B); float m = min(min(R,G),B); float C = M – m; float Htemp = 0;
if (C != 0) {
    if (M == R) {
        Htemp = (float) fmod((double) (G-B)/C, 6.0); //mod 6
    } else if (M == G){
        Htemp = (B-R)/C + 2;
    } else if (M == B){
        Htemp = (R-G)/C + 4;
    }
}
float H = (3.14/3)*Htemp; //60 degrees
float S = 0;
if (C != 0) S = C/V;
float V = M;
hsv[0].at<float>(i,j) = H; hsv[1].at<float>(i,j) = S; hsv[2].at<float>(i,j) = V;
```

HSV - Result



Color space - RGB
Color space - H
Color space - S
Color space - V
120°
60°
180°
0°
240°
300°
Hue
Color Intensity
0
255
Color Saturation
0
1

HSI vs HSV

- HSV: presents color choice
- HSI: facilitates separation of shapes in an image
- Hue is the same in both
- Saturation

$$S_{HSV} = \begin{cases} 0 & C = 0 \\ \frac{C}{V} & \text{otherwise} \end{cases} \qquad S_{HSI} = \begin{cases} 0 & C = 0 \\ 1 - \frac{m}{I} & \text{otherwise} \end{cases}$$

- Intensity: 1/3 (R+G+B)

YIQ

- Used by the NTSC color TV system
- ``Takes advantage of human color-response characteristics’’
- Encoding schemes based on the luma-chroma color theory
- Y = luma (used by black and white television receivers)
- I = in-phase and Q = quadrature -> chrominance

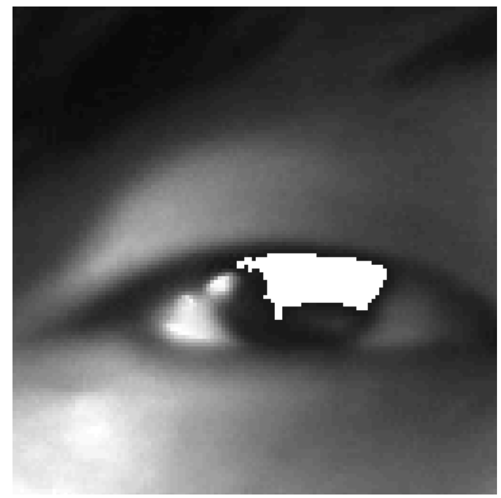
YIQ vs YCrCb

- Basis for PAL and NTSC color TV system
- Based on how the human eye works
- YIQ: first used when color was introduced to television (analog)
- YCrCb: digital transmissions, but also supports analog TV

Detect Eye

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

Detect Eye



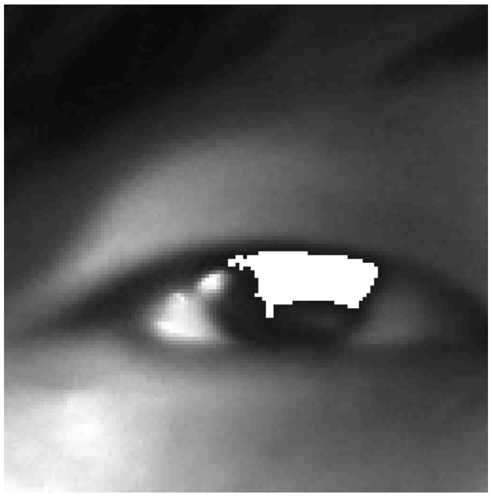
- Find contours:

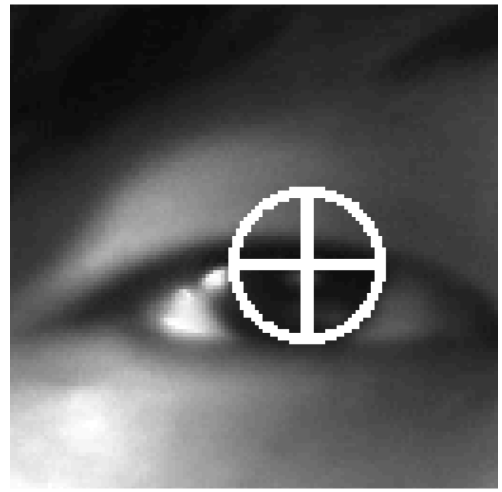
```
findContours(eyeInRange.clone(), contours, CV_RETR_EXTERNAL,
    CV_CHAIN_APPROX_NONE);
```

- Find second largest contour, and:

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

Detect Eye





```
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);
```