Notes for Large Scale Drone Perception

Henrik Skov Midtiby

2026-02-02 14:18:38

Contents

Introduction

The topic of computer vision is a rather large topic, with a wealth of sources of information. Each source with a certain focus on the topic. These notes, that were written for the Large Scale Drone Perception course taught at the University of Southern Denmark, will focus on certain aspects of computer vision that we find especially relevant when analysing images from a drone.

It is not possible to cover all the topics extensively, but I have tried to provide a suitable overview of each topic and furthermore some additional resources are listed if you want to dig deeper into the topics.

Best regards,

Henrik Skov Midtiby

Recommended resources for computer vision

The following resources are highly recommended if you want to explore the topics of computer vision.

- The lecture series *First Principles of Computer Vision* presented by Shree Nayar from School of Engineering at Columbia University provides both an overview of the general topics and provides a treasure trove of details about many central algorithms
  Web: First Principles of Computer Vision
- The monographs from the course *First Principles of Computer Vision* by Shree Nayar.
  Web: First Principles of Computer Vision - Monographs
- The book *Computer vision: Algorithms and Applications* by Richard Szeliski also provides a good overview of the topic
  Web: Computer Vision: Algorithms and Applications by Richard Szeliski

Chapter 1

Color segmentation

The topic of this chapter is color segmentation, i.e. how to segment an image into different components based on color information on the pixel level. A way of representing colors are needed to do color segmentation, this is the color space, which will be discussed in section 1.1. Given a certain color representation, the next step is to make a decision rule for which colors belong to each of the classes of the segmentation. Some often used decision rules are described in section 1.2. In section 1.4 we will look at how to implement this in python using the opencv and numpy libraries.

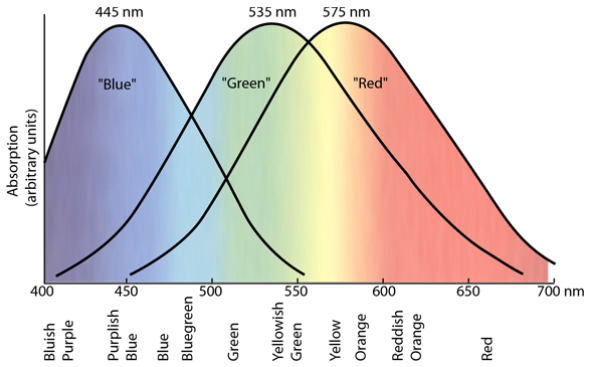

Figure 1.1: *Spectral sensitivity of the three types of cones in our eyes (Hyperphysics 2023).*

1.1 Color spaces

Digital images consist of pixels arranged in a pattern, usually a rectangular grid. Each pixel contain information about the color of that particular part of the image. How the color of a pixel is represented is denoted the *color space*. There exists many different color spaces, in this chapter the following color spaces will be discussed.

- Red, Green and Blue (RGB)
- Hue, Saturation and Value (HSV)
- CieLAB
- OK LAB

Each color space has some associated benefits and disadvantages.

1.1.1 Red, Green and blue (sRGB and linear sRGB) color space

The inspiration for the RGB color space, is how the human eyes perceive light. To sense color our eyes are equipped with three different types of *cones*, which each are sensitive to a certain range of the electromagnetic spectrum. That humans have three different types of cones, means that three color values / properties are needed to describe the colors that the human eye can perceive. The spectral sensitivity of the cones are shown in figure 1.1. By adding different amounts of red, green and blue light respectively, it is possible to generate nearly all the color that the human eye can perceive[1]. The RGB color cube is a visualization of the arrangement of colors described by the RGB color space, it is shown in figure 1.2.

In RGB, the amount of red, green and blue light that should be added to form a color is described using three numbers; often integers in the range $[0 - 255]$.

As humans are more sensitive to variations in light in dark colors, a γ (gamma) value is used to transform stored values into amounts of light using the equation

$$V_{\text{out}} = A \cdot V_{\text{in}}^{\gamma} \tag{1.1}$$

Where V_{in} is the encoded value, V_{out} is the amount of emitted light, A is a scaling factor and γ is the gamma value. The gamma value is usually around 2[2]. Two examples of a gradient between black and white is shown in figure 1.3, for the linear gradient it is very difficult to see the change in the dark colors,

1. Web: RGB color model
2. Web: Gamma correction

whereas the change is distributed more equally in the gamma corrected sGRB gradient. The nonlinear gamma correction can be problematic when doing calculations with colors[3]. In an sRGB gradient between two primary colors, the mixed colors appear to be darker than it should be. See an example in figure 1.4.

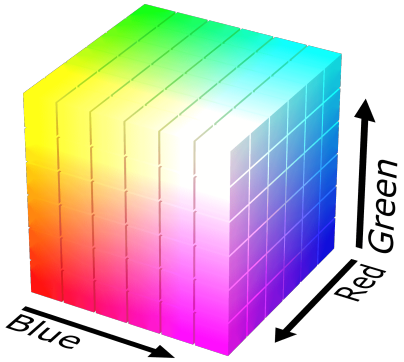

Figure 1.2: *The RGB color cube with the bright faces oriented towards the camera (Wikipedia 2023).*

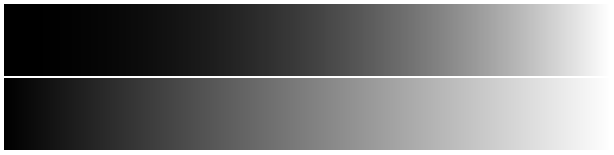

Figure 1.3: *Linear sRGB gradient (top) and gamma corrected sRGB gradient (bottom).*

Figure 1.4: *Linear sRGB gradient (top) and gamma corrected sRGB gradient (bottom).*

1.1.2 Hue, Saturation and Value / Lightning

One issue with the RGB color space is that it is very different from the way we usually describe colors, e.g.

- a dark green color
- a vibrant red color
- a pale yellow color

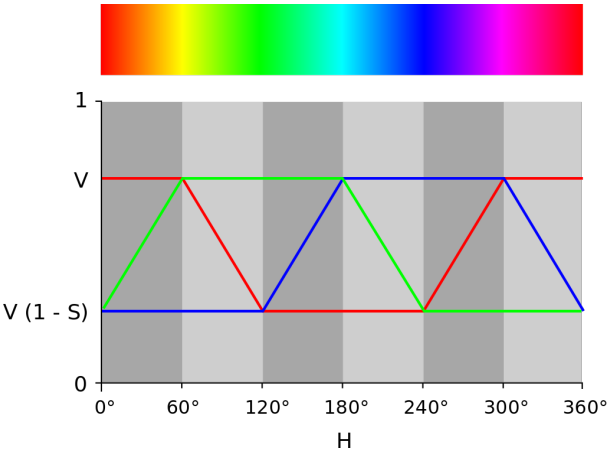

Figure 1.5: *Visual description of how to convert between HSV and RGB color representations. From wikipedia.*

In these cases a color (red, green, blue, yellow, ...) is mentioned along with a description of its brightness and saturation. The HSV color space describes colors in a very similar way. In the HSV color space a color is described in terms of the following three values[4]:

- Hue
- Saturation
- Value

Hue describe the basic color (red $\sim$ 0, yellow $\sim$ 60, green $\sim$ 120, cyan $\sim$ 180, blue $\sim$ 240 and magenta $\sim$ 270) as a number between 0 and 360. Hue is cyclic, which means that the hue values 1 and 359 are close to each other. Saturation is a number between 0 an 255 describing how much of the pure color specified by the hue is present in the color to describe, if the saturation is low, the color is a shade of gray and if it is high the color appear bright.

1.1.3 CIE LAB

The CIE LAB color space is an attempt at making a perceptually uniform color space, where distances in the color space matches the perceived difference between two colors as a human would interpret it. The three components of the color space are the lightness value L, green/red balance a and the blue/yellow balance b[5].

3. Video: Computer color is broken (4 min)
4. Web: HSL and HSV
5. Web: CIELAB color space

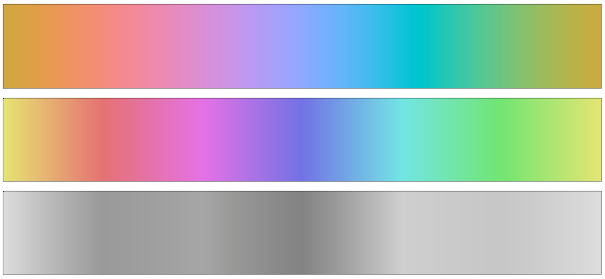

Figure 1.6: *A color gradient from OKLAB (top) compared to a similar color gradient from the HSV (middle) color space. According to the used color model, both color gradients should have constant luminosity. At the bottom is the predicted luminosity of the HSV gradient using the OK LAB color space. From (Ottosson 2023).*

1.1.4 OK LAB

OK LAB is a quite new color space, that was first described in 2020. The OK LAB color space have better numerical properties and appear more perceptually uniform than CIE LAB[6].

To explore some properties of different color spaces this interactive gradient tool is highly recommended:

- https://raphlinus.github.io/color/2021/01/18/oklab-critique.html

For more information, see

- Captain Disillusion / Color, https://www.youtube.com/watch?v=FTKPOY9MVus

1.2 Color segmentation

Given a color balue (e.g. RGB), the color based segmentation task is to determine if the color value belongs to a certain group of colors (i.e. green vegetation).

1.2.1 Independent channel thresholds

A basic approach for color based segmentation is to look at each color channel separately. E.g. if we want to see if a color is close to orange . In RGB the orange color is given by the values $(R = 230, G = 179, B = 51)$. A set of requirements for a new color to be classified as orange could be the following

$$\begin{aligned} 200 < R < 255 \\ 149 < G < 209 \\ 21 < B < 81 \end{aligned}$$

To perform such a color classification the opencv function `inRange` is useful.

This approach forces the shape of the regions in the color space to accept to have a rectangular shape.

1.2.2 Euclidean distance

A different approach is to look at the the color to classify and the reference color and then calculate a distance between these. Let R_r, G_r and B_r be the reference color value (in RGB color space) and let R_s, G_s and B_s be the color sample that should be classified.

The Euclidean distance can then be calculated using the Pythagorean theorem as follows:

$$\text{distance} = \sqrt{(R_s - R_f)^2 + (G_s - G_r)^2 + (B_s - B_r)^2}$$

If the notation $C_s = [R_s, G_s, B_s]^T$ and $C_r = [R_r, G_r, B_r]^T$, the equation can be written in a more compact form

$$\text{distance} = \sqrt{(C_s - C_r)^T \cdot (C_s - C_r)}$$

The decision rule to accept a color as being close enough to a reference colors can then be written as a requirement on the maximum allowed value of the calculated distance. This decision boundary has the shape of circles in the color space.

To determine the reference color and the threshold, a number of pixels of the object to recognize can be sampled. The reference value can then be set to the mean color value of these pixel and the threshold distance can be set to the maximum distance from the mean color value to the sampled color values.

1.2.3 Mahalanobis distance

To further adapt the decision surface to a set of sampled color values, the Mahalanobis distance can be used[7].

$$\text{distance} = \sqrt{(C_s - C_r)^T \cdot S^{-1} \cdot (C_s - C_r)}$$

6. Web: A perceptual color space for image processing
7. Web: Mahalanobis distance

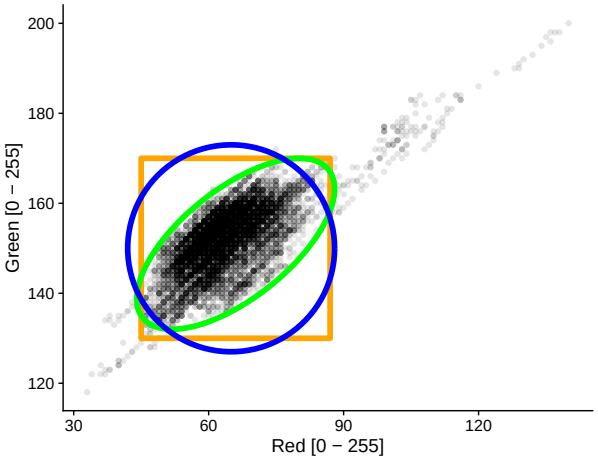

Figure 1.7: *Decision boundaries generated with the* inrange *function (orange box), the euclidean distance (blue circle) and the Mahalanobis distance (green ellipse). The plotted pixel values are sampled from an image of a patch of grass.*

where C_r is the reference color, S is a covariance matrix and C_s is the sample color.

The decision surface generated by the Mahalanobis distance is an ellipsoid in the color space.

1.2.4 Decisions boundaries

By using the described techniques for color based segmentation, a number of different regions in the color space can be delineated. These regions are visualized in figure 1.7. In the figure the red and green color values of a number of pixels are shown in a coordinate system. The boundary produced by the `inRange` function is a square depicted in orange, the boundary produced by thresholding the euclidean distance is a circle shown in blue and the boundary of thresholding the Mahalanobis distance is shown as a green ellipse.

The quality of the obtained segmentation results depend on both the chosen color space and the chosen type of decision boundary. For some tasks, tools like `inRange` works very well, and for other tasks it might be necessary to convert to a different colorspace or use a more flexible decision boundary.

1.3 Choosing a proper threshold value

For both the euclidean and the Mahalanobis distance based color segmentations, a threshold is needed to decide where to place the decision boundary between accepting the color as close enough to the reference color or not.

1.3.1 Otsu thresholding

Otsu's method is the classic approach for determining a suitable threshold to differentiate between two groups of color values. It works decently in most cases.

https://en.wikipedia.org/wiki/Otsu%27s_method

1.3.2 Generalized Histogram Thresholding

In 2020 the *Generalized Histogram Thresholding* (GHT) was introduced by Jonathan T. Barron. Generalized Histogram Thresholding performs better than Otsu's method, but uses some more involved mathematics.

The central element in the Generalized Histogram Thresholding method is the function

$$\mathrm{GHT}(\mathbf{n}, \mathbf{x}, \nu, \tau, \kappa, \omega)$$

where $\mathbf{n}$ and $\mathbf{x}$ is the histogram counts and bin centers and ν, τ, κ and ω are tuning parameters. The tuning parameter ω lets the user specify an estimated location of the threshold (as a number from zero to one) and the associated κ value is how much the provided ω value will influence the determined threshold. By varying the ν parameter from zero to infinity, the behavior of GTH changes from Minimum Error Thresholding (MET) at $\nu = 0$ to Otsu's method for $\nu \to \infty$. In between these two extremes, the GHT seems to perform better than these two methods. The following values seems to work well for many cases: $\nu = 2^5$, $\tau = 2^10$, $\kappa = 0.1$ and $\omega = 0.5$.

https://arxiv.org/abs/2007.07350

https://github.com/jonbarron/hist_thresh

1.4 Color segmentation in python

Color segmentation based on independent color channels are implemented in opencv's `inRange` func-

tion. The following example demonstrates how to use the `inRange` function

```
filename = "inputfile.jpg"
lower_limits = (130, 130, 100)
upper_limits = (255, 255, 255)
img = cv2.imread(filename)
segmented_image = cv2.inRange(img,
    lower_limits, upper_limits)
```

To extract a sample of pixels from an image, it is effective to annotate a copy of the image with a unique color (e.g. pure red (255, 0, 0)) in an image editing program like Gimp. Then the location of the annotated pixels can be determined with `inRange` and an image mask can be generated. Given the mask, the function `meanStdDev` can be used to calculate the mean and standard deviation of the color values in the original image.

```
file = "image.jpg"
file_annot = "image_annotated.jpg"
img = cv2.imread(file)
img_annot = cv2.imread(file_annot)
lower_limit = (0, 0, 245)
upper_limit = (10, 10, 256)
mask = cv2.inRange(img_annot,
    lower_limit, upper_limit)
mean, std = cv2.meanStdDev(
    img, mask = mask)
```

As there is no builtin function for calculating the covariance matrix, we need to extract the pixel values into a list (here named pixels) and then select the observations with the obtained mask. The `reshape` function is used to change the image dimensions to an array with one row per pixel and three columns with the associated color values. The average pixel value and the covariance matrix can then be found as follows using the `np.cov` and `np.average` functions:

```
pixels = np.reshape(img, (-1, 3))
mask_pixels = np.reshape(mask, (-1))
annot_pix_values = pixels[mask_pixels == 255, ]
avg = np.average(annot_pix_values, axis=0)
cov = np.cov(annot_pix_values.transpose())
```

Often it can be beneficial to perform further analysis of the pixel values (ie. to visualize them). To save the pixel values to a file, this code can be used:

```
np.savetxt("annotated_pixel_values.csv",
        annot_pix_values,
        delimiter=",",
        fmt="%d")
```

To visualize the distribution of the extracted color values, the python package `matplotlib` can be used as follows

```
import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
ax1.plot(pixels[:, 1], pixels[:, 2], '.')
ax1.set_title('Color values of a patch of grass')
plt.xlabel("Green [0 - 255]")
plt.ylabel("Red [0 - 255]")
fig.tight_layout()
plt.savefig("color_distribution.pdf", dpi=150)
```

To calculate the squared Euclidean distance to a reference color, the following code can be used:

```
shape = pixels.shape
avg_value = np.repeat([avg],
    shape[0], axis=0)
diff = pixels - avg_value
dotproduct = diff * diff
euc_dist = np.sum(dotproduct, axis=1)
euc_dist_image = np.reshape(euc_dist,
        (img.shape[0], img.shape[1]))
```

The code is based on vectorized operations of the pixel arrays. This is much faster (one or two orders of magnitude) than doing the calculation pixel by pixel[8]. To enable the vectorized calculations, the image data needs to be reshaped to a $(n \times 3)$ matrix to do the distance calculation efficient and then back to the original image dimensions.

Similarly the squared Mahalanobis distance can be computed with the code:

```
inv_cov = np.linalg.inv(cov)
moddotproduct = diff * (diff @ inv_cov)
mahalanobis_dist = np.sum(moddotproduct,
    axis=1)
mahalanobis_distance_image = np.reshape(
    mahalanobis_dist,
      (img.shape[0],
       img.shape[1]))
```

In the shown code, two different kinds of matrix multiplications are used. `*` is used to do point wise multiplication of two matrices (that have the same shape) and `@` is used to perform regular matrix multiplication.

1.5 Reference to python and OpenCV

It is highly recommended that you create a virtual python environment that you can work within when working on exercises in this course.

8. This is case in Python, if you use C / C++ this is not an issue.

- Web: Pipenv & virtual environments

Some references on how to use OpenCV in python:

- Web: Getting started with images
- Web: Drawing functions in OpenCV
- Web: Basic operations on Images
- Web: Changing Colorspaces
- Web: Image Thresholding

1.6 Getting started exercises

To get access to the support files for these exercises, do the following

- Download the zip folder with exercises from Its Learning and unzip it in a suitable location
- Open a command line and enter the directory containing the unpacked exercises
- Run the code for the first exercise as follows

```
cd 01_getting_started/solutions
uv run python ex01_draw_on_empty_image.py
```

You should now have all the required dependencies installed in a virtual environment. To run python programs from within the virtual environment use the following code

```
uv run python name_of_python_script.py
```

Each group of exercises are placed in a directory. The exercises described in this section are in the `01_getting_started` directory.

Exercise 1.6.1

Put the following content into a file named *draw_on_empty_image.py*

```
import numpy as np
import cv2
img = np.zeros((100, 200, 3), np.uint8)
cv2.line(img, (20, 30), (40, 120),
         (0, 0, 255), 3)
cv2.imwrite("test.png", img)
```

Run the following command on the command line

```
uv run python draw_on_empty_image.py
```

If everything worked, there should now be an image named "test.png" in the directory containing a black canvas with a thick red line drawn on top.

Exercise 1.6.2 hint

Load an image into python / opencv, draw something on it and save it again.

Exercise 1.6.3 hint

Load an image and save the three color channels (RGB) in separate files.

Exercise 1.6.4 hint

Load an image and save the upper half of the image in an file.

Exercise 1.6.5 hint

Load an image, convert it to HSV and save the three color channels.

Exercise 1.6.6 hint

Load an image. Locate the brightest pixel in that image and draw a circle around that pixel.

Exercise 1.6.7 hint

Load the image `flower.jpg`[9] and generate a pixel mask (a black and white image) of where the image contains yellow pixels. Use color information to determine the location of the pixel mask. Save the pixel mask to a file. Experiment with using different color spaces.

Exercise 1.6.8 hint

Load the image from exercise 1.6.7. Plot the amount of green in each pixel in a single row of the image. Use matplotlib to visualise the data. Repeat this for the two other color channels.

Exercise 1.6.9 hint hint

Load an image. Use matplotlib to visualise a histogram of the pixel intensities in the image.

Exercise 1.6.10

Calculate the average RGB pixel values of the flower petals in the image `flower.jpg`. You can use the red regions in `flower-petals-annotated.jpg` as a mask.

Exercise 1.6.11

Calculate the euclidean distance in the RGB color space from the reference color $(178, 180, 187)$ to all pixels in the image from exercise 1.6.7.

9. The image is from wikimedia: https://commons.wikimedia.org/wiki/File:Daubeny%27s_water_lily_at_BBG_(50824).jpg

Exercise 1.6.12

Plot the histogram of the euclidean distance image from exercise 1.6.11.

Exercise 1.6.13

Use the Otsu method for determining the threshold to segment the distance image in exercise 1.6.11.

Exercise 1.6.14

Use the Generalized Histogram Thresholding method for determining the threshold to segment the distance image in exercise 1.6.11.

1.7 Counting brightly colored objects

These exercises are located in the directory `02_counting_bright_objects` in the git repository with exercises. These exercises will be based on some sample images of colored plastic balls on a grass field. To download the images, run the command `make download_images` in the directory.

You should complete the three exercises with at least one under exposed image, one well exposed and one over exposed image.

To get access to the images, do the following

- Enter the subdirectory `02_counting_bright_objects/input`
- Run the command `make download_images`

The code snippet shown in figure 1.8 might be handy for debugging the segmentation exercises.

Exercise 1.7.1 hint hint

For each image locate pixels that is a) saturated in all color channels (R = G = B = 255) and b) saturated in at least one color channel (max(R, G, B) = 255). In addition count the number of saturated pixels according to the two measures.

Exercise 1.7.2 hint

We want to count the number of colored balls in the images. As a first step towards that goal, segment the images in the RGB color space. You are only allowed to look at the color channels individually (eg. $R \geq 55$) or look at linear combinations of the color channels ($G - R \geq -10$). Which of the three images are easiest to segment?

Exercise 1.7.3 hint

Segment the images in the HSV color space. You are only allowed to look at the color channels individually (eg. $H \geq 33$) or look at linear combinations of the color channels ($V - S \geq 0.2$). Which of the three images are easiest to segment?

Exercise 1.7.4 hint

Segment the images in the CieLAB color space. You are only allowed to look at the color channels individually (eg. $L \geq 33$) or look at linear combinations of the color channels ($a - b \geq 0.2$). Which of the three images are easiest to segment?

Exercise 1.7.5 hint

Choose one of the segmented images, filter it with a median filter of an appropriate size and count the number of balls in the image.

Exercise 1.7.6 hint hint

Use the python package `exifread` to extract information about the gimbal pose (yaw, pitch and roll) from one of the images.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def compare_original_and_segmented_image(original, segmented, title):
    plt.figure(figsize=(9, 3))
    ax1 = plt.subplot(1, 2, 1)
    plt.title(title)
    ax1.imshow(original)
    ax2 = plt.subplot(1, 2, 2, sharex=ax1, sharey=ax1)
    ax2.imshow(segmented)

img = cv2.imread("input/under_exposed_DJI_0213.JPG")
segmented_image = cv2.inRange(img, (60, 60, 60), (255, 255, 255))
compare_original_and_segmented_image(img, segmented_image, "test")
plt.show()
```

Figure 1.8: *Codesnippet for comparing images next to each other in python, eg. an input image and a segmented image.*

Chapter 2

Hints

First hint to 1.6.2 Back to exercise 1.6.2

Use the methods `cv2.imread`, `cv2.imwrite` and `cv2.circle`.

First hint to 1.6.3 Back to exercise 1.6.3

Look at numpy indexing.

First hint to 1.6.4 Back to exercise 1.6.4

Look at numpy indexing.

First hint to 1.6.5 Back to exercise 1.6.5

Look at `cv2.cvtColor`

First hint to 1.6.6 Back to exercise 1.6.6

Look at `cv2.minMaxLoc`

First hint to 1.6.7 Back to exercise 1.6.7

Look at `cv2.inRange`.

First hint to 1.6.8 Back to exercise 1.6.8

An example of how to plot values with matplotlib is given here.

```
import matplotlib.pyplot as plt
plt.plot([8, 3, 6, 2])
filename = "outputfile.png"
plt.savefig(filename)
```

First hint to 1.6.9 Back to exercise 1.6.9

Look at the `plt.hist` from matplotlib.

First hint to 1.7.1 Back to exercise 1.7.1

The `cv2.inRange` function can be used for a).

First hint to 1.7.2 Back to exercise 1.7.2

Open the images in gimp and inspect the color values of the balls with the chosen color.

First hint to 1.7.3 Back to exercise 1.7.3

Open the images in gimp and inspect the color values of the balls with the chosen color.

First hint to 1.7.4 Back to exercise 1.7.4

You will need a tool to convert from RGB to CIELAB values.

First hint to 1.7.5 Back to exercise 1.7.5

199 balls was taken to the field. Do not expect to see them all in an image.

First hint to 1.7.6 Back to exercise 1.7.6

The function `exifread.process_file` is a good place to start.

Second hint to 1.6.9 Back to exercise 1.6.9

The `np.reshape` function can also be handy.

Second hint to 1.7.1 Back to exercise 1.7.1

You can use `cv2.inRange` to find all pixels that are not saturated by using suitable limits. Invert the generated mask to find the partially saturated pixels.

Second hint to 1.7.6 Back to exercise 1.7.6

The `parseString` from `from xml.dom.minidom` is also handy.

Bibliography

Aanæs, Henrik (2015). *Lecture Notes on Computer Vision*.

Bishop, Christopher M (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 1st ed. Springer. ISBN: 0387310738. URL: http://www.amazon.com/Pattern-Recognition-Learning-Information-Statistics/dp/0387310738?SubscriptionId=13CT5CVB80YFWJEPWS02&tag=ws&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0387310738.

Chien, Hsiang-Jen et al. (Nov. 2016). "When to use what feature? SIFT, SURF, ORB, or A-KAZE features for monocular visual odometry". In: *2016 International Conference on Image and Vision Computing New Zealand (IVCNZ)*. IEEE, pp. 1–6. ISBN: 978-1-5090-2748-4. DOI: 10.1109/IVCNZ.2016.7804434.

Csurka, Gabriella et al. (2004). "Visual Categorization with Bags of Keypoints". In: *In Workshop on Statistical Learning in Computer Vision, ECCV*, pp. 1–22. ISBN: 9780335226375. arXiv: arXiv:1210.1833v2. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.72.604.

Garrido-Jurado, S. et al. (June 2014). "Automatic generation and detection of highly reliable fiducial markers under occlusion". In: *Pattern Recognition* 47.6, pp. 2280–2292. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2014.01.005.

Hartley, Richard (2004). *Multiple view geometry in computer vision*. Cambridge, UK New York: Cambridge University Press. ISBN: 978-0521540513.

Hata, Kenji and Silvio Savarese (2021). *CS231A Course Notes 1: Camera Models*. URL: https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf.

Hyperphysics (2023). *The Color-Sensitive Cones*. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/vision/colcon.html (visited on 01/25/2023).

KaewTraKulPong, P. and R. Bowden (2002). "An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection". In: *Video-Based Surveillance Systems*. Boston, MA: Springer US, pp. 135–144. DOI: 10.1007/978-1-4615-0913-4_11.

Karami, Ebrahim, Siva Prasad, and Mohamed Shehata (Oct. 2017). "Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images". In: *CoRR* abs/1710.0. arXiv: 1710.02726. URL: http://arxiv.org/abs/1710.02726.

Lowe, D.G. (1999). "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. IEEE, 1150–1157 vol.2. ISBN: 0-7695-0164-8. DOI: 10.1109/ICCV.1999.790410.

Lowe, David G. (Nov. 2004). "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60.2, pp. 91–110. ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000029664.99615.94.

Ottosson, Björn (2023). *A perceptual color space for image processing*. URL: https://bottosson.github.io/posts/oklab/ (visited on 01/25/2023).

Oxford Instruments (2023). *Rolling Shutter vs Global Shutter sCMOS Camera Mode*. URL: https://andor.oxinst.com/learning/view/article/rolling-and-global-shutter (visited on 03/07/2023).

Romero-Ramirez, Francisco J., Rafael Muñoz-Salinas, and Rafael Medina-Carnicer (2018). "Speeded up detection of squared fiducial markers". In: *Image and Vision Computing* 76, pp. 38–47. ISSN: 02628856. DOI: 10.1016/j.imavis.2018.05.004.

Sadekar, Kaustubh (2023). *Understanding Lens Distortion*. URL: https://learnopencv.com/understanding-lens-distortion/ (visited on 03/07/2023).

Sagitov, Artur et al. (May 2017). "Comparing fiducial marker systems in the presence of occlusion". In: *2017 International Conference on Mechanical, System and Control Engineering (ICMSC)*. IEEE, pp. 377–382. ISBN: 978-1-5090-6530-1. DOI: 10.1109/ICMSC.2017.7959505.

Scaramuzza, Davide and Friedrich Fraundorfer (Dec. 2011). "Visual Odometry: Part I: The First 30 Years and Fundamentals". In: *IEEE Robotics & Automation Magazine* 18.4, pp. 80–92. ISSN: 1070-9932. DOI: 10.1109/MRA.2011.943233.

Sivic and Zisserman (2003). "Video Google: a text retrieval approach to object matching in videos". In: *Proceedings Ninth IEEE International Conference on Computer Vision*. Vol. 2. Iccv. IEEE, 1470–1477 vol.2. ISBN: 0-7695-1950-4. DOI: 10.1109/ICCV.2003.1238663.

Wikipedia (2023). *RGB color solid cube.png*. URL: https://commons.wikimedia.org/wiki/File:RGB_color_solid_cube.png (visited on 01/25/2023).

Xiang Zhang, S. Fronz, and N. Navab (2002). "Visual marker detection and decoding in AR systems: a comparative study". In: *Proceedings. International Symposium on Mixed and Augmented Reality*. IEEE Comput. Soc, pp. 97–106. ISBN: 0-7695-1781-1. DOI: 10.1109/ISMAR.2002.1115078.