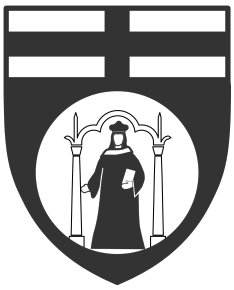

Università degli Studi di Genova

Computer Vision

Assignment 4: Color-based segmentation

Giovanni Battista Borrè, Tommaso Gruppi, Federico Tomat

April 16, 2019

Contents

Introduction

1 Introduction

1.1 Color fundamentals

Color is a rich and complex experience caused by the human vision system responding in different ways to different wavelengths of light. Color spaces (color model) are normally invented for practical reasons and so a wide variety of models may be adopted. The most common models are RGB and HSV.

1.2 RGB color space

Each color can be decomposed into combinations of the three *primary color*. Color monitors, for instance, can display millions of colors simply by mixing different intensities of red, green and blue. It is most common to place the range of intensity for each color on a scale from 0 to 255 (one byte). The range of intensity is also known as the *color depth*. The possibilities for mixing the three primary colors together can be represented as a three dimensional coordinate plane with the values for R (red), G (green) and B (blue) on each axis. This coordinate plane yields a cube called the RGB color space. If all three color channels have a value of zero, it means that no light is emitted and the resulting color is black. If all three color channels are set to their maximum values (255 at a one byte color depth), the resulting color is white. This type of color mixing is also called *additive color mixing*.

If you draw a diagonal from the black origin point of the color cube to the white point, you will get a line where each point on the line has identical RGB values. The result of having the same value for all three color channels is the color gray. The only thing that changes as you move along this diagonal is the intensity of the shade of gray as you go from black to white.

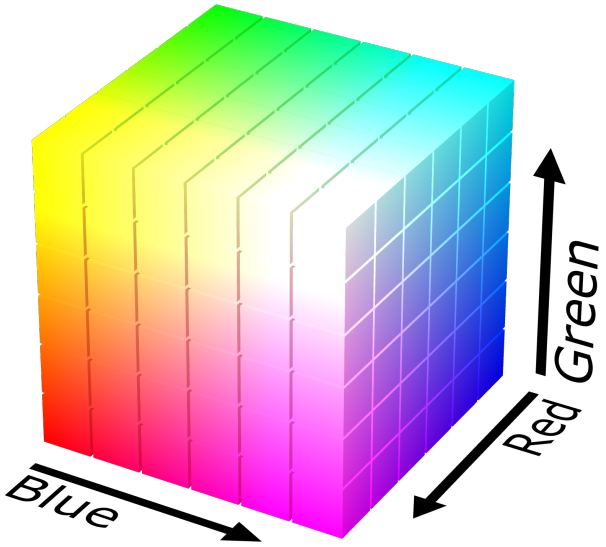

Figure 1: RGB cube

1.3 HSV color space

The HSV color space is an alternative color model (hue, saturation, value) starting at the red primary at 0 degrees, passing through the green primary at 120° and the blue primary at 240°, and then wrapping back to red at 360°. The central vertical axis comprehends the neutral, achromatic or gray colors, ranging from black at value 0% at the bottom, to white at value 100% at the top. Hue, saturation and value are defined as below reported:

- **Hue**: it is the color portion of the color model, expressed as a number from 0° to 360° degrees;
- **Saturation**: it is the amount of gray in the color, from 0 to 100%. Reducing the saturation toward zero to introduce more gray produces a faded effect. Sometimes, saturation is expressed in a range from just $(0, 1)$, where 0 is gray and 1 is a primary color.
- **Value (or Brightness)**: it works in conjunction with saturation and describes the brightness or intensity of the color, from from 0 t0 100%, where 0 is completely black, and 100 is the brightest and reveals the most color.

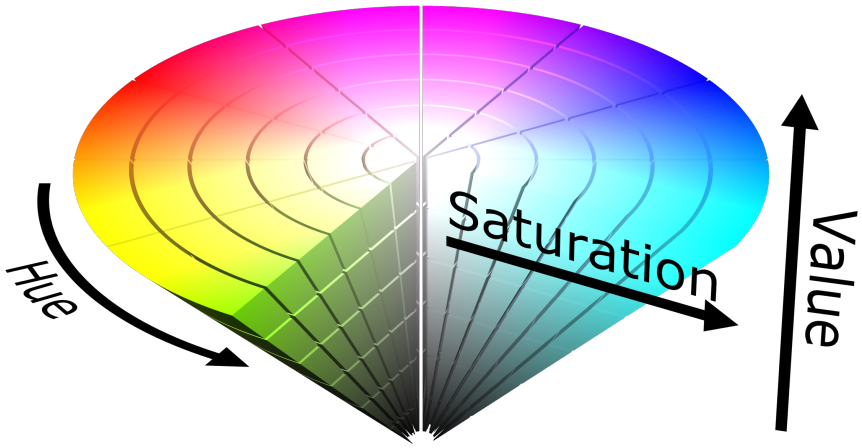

Figure 2: HSV cone

1.4 Image segmentation color-based

The region in which an object is located can be described both by edge and color. This second approach is based on the assumption that image regions have homogeneous characteristics (e.g. intensity, color) that are different from one to the other. The result is a segmentation of the image in different regions (or blobs) characterized by the same properties, since the color is well represented by hue parameter. The aim is to select each pixels in the given image that has the requested color, or at lest in the specified range. From this operation a logical image is obtained with selected pixels turned on (white) and discarded turned off (black).

First of all we separate color information (chroma) from intensity or lighting (luma). Since value is separated, you can construct a histogram or thresholding rules using only saturation and hue. This in theory will work regardless of lighting changes in the value channel. In practice it is just a nice improvement. Even by singling out only the hue you still have a very meaningful representation of the base color that will likely work much better than RGB. The end result is a more robust color thresholding over simpler parameters.

MATLAB Resolution

The first step of this assignment consists on displaying the 5 input images in grayscale and then splitting them in the respective three RGB and HSV channels.

1 Channel Creation

This step is done inside our function called:

$$[imgHsv, imgGray] = colorMaps(imgRgb) \tag{1}$$

that uses MATLAB® functions *rgb2gray*[5] and *rgb2hsv*[6] to convert the input RGB images into grayscale and HSV that are finally plotted in the main function in order to note the variations for each channel.

2 Manual Thresholding

$$[sThresh, vThresh, hThresh] = manualThreshold(imgInput) \tag{2}$$

This function requires user to select a rectangular area on the image by using the MATLAB function *getrect*[7], on whose outputs some computation are performed in order to have a well defined area on the image. The standard deviation, with *std2*[8], and the mean value, with *mean2*[9], of the selected area are compute.

3 Color Detection by HSV

$$imgDetect = colorDetection(inputImg, sThresh, vThresh, hThresh) \tag{3}$$

This function can accept up to four arguments, more precisely: if it receives just one parameter, it needs the input image, if they are three, they will be the image and the saturation and value thresholds, if they are four, the function will also have the third threshold. In the first case, using predefined threshold value, each color is found in the original image. In the second case, just the hue is predefined. In the last case all thresholds are given as argument. The output is a logical image with turn on just the pixel for which the hue, saturation and the value are included inside the thresholds.

4 Bounding Box Creation

The next step consists on selecting the region corresponding to the red car in the input image and then computing the mean value and the standard deviation of the Hue component in this area. This step is executed inside the function called:

$$[imgWithBox, frame] = colorBoundingBox(imgInput, color, \dots) \quad (4)$$

If the *colorBoundingBox* function receives four parameters, it detects the color specified as second parameter in the image passed as first one, *via colorDetection* function in automatic mode (all thresholds are predefined). If the parameters passed to the function are five and the color parameter is set to *manual* the function executes the *colorDetection* one using the thresholds passed as parameters to the original function, and obtain by using *colorDetection* and *manualThreshold*. The parameter type is used to plot the output image two different ways: *RGB* or logical. Also the last step of this assignment is performed by the property of the function *regionprops*[11], in fact it deal with the segmentation, entering the centroid and design the bounding box. The outputs of the function are: a frame obtained with *getframe*[12] and an image obtained starting from the frame using *frame2im*[13].

5 Run Simulation

The simulation is run with the function:

$$makeGif(arrayImg, delayTime, FileName, \dots) \quad (5)$$

In this part we take as input the array of images that will be the frames of the gif, the delay between each frame in the gif, the path where the gif will be saved, and all the argument needed for *colorBoundingBox*, *colorDetection* and possibly *manualThreshold* functions. In a *for* cycle the computations are performed on each of this array of input images. Please note that inside the *makeGif* function the thresholds can be set automatically or manually, calling the *manualThreshold* function. For each image the *colorBoundingBox* function with the selected threshold modality is executed. Each frame obtained as output from *colorBoundingBox* function at each cycle is stored in a frame array. At the end of the cycle the animation of all frame is showed with the function *movie*[14].

Conclusion

1 Result

First of all we have plotted, for each image, stored here or attached to the submission, all the RGB and HSV components and in the following we will focus on the HSV analysis, since it is more robust for the segmentation based on the color. After that we have compared each RGB and HSV channel of each image(from image 376 to 381). From this comparison we can note as expected that from all images the components are very similar for both models. In fact those images seem to be different frame from a single video so the scene does not change a lot. This segmentation based on the color could be a good choice since color properties remain similar in each image. From the HSV components we can note that:

- Hue: this component is pretty noisy but from this we can detect accurately the area of the car. Since it is affected from this huge quantity of noise we have previously filtered this component with a median filter through the MATLAB function *medfilt2*[10];
- Saturation: it works perfectly on detecting the car in the image and it is as expected very similar from an image to another one;
- Value: this component is not very useful and it looks like the RGB components

From the gif, stored here or attached to the assignment submission, we note that the bounding box and the centroid work well both in manual and automatic setting of the thresholds in fact the box follows the region requested and the centroid seems to be positioned in the correct position in both the running mode *red*, with predefined thresholds and *maual* with threshold adaptively computed in runtime.

2 External links

Images:
https://drive.google.com/open?id=17jlDHM-T6jtWcht1aN6eUGpDyuFUUkOC

Gifs:
https://drive.google.com/open?id=1vktIBb5N58VHFebjpCQlz_WjOO318EL2

3 GitHub Repository

https://github.com/federicotomat/Computer-Vision

Bibliography

[1] Richard Szeliski, *Computer Vision: Algorithms and Applications*, (University of Washington, 2010).

[2] Francesca Odone & Fabio Solari, *Computer Vision Course: Image Foundamental*, (University of Study of Genoa, 2019).

[3] Wikipedia on RGB

[4] Wikipedia on HSV

[5] Mathworks documentation on *rgb2gray*

[6] Mathworks documentation on *rgb2hsv*

[7] Mathworks documentation on *getrect*

[8] Mathworks documentation on *std2*

[9] Mathworks documentation on *mean2*

[10] Mathworks documentation on *medfilt2*

[11] Mathworks documentation on *regionprops*

[12] Mathworks documentation on *getframe*

[13] Mathworks documentation on *frame2im*

[14] Mathworks documentation on *movie*