Chapter 5

Colours

5.1 Introduction

Images represented in greyscale only contain information about the brightness and the shape of objects. Colours present additional cues about the objects. Humans can distinguish colours very well, and therefore it is useful for some applications that the image processing is based in the properties of a certain colour space.

The use of colours can be limited by the application and by the acquisition system. In terms of application, suppose that one wants to find cars in a road scene. Colours might be of very limited help in this case, as colours do not uniquely identify cars. In terms of the acquisition of images, there are cameras that are notoriously selective in colour. Cameras can make certain colours disappear or change under different lighting conditions. The same images might get the "correct" colours if seen be human eyes. Human beings can perceive a very narrow range of light frequencies that stretches from the red to the violet. That is why we often refer to wave lengths that are longer than red as infrared and to wavelengths that are shorter than violet as ultraviolet (remember that the wavelength is inversely proportional to the frequency). Figure 5.1 shows the sensitivity for the three components based on the frequency of the light. Notice that green is the most spread, but less sensitive.

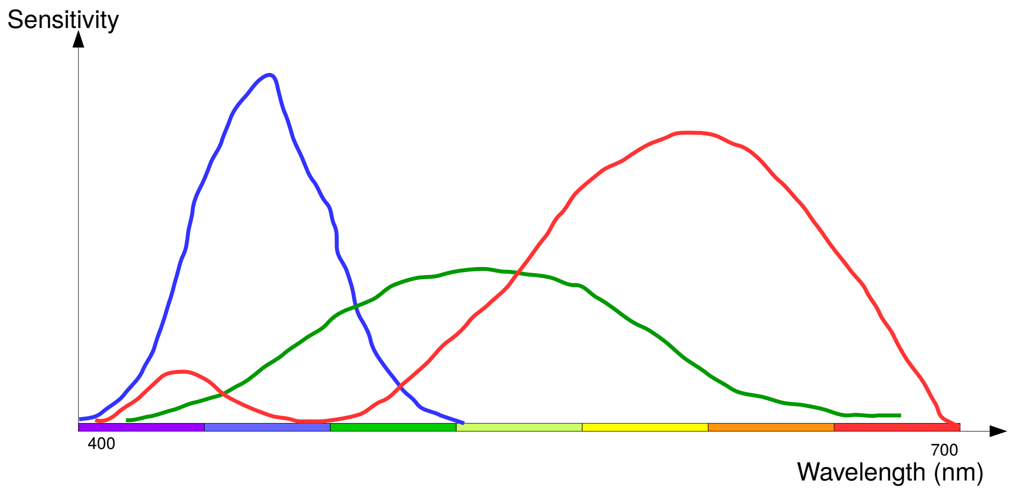

Figure 5.1: Spectral sensitivity for R, G and B.

Cameras are sometimes sensitive to a different range of frequencies, in such a way that they may have capabilities that human eyes lack. For example, many web cameras are capable of acquiring images in the infrared spectrum. Due to restrictions on the electronic sensors, CCD and CMOS cameras use the so-called Bayer pattern (figure 5.2), where there are two green sensors for each red and each blue sensor.

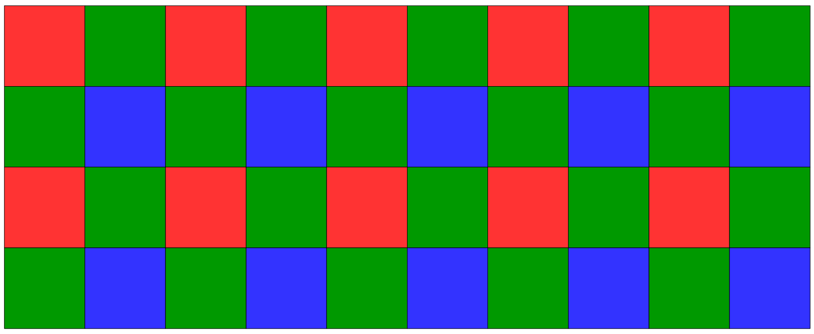

Figure 5.2: The Bayer pattern.

A mixture of different frequencies of light can be perceived as a certain colour by humans. For example, a mixture of green and red may yield yellow. However, the perceived yellow colour might be different than the pure yellow obtained from a single frequency. Another good example is the colour pink. There is no pink as a pure frequency in the visible spectrum, what we perceive as the pink colour is a combination of frequencies of visible light.

The human eye has different sensors (cones), each responsible for certain frequency ranges. We can perceive colours by interpreting the mixed signals of these three sensors. And that is why we can use three basic colours to make up (almost) any other colours in the visible spectrum. In order to create colours for a computer monitor, it suffices to have RED, GREEN and BLUE. By combining different intensities of each of these three colours, one can get the desired resulting colour. This is an *additive* colour system.

For printing, a different system was devised. Because printers print on a white background, the ink has to cover the white colour. By adding different amounts of ink of each of three colours, MAGENTA, YELLOW and CYAN, one can get (almost) any colour in paper. This is an example of a *subtractive* colour system. Figure 5.3 shows both additive and subtractive systems side by side. One interesting fact is that by combining the maximum strength of the three colours we should get the black colour. In practise the mixture results in a dark brown colour. For this reason, most printing systems have a black ink cartridge

Follow some keyword definitions used in this chapter:

- Luminance: gives the amount of energy perceived by the eye (lumen).
- Radiance: the amount of energy given out by the object (watts).
- Brightness: sensation of the strength or intensity of a colour.
- Hue: dominance of the wavelength of a particular colour.
- Saturation: purity of the wavelength or degree of mixing with white. E.g., pink (red+white) is less saturated than red.

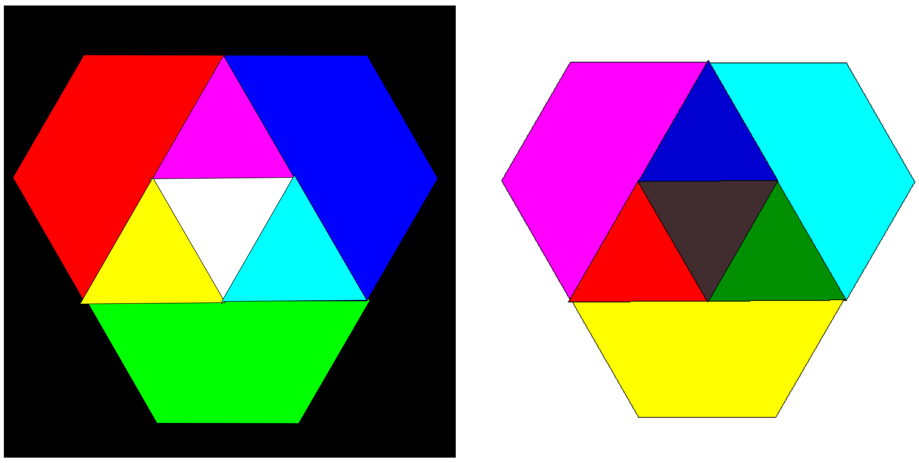

Figure 5.3: a) RGB space (additive colour). b) YMC space (subtractive colour) .

5.2 Colour Spaces

Colour spaces can be defined as a representation of a certain colour model. There are standard colour spaces used nowadays, e.g.,CIE XYZ, CIELUV, CIEUVW, CIELAB, CMYK (cyan, magenta, yellow, black), HSL, HSV, RGB, RGBA (with an alpha channel to indicate transparency), YIQ (applied to NTSC), YUV, YPbPr, YCbCr, xvYCC and the list goes on. Only RGB and HSV, the most common colour spaces used in computer vision applications, are described next.

5.2.1 RGB or GBR

The RGB (red/green/blue) space is represented by a cube (figure 5.4). The three colours are completely independent from each other. The ranges for the three primary colours vary in practise and are dependent on the data structure used. For example, a system that uses one byte per colour has 256 states for each colour. That amounts to 24 bits and therefore a maximum of 2^{24} colours can be represented. Different grades of grey are located in the diagonal of the cube (see figure 5.4). Note that in the RGB space, where each colour has 256 states, the greyscale line is shorter than 256. When converting RGB to greyscale images, the greyscale values have to be rounded up. Not only some information is lost regarding the colour, but also in relation to the brightness of each pixel converted to greyscale.

5.2.2 HSI (or HSL) and HSV (or HSB)

The HSI space is composed by H (hue), S (saturation) and I (intensity). A variation of this system is called HSV [1] (V stands for value). This space was originally represented by a cylinder and the colours were defined within a hex-cone (figure 5.5. It is a common way of expressing the H ranges in angles, e.g., from 0^o to 360^o. The saturation ranges from 0 to 1 (or in some systems from 0% to 100%). Similar ranges are used for the brightness values.

The main advantage of using these colour spaces is that they can isolate the value (or intensity) from the colour. Therefore, in the case of colour classification it would transform a 3 dimensional problem into a 2 dimensional one. In fact, for certain colours the H component (hue) may suffice to get a good classification.

[1]aka HSB, B for brightness. The difference between HSI and HSV is that the V of a pure colour is equal to white's brightness, while the intensity of pure color is equal to the intensity of a medium gray

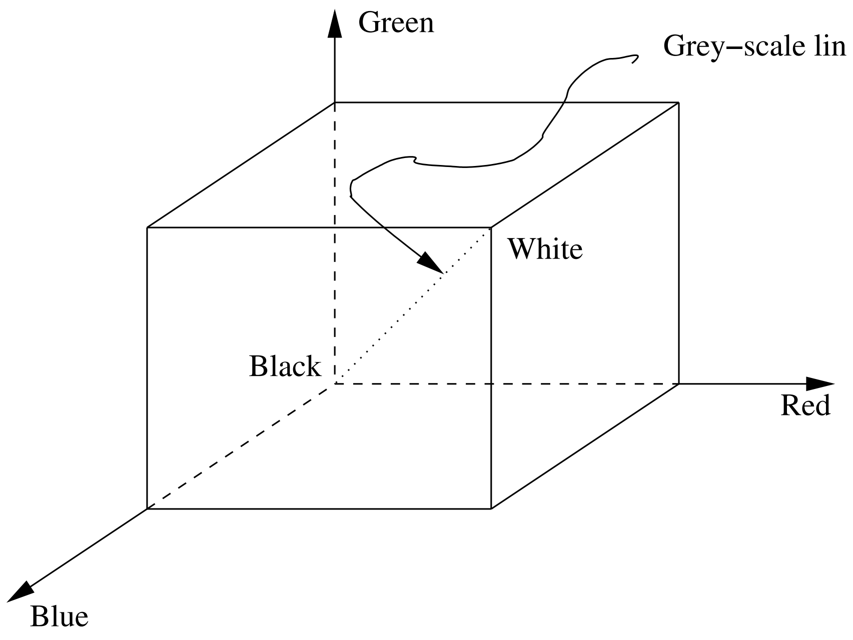

Figure 5.4: RGB colour space.

5.2.3 Converting RGB to HSV and vice-versa

Implementations of colour space try to optimise storage space by changing the ranges of H, S and V to 0 255. This affects the conversion to and from RGB space. One has to carefully analyse the data structure and adapt the conversion algorithms appropriately in order to convert colour spaces correctly.

Follow an example of conversion from RGB to HSV and vice-versa. Refer to [1] for alternative equations. It is assumed that the values for R,G and B are normalised and that the HSV values are within the following ranges:

- H - between 0 and 360
- S - between ≥0 and 1 (if S is 0 then H is undefined)
- V - between 0 and 1

Listings 5.1 and 5.2 show the transformations.

Listing 5.1: Converting RGB to HSV

```
void ConvertRGBtoHSV(int R, int G, int B){
    min = MIN(R,G,B);
    max = MAX(R,G,B);
    V = max;
    if(max != 0) S=(max-min)/max;
    else S=0;
    if(R==max) H = (G-B)/(max-min);
    else{
        if(G==max) H=2+(B-R)/(max-min);
        else H=4+(R-G)/(max-min);
```

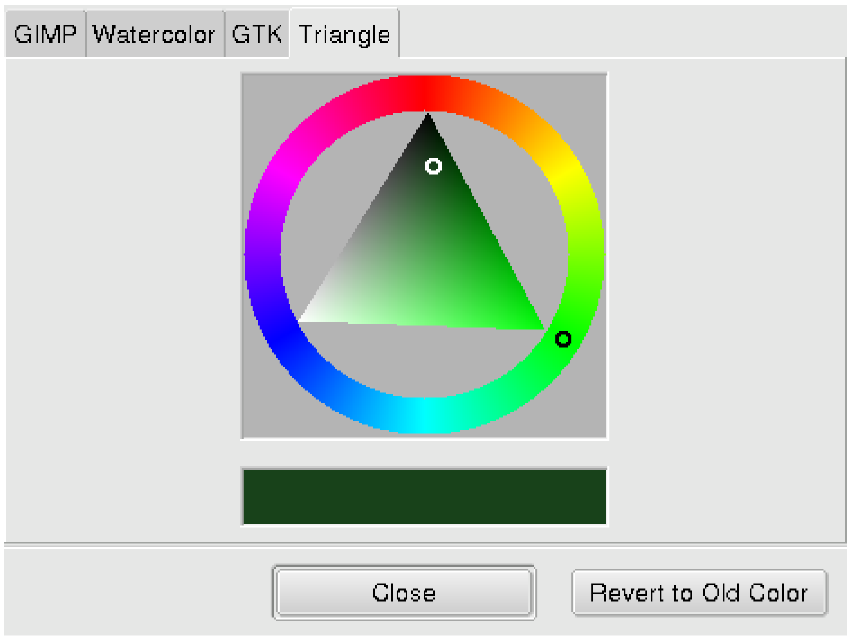

Figure 5.5: HSI space based on triangle.

```
    }
    H=H*60;
}
```

Listing 5.2: Converting RGB to HSV

```
//H between 0 and 360, in OpenCV H = H * 2;
void ConvertHSVtoRGB(int H, int S, int V){
   int C = V * S;
   int X = C * (1 - abs(((H/60.0)%2-1)));
   int M = V -C;
   if ( H >=0 && H < 60 )      {R=C; G=X; B=0;}
   if ( H >=60 && H < 120 )    {R=X; G=C; B=0;}
   if ( H >=120 && H < 180 )   {R=0; G=C; B=X;}
   if ( H >=180 && H < 240 )   {R=0; G=X; B=C;}
   if ( H >=240 && H < 300 )   {R=X; G=0; B=C;}
   if ( H >=300 && H < 360 )   {R=C; G=0; B=X;}
   R = (R+M)*255;
   G = (G+M)*255;
   B = (B+M)*255;
}
```

5.2.4 YUV and YUQ

These are used in PAL or NTSC TV systems. They are very similar to HSV. Humans are more sensitive to changes in the luminance (Y component) than on the chrominance (U and V components). If the values related to the chrominance are rounded up or quantised, the human eye cannot perceive any loss of information. While the Y component uses the full resolution, only one value of U and V are used to describe a small set of pixels. The most common YUV is the so-called YUV411. That is equivalent to sending the whole resolution for the Y while sending only half the resolution for the U and V components. For every 4 neighbour pixels the chrominance is the same (see figure 5.6). This approach is often used by web cameras because it minimises the amount of data transmitted.

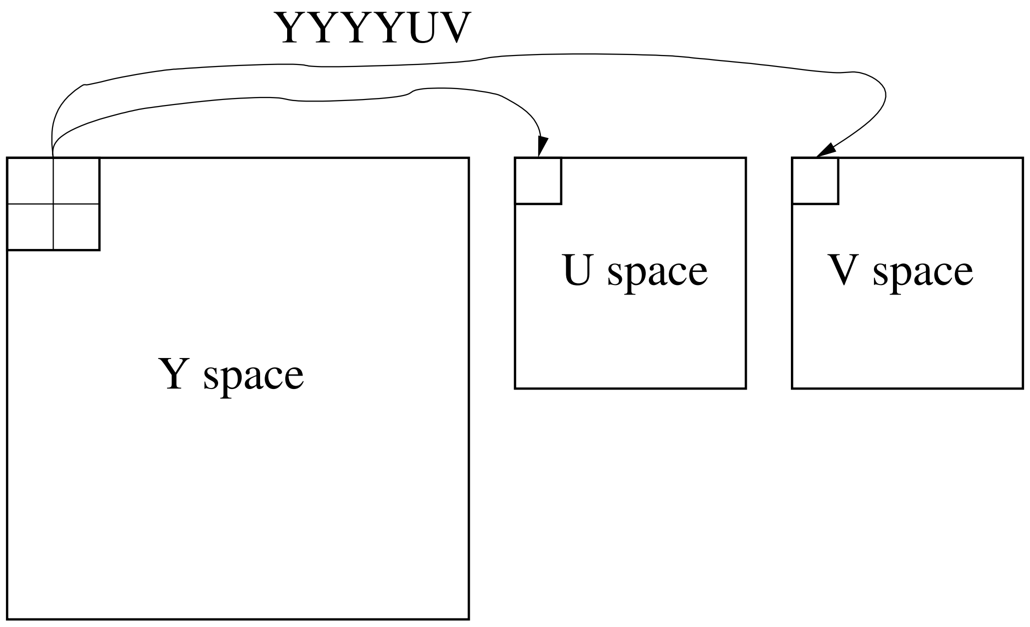

Figure 5.6: YUV approach.

5.2.5 Converting YUV to RGB and vice-versa

For YUV to RGB conversion, one can find slightly different approaches in the literature. Again, it is important to keep in mind the limitations of the data structures to carry out the conversion properly. Also, some knowledge about the acquisition system might be important because the implementation does not necessarily follow any "logical" or standard practise. The implementation often results from some circumstantial hardware or software limitation (e.g., certain cameras might use only part of the 256 values for U or V, say U could only accept values between 10 to 248).

To convert YUV to RGB using a web camera, the following code works (listing 5.3:

Listing 5.3: Converting YUV to RGB

```
void YUVtoRGB(int Y, int U, int V){
  R = (int)((Y-16)*1.164+1.596*(V-128));
  G = (int)((Y-16)*1.164-0.392*(U-128)-0.813*(V-128));
  B = (int)((Y-16)*1.164+2.017*(U-128));
}
```

To convert from RGB to YUV (listing 5.4):

Listing 5.4: Converting RGB to YUV

```
void RGBtoYUV(int R, int G, int B){
  Y = (int) (0.299 * R + 0.587 * G + 0.114 *B);
  U = (int) (0.492 * (B - Y));
  V = (int) (0.877 * (R - Y));
}
```

5.2.6 Converting Colours in OpenCV

OpenCV has many conversion possibilities. The most common you may want to use is to convert from RGB to HSV. By default, the IplImage structure will work with RGB space. Most cameras will also work with RGB. For segmentation it is useful to convert the colours to a different space. The following listing is an example of simple binarization based on HSV colour. The thresholds (marker_upS, marker_loS etc) define an oblong region in the HSV space, and if the illumination thresholds are kept to their limits, then only H and S are being used for classification.

Listing 5.5: Function to convert and classify colours

```
void Classify_Colour(IplImage* image, IplImage *imageHSV){
  cvtColor(image,imageHSV, CV_RGB2HSV);
  for (int x=0;x<imageHSV->width;x++){
    for (int y=0;y<imageHSV->height;y++){
      if( pixelR(imageHSV,x,y) < marker_loV || pixelR(imageHSV,x,y
          ) > marker_upV ||
        pixelG(imageHSV,x,y) < marker_loS || pixelG(imageHSV,x,y)
            > marker_upS ||
        pixelB(imageHSV,x,y) < marker_loH || pixelB(imageHSV,x,y)
            > marker_upH   ){
          pixelR(image,x,y)=0;
          pixelG(image,x,y)=0;
          pixelB(image,x,y)=0;
      }
      else {
        pixelR(image,x,y)=255;
        pixelG(image,x,y)=255;
        pixelB(image,x,y)=255;
      }
    }
  }
}
```

In cases where colours are an important property of an object in the image they can be used to separate (or segment) regions of interest. Examples of application include skin colour segmentation, fruit sorting (quality of the fruit), quality assurance etc.

Segmentation can use different colour spaces. Researchers noticed that for certain groups of colours it is possible to use simple thresholds, while other groups of colours require a more complex classification (non-cubic regions in the colour space). One good example is skin colour, used in many applications such as gesture recognition and games. It is easier to segment skin colour in HSI space than it is in RBG space.

5.3 High Dynamic Range (HDR)

Dynamic Range usually refers to the range (interval) of the luminance of the image being processed. The luminance range can be defined simplistically as the difference between the brightest and the darkest pixels of the image. The High dynamic range is a somewhat arbitrary name to indicate that systems with HDR are able to deal with a higher range than the usual devices, but there is no indication of how better it really is.

Typically there are three arbitrary ranges, namely HDR (High Dynamic Range), SDR (Standard Dynamic Range) and LDR (Low Dynamic Range). These three acronyms give an indication of the quality or capacity of the device involved. Most cameras with cheap CCDs are LDR, as are monitors. Photographic paper tends to be SDR or HDR depending on the quality.

Professional cameras are sometimes referred to as HDR capable, but they still use LDR CCDs. The HDR is achieved by taking several frames in a short time interval using different exposures and then combining these frames into a single HDR image. Unfortunately, if the camera moves during the frames acquisition or if objects in the image move too fast, the resulting HDR image can be blurred.

To give an idea of how luminance varies in the real world, table 5.3 shows a few scenarios and their luminance.

Table 5.1: Luminance in $candela/m^2$

Scenario	Luminance
Average star	0.001
Moon light	0.1
Indoors during the day	50
Outdoors in a sunny day	100000

The next table (5.3) shows the range of several devices compared to the human eye (the values are estimates only). Notice that the human eye presents the best dynamic range of them all. There are mammals and birds that have an even better dynamic range than humans.

Table 5.2: Devices versus human eye

Device	Range
Human eyes	1,000,000 : 1
Film (old school sense)	2000 : 1
Monitor and TVs	500 : 1
Quality digital camera	300 : 1
Printing (good quality)	200 : 1
Cheap digital camera	100 : 1

The tables above explain why we get the annoying result when taking pictures where a high dynamic range is needed. Either we get the bright part of the image accurately, or the dark part of the image. Even though we can see both clearly, the devices cannot produce a high dynamic range image using one frame.

5.3.1 Image Acquisition

The simplest approach to obtain HDR images is to use multiple frames with different exposures (sometimes called AEB (automatic exposure bracketing)). The problem with that is to get the geometric and colour calibrations. How to combine accurately pixels that belong to different frames? Also, the images are never completely aligned, which causes the resulting HDR image to blur.

The other approaches involve the use of multiple CCDs. Nowadays one can buy industrial cameras with dual CCDs. These commercial cameras have a prism to create two separate images for each CCD. Different exposures give the range. In this approach the geometric calibration problem is minimised, and it can deal with fast moving objects or moving camera.

More CCDs can be used, but this makes it more difficult to do the geometric calibration. The calibration has to assume that the objects are restricted to certain distances from the camera.

5.3.2 Combining Several LDR into one HDR image

Once the LDR images are acquired they need to be processed into one coherent HDR image. Assuming that they are geometrically aligned, we are left with the problem of deciding about the colour and value for each HDR pixel.

In the literature there is a number of different methods. These methods are classified into three main branches: Parametric (pixel based), Parametric (histogram based) and Non-parametric (pixel based). Important work in area are from Mitsunaga and Nayar [2], Grossberg and Nayar [3] and Debevec and Malik [4].

5.3.3 Tone Mapping

After the HDR image is finally created, it can be used for further processing. Unfortunately it cannot be shown in any existing device (see table 5.3 again to understand why this is the case!). In practise, the HDR image is transformed into an LDR image again. However, the resulting LDR image has to present all the details on all parts of the image clearly. It is a process similar to the contrast stretching seen before, but this is applied in a more complex manner.

5.3.4 Colour Calibration

CCDs are notoriously inaccurate with colours. The fabrication method and the materials used make each individual sensor to respond differently to light, so even two cameras of the same model will not behave the same. Colour calibration is a common approach to resolve this issue. Colour calibration should standardise the behaviour of the camera to light, and make it work coherently with darker and brighter pixels. For that purpose, a colour chart is often used. In figure 5.7 it shows an experimental set up with three cameras and a colour chart (the colour chart is an *X-Rite 24 patch ColorChecker*).

Instead of using colour calibration alone, an alternative is to use a limited number of colours and use classifiers to deal with the values from the 3 cameras. The reader should be convinced about the impossibility of producing a lookup table in memory for each combination of RGB values from the three cameras. The number of entries in such a table would be $2^{(3*24)} = 2^{72} = 4722366482869645213696$. Rather than a look-up table, a classifier can be used instead.

Now remains the question of what colour space should be used for the classifier. Some preliminary experiments were published by our research group (Barczak et al. [5]). The results show that using a colour space that separates the illumination from the colour gives better

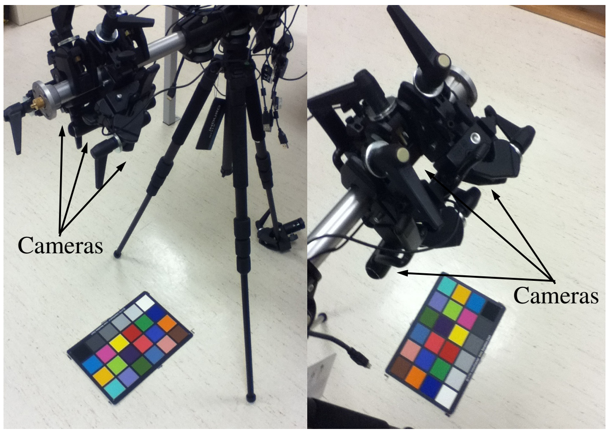

Figure 5.7: The HDR approach using 3 cameras [5].

results. Figure 5.8 shows an example with eight colour classifiers, and figure 5.9 shows the percentage of correct classifications.

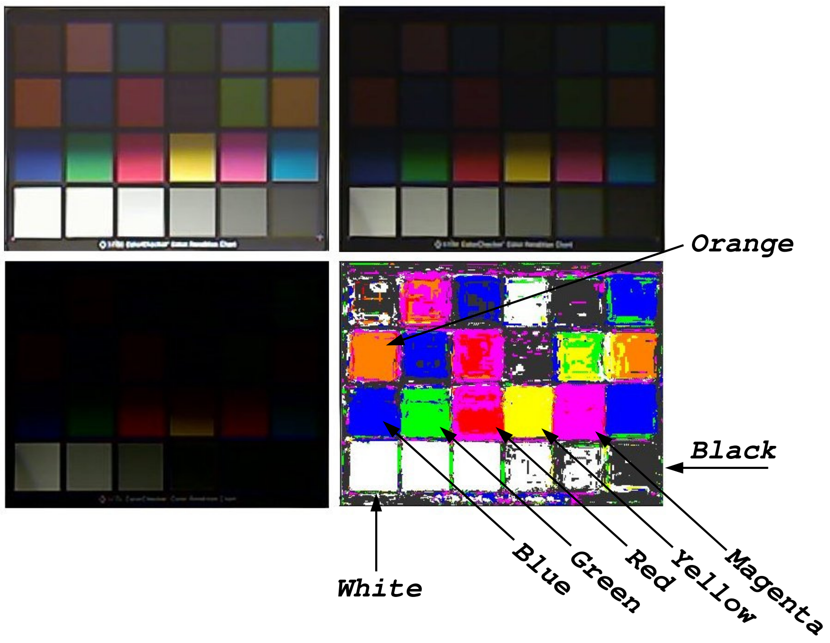

Figure 5.8: The HDR classifier using 3 cameras. Note that the patches with strong illuminations difference still get good classifications [5].

5.3.5 Colour Classification

This is a sub-problem of tone mapping. Instead of getting an equivalent colour in LDR from the HDR image, all the user want is to classify a patch of colours into a small number of classes. This is a useful method for applications that require segmentation of regions based on colours, where the illumination conditions cannot be guaranteed.

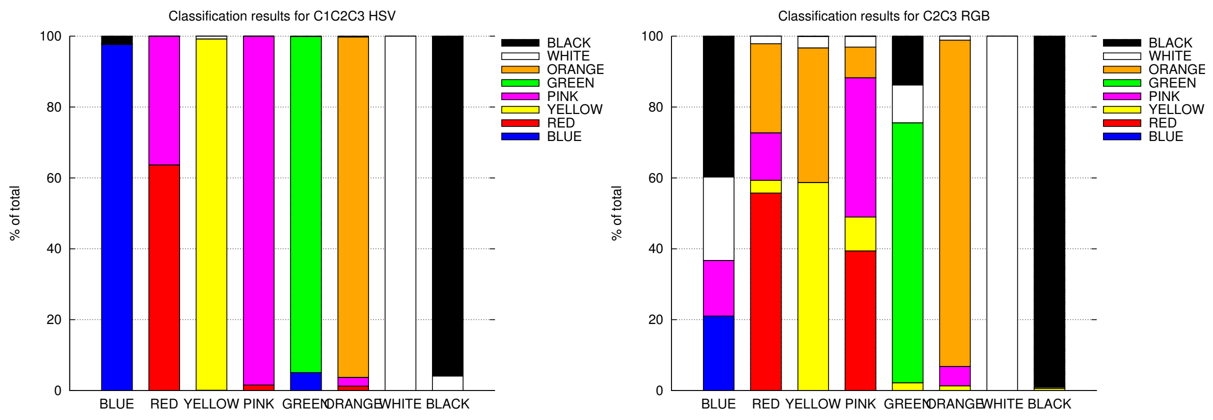

Figure 5.9: Histograms for the classification results. a) HSV space. b) RGB space. [5]

5.4 Fuzzy Colour Contrast Fusion: an example of colour constancy algorithm

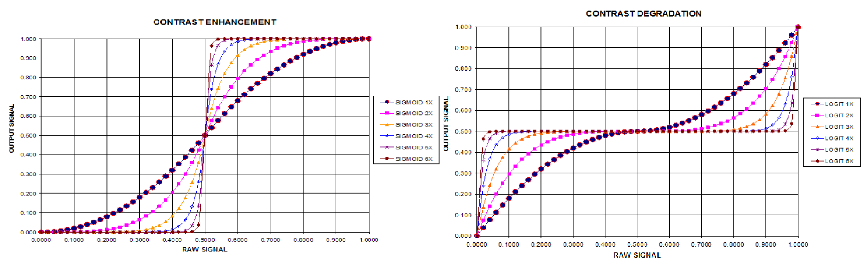

Figure 5.10: On the left is the Contrast Enhance Operator, while on the right is the Contrast Degrade Operator.

In many cases it is important that the colours depicting an object have be adaptively corrected based on the relative illumination conditions of the environment they are exposed to. FCCF adaptively performs colour correction by either colour contrast enhancing or degrading the colour channels at different levels of intensity, prior to classifying the sensed colour tristimulus. For each target colour at hand (e.g. pink, orange), the RGB components will receive a unique set of fuzzy colour contrast operations.

This section is a short version of a paper presented at a conference, refer to ([6]) for the complete description of the algorithms and the experimental results.

Enhance or degrade operations are implemented via non-linear functions [7]. Figure 5.10 depicts the curve exhibiting the contrast enhance operator applied in different levels (1x, 2x, 3x, etc). The input signal can be any of the normalized RGB components within the range $[0, 1]$. In turn, the function amplifies input values greater than 0.5; and otherwise, attenuates it [8].

On the other hand, the contrast degrade operator performs the opposite fashion [9], as depicted in the curve in Fig. 5.10. It amplifies all signals less than 0.5; and otherwise, attenuates it. FCCF works on any desired colour space, provided that the colour pixels are expressed in terms of polar coordinates so that colour contrast rules can be applied selectively on colour pixels that fall within a pie-slice region classified as the general target colour region or colour contrast constraints.

5.4.1 rg Pie Slice Classifier

Colour recognition algorithms work by taking a single pixel and determining if it is of any of the colours specified by the current colour descriptors. This classifier works in the rg-chromaticity colour space because it helps to reduce the effects of illumination intensity [9].The algorithm takes as input a pixel in RGB format and converts it into the rg colour space.

Once the pixel has been converted into rg-Hue and rg-Saturation [9], it can simply be checked to see if it is within the bounds of the colours as defined by the pie-sliced colour descriptors.

The algorithm does not have time to calculate the rg-hue and rg-saturation values for each pixel as the inverse tangent and square root calculations take too long, so look-up tables (LUT) were created to improve the performance. The program creates this LUT on initialization by calculating the rg-Hue and rg-Saturation values for every possible combination of RGB values. These look-up tables take several minutes to build at the beginning of the program but significantly speed up the classification process ($<$ 33msec.). When a pixel is classified, the algorithm simply has to access the look-up table and the positions of the RGB values to discover the rg-Hue and rg-Saturation values.

5.4.2 Motion-Based Predictive Colour-Learning Algorithm (MPCL)

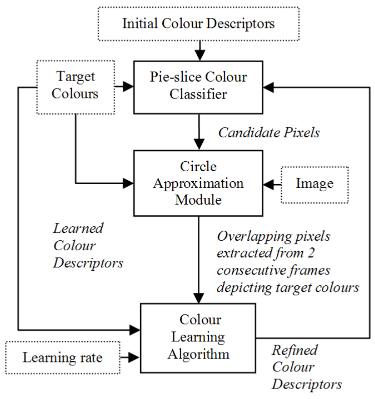
Initial Colour Descriptors
Target Colours
Pie-slice Colour Classifier
Candidate Pixels
Circle Approximation Module
Image
Learned Colour Descriptors
Overlapping pixels extracted from 2 consecutive frames depicting target colours
Colour Learning Algorithm
Learning rate
Refined Colour Descriptors

Figure 5.11: The MPCL algorithm.

In general, MPCL looks at two successive frames, extracting the best candidate pixels representing the object and fine-tuning the colour descriptors based on those pixels. For the purpose of easily finding the candidate pixels, a circularly shaped object was used during the calibration process. Nonetheless, after the system learns all the colour descriptors, the objects for tracking can come in any shape.

The series of steps for learning the colour descriptors are shown in Fig. 5.11. Initially, a broad set of colour descriptors is used by the pie-slice classifier to find the set of candidate pixels representing the target object. In turn, these pixels are fed into a circle approximation module that searches for the largest, most circular patch of colour present on the board. It calculates a formula approximating the circle by calculating the centre of the colour patch and averaging the extreme x and y values to approximate the radius of the circle. Two circle formulas will be generated for two consecutive images and the overlap of the two circles will be calculated. Once this overlap has been found the algorithm will find every pixel inside the area and filter them with the broad colour classifier to ensure that the approximated area does not include any non-colour pixels. Next, it takes all of the filtered pixels and record the extreme values for the rg-Hue and rg-Saturation values of the pixels to find the smallest possible pie-slice area that would classify every pixel inside the overlapping area. Once these extreme values have been calculated, the algorithm uses a moving average technique to adjust the actual colour descriptor parameters. The amount each set of extreme values affects the actual parameters depends on the learning rate.

Circle Generation

The circle generated for each colour patch is generated by averaging the height and width of the circular patch from the centre of the circle. Once all of the pixels in the patch have been found, a centre of gravity equation is used to find the centre of the patch:

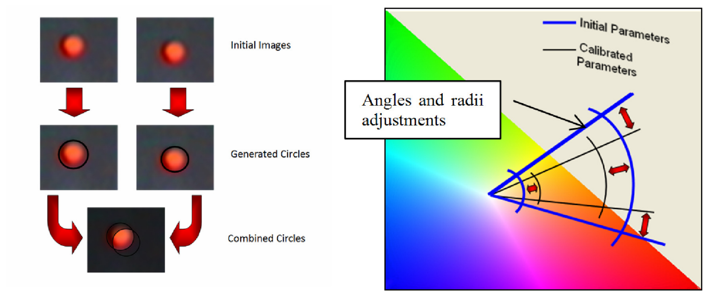

Figure 5.12: On the left is the extracted object colour pixels from two consecutive frames. On the right is the calibration of colour descriptors.

$$x_{centre} = \sum_{i=0}^{n} x_i \qquad y_{centre} = \sum_{i=0}^{n} y_i \tag{5.1}$$

Once the centre of the patch has been located, the height and width of the patch are found:

$$height = max(x_{centre}, y) \qquad width = max(x, y_{centre}) \tag{5.2}$$

Then the radius is calculated with the following equation:

$$radius = \frac{height + width}{4} \tag{5.3}$$

The centre and radius of the circle has now been found so the next part of the algorithm can run. The learning algorithm works on a moving average system combined with a decaying learning rate algorithm. The algorithm will run for a set number of iterations and keep moving average of the maximum and minimum rg-Hue and rg-Saturation:

$$rgHue_{max} = \frac{rgHue_{max}(i-1) + max(rgHue)}{i} \tag{5.4}$$

$$rgHue_{min} = \frac{rgHue_{min}(i-1) + min(rgHue)}{i} \tag{5.5}$$

$$rgSat_{max} = \frac{rgSat_{max}(i-1) + max(rgSat)}{i} \tag{5.6}$$

$$rgSat_{min} = \frac{rgSat_{min}(i-1) + min(rgSat)}{i} \tag{5.7}$$

The idea of the algorithm is to move an object to calibrate the colour. Because the object will move through all of the different illumination conditions, the algorithm will calibrate the colour classifier to work for the entire board, accounting for all possible illumination conditions.

5.5 Reading

Read Chapter 6 of Gonzales and Woods.

5.6 Exercises

1. Write a program that shows three greyscale images representing the three components R, G and B of a colour image. (hint: just create three greyscale images and copy each component to one of them).

2. Write a program that converts RGB to HSV and show the three components separately. Remember that the range of the HSV space is different than that for RGB. Adopt your own conversion table (e.g., for H, 0 is converted to 0, and 360 is converted to 255).

3. Write a program that segments images by colour. Use a range of values that is equivalent to a cubic region in the RGB space (i.e., simple thresholds for each colour component). Test your code using the images *peppers.png* and *peppers2.png* to try to separate the green peppers from the red ones.

4. Write a program that segments skin colour using two different colour systems, RGB and HSV. Collect at least 10 different images from the Internet and test different parameters, until the segmentation is comparable to the results done by hand. Which colour system achieved the best results?

Bibliography

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing.* Prentice Hall, 2002.

[2] S. K. Nayar and T. Mitsunaga, "High dynamic range imaging: Spatially varying pixel exposures," in *CVPR*, (Hilton Head, SC, USA), pp. 1472–1479, June 2000.

[3] M. D. Grossberg and S. K. Nayar, "Determining the camera response from images: What is knowable?," *IEEE Trans. PAMI*, vol. 25, pp. 1455–1467, November 2003.

[4] P. E. Debevec and J. Malik, "Recovering high dynamic range radiance maps from photographs," in *ACM SIGGRAPH 2008 classes*, SIGGRAPH '08, (New York, NY, USA), pp. 31:1–31:10, ACM, 2008.

[5] A. Barczak, T. Susnjak, N. Reyes, and M. Johnson, "Colour segmentation for multiple low dynamic range images using boosted cascaded classifiers," in *IVCNZ 2013 (Image and Vision Computing New Zealand)*, (Wellington, NZ), 2013.

[6] D. Playne, V. Mehta, N. Reyes, and A. Barczak, "Hybrid fuzzy colour processing and learning," in *ICONIP07*, (Hibikino, Japan), 2007.

[7] N. H. Reyes and E. P. Dadios, "Dynamic color object recognition," *Journal of Advanced Computational Intelligence*, vol. 8, no. 1, pp. 29–38, 2004.

[8] T. Ross, *Fuzzy Logic with Engineering Applications.* Singapore: McGraw-Hill, Inc., 1997.

[9] N. H. Reyes, *Colour-Based Object Recognition Analysis and Application.* PhD thesis, De La Salle University, 2004.