

Notes for Large Scale Drone Perception

Henrik Skov Midtiby

2024-08-08 12:06:30

Contents

1	Color segmentation	3
2	Camera settings	10
3	Shape based recognition	14
3	Pumpkin counting miniproject	14
4	Dealing with videos	16
5	Fiducial markers in images	20
6	Feature detectors and video stabilization	24
7	Monocular Visual Odometry	29
8	Bundle adjustment with g2o	33
9	Visual odometry miniproject	38
10	Scene recognition	40
11	Hints	44

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 describes in section 1.2. In section 1.4 we will look at how to implement this in python using the opencv and numpy libraries.

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 /

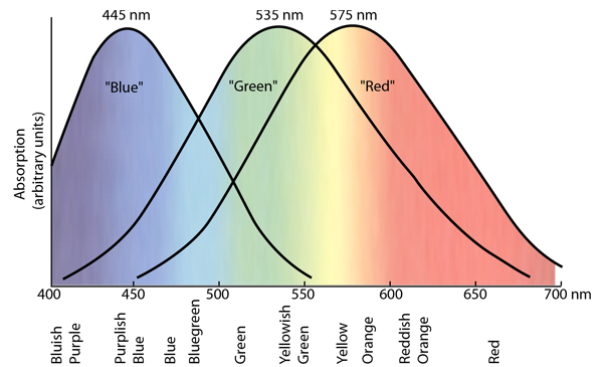


Figure 1.1: Spectral sensitivity of the three types of cones in our eyes (*Hyperphysics 2023*).

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 perceive¹. The RGB color cube is a visualization of the arrangement of colors described by the RGB color space, it is shown in figure 1.3.

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

$$V_{\text{out}} = A \cdot V_{\text{in}}^{\gamma} \quad (1.1)$$

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 2². Two examples of a gradient between black and white is shown in figure 1.2, for the linear gradient

1. Web: [RGB color model](#)
2. Web: [Gamma correction](#)

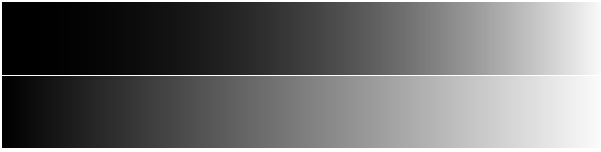


Figure 1.2: Linear sRGB gradient (top) and normal sRGB gradient (bottom).

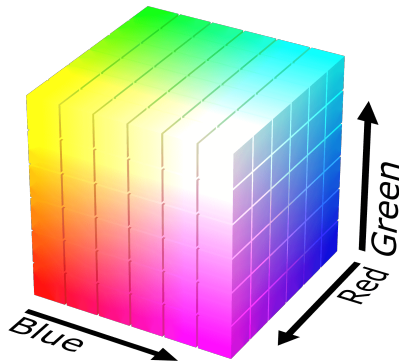


Figure 1.3: The RGB color cube with the bright faces oriented towards the camera (Wikipedia 2023).

it is very difficult to see the change in the dark colors, whereas the change is distributed more equally in the gamma corrected sRGB gradient. The non-linear gamma correction can be problematic when doing calculations with colors³. In sRGB gradients between two primary colors appear to be darker right between the two colors.

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

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 values⁴

- Hue
- Saturation

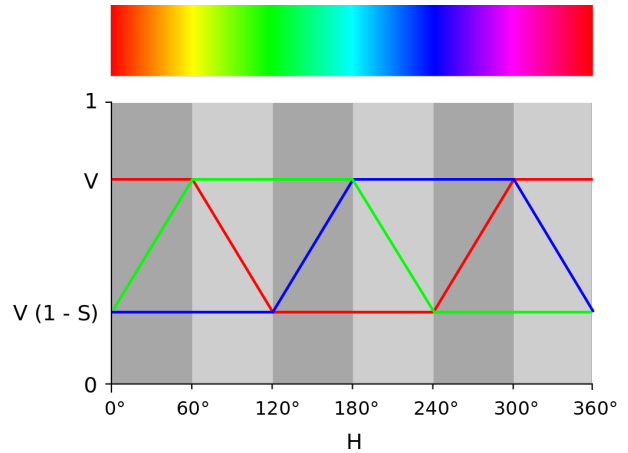


Figure 1.4: Visual description of how to convert between HSV and RGB color representations. [From wikipedia.](#)

- Value

Hue describes the basic color (red, yellow, green, cyan, blue, magenta) 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 and 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 is clear.

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 match 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.

1.1.4 OK LAB

OK LAB is a brand new color space, that was first described in 2020. The OK LAB color space has better numerical properties and appears more perceptually uniform than CIE LAB⁶.

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>

3. Video: [Computer color is broken](#) (4 min)

4. Web: [HSL and HSV](#)

5. Web: [CIE LAB color space](#)

6. Web: [A perceptual color space for image processing](#)

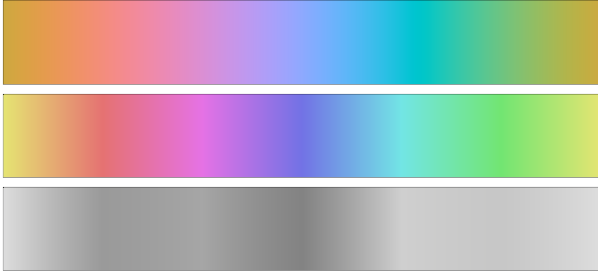



Figure 1.5: A color gradient from OKLAB (top) compared to a similar color gradient from the HSV (middle) color space. According to the used color model, both color gradients should have constant luminosity. At the bottom is the predicted luminosity of the HSV gradient using the OK LAB color space. From (Ottoosson 2023).

For more information, see

- Captain Disillusion / Color, <https://www.youtube.com/watch?v=FTKP0Y9MVus>

1.2 Color segmentation

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:

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

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 used⁷.

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

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 1.6. 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.

7. Web: [Mahalanobis distance](#)

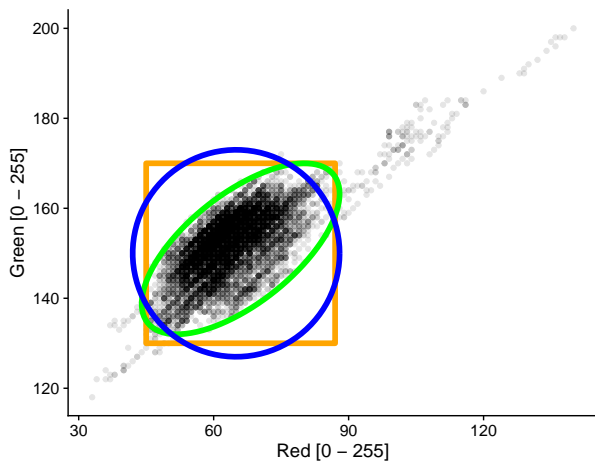


Figure 1.6: Decision boundaries generated with the `inrange` function (orange box), the euclidean distance (blue circle) and the Mahalanobis distance (green ellipse). The plotted pixel values are sampled from an image of a patch of grass.

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

Jonathan T. Barron presented in 2020 the *Generalized Histogram Thresholding* which performs better than Otsu's method, but uses some more involved mathematics.

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

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

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 GHT changes from Minimum Error Thresholding (MET) at $\nu = 0$ to Otsu's method for $\nu \rightarrow \infty$. 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` function. 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]))
```

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

To ensure that you have a proper python environment to work in, the `pipenv` module is recommended

- 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

- Clone the exercise git repository that you have been given access to on <https://gitlab.sdu.dk>
- Enter the directory containing the cloned git repository
- Create and activate a virtual python environment by running the commands shown in the box below

```
python3 -m venv env
source env/bin/activate
pip install opencv-python
pip install -r requirements.txt
```

You should now have all the required dependencies installed. 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

```
pipenv 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*⁸ 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. ’

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 1.7 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 ($\max(R, G, B)$

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


```

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

= 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.

Chapter 2

Camera settings

When capturing images there is a set of settings that needs to be chosen. These settings determines the quality of the acquired images and is therefore important to consider.

The image processing pipeline consists of multiple steps, as visualized in figure 2.1. Most courses in machine vision and image analysis will only look at the last step of the pipeline. If you are able to adjust the image acquisition it can make the following image processing much easier.

This chapter will look at the typical camera settings that can be adjusted before or during image acquisition. Some often seen image artefacts will also be discussed.

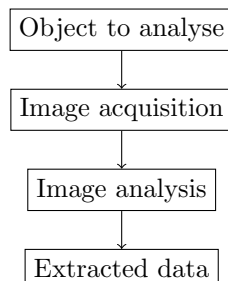


Figure 2.1: The image processing pipeline. Most courses on machine vision focus on the last two steps, whereas the LSDP course also considers the image acquisition step.

2.1 A simple image sensor

A digital camera is built around a device that is able to measure properties of the received light. This device is the imaging sensor. Today the Complementary Metal Oxide Sensor (CMOS) is the most commonly used¹. Earlier the Charge Coupled Device (CCD) was a suitable alternative to CMOS sensors. Before 2020 CCD's provided better signal to noise levels than CMOS sensors, and were preferred in astronomical telescopes.

The imaging sensor is able to count the number of photons that each pixel in the sensor receive. The pixels are arranged in a rectangular pattern. In combination with the lens, this enables the sensor to measure the number of photons coming from different directions².

The properties that can be adjusted in a camera system are the following^{3 4}:

- the aperture, which limits the amount of light that passes through the lens
- the shutter time, in which the sensor registers new photons
- the ISO sensor sensitivity, which determines the gain of the sensor, how many photons can be received before the sensor is saturated?

It depends on the situation, but usually the settings are specified in the following order

1. shutter time, high values are preferred, but motion blur should usually be avoided
2. aperture, high values are preferred, if it is difficult to get the object in focus the aperture can be lowered
3. ISO, choose such that the exposure of the image is as desired. This can be checked through a histogram.

Personally I prefer to have images slightly underexposed.

1. Web: [Active pixel sensor](#)
2. Web: [What camera measure](#)
3. Web: [A Comprehensive Beginner's Guide to Aperture, Shutter Speed, and ISO](#)
4. Video: [Aperture, Shutter Speed, ISO, & Light Explained ... \(14 min\)](#)

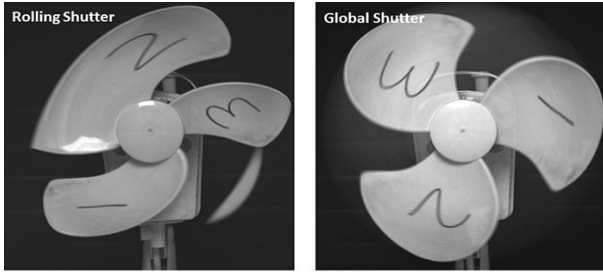


Figure 2.2: Illustration of artefacts from using a rolling shutter camera. From (Oxford Instruments 2023).

2.1.1 Rolling and global shutter

There are also two different ways of reading out data from image sensors, rolling shutter and global shutter. In an imaging sensor with global shutter, the entire image is exposed simultaneously. After exposure all pixel values are read out. This makes it easier to interpret the images and they do not suffer from distortions based on motion of either the camera or the object in the scene. The main disadvantage of global shutter is that the frame rate is reduced compared to a rolling shutter sensor.

In a rolling shutter sensor, one row of pixels is read out at a time. This means that the acquired image contains elements exposed at different times; this causes deformation of objects in motion⁵.

Additional resources on rolling and global shutter

- Web: [Global & rolling shutter](#)
- Web: [Rolling vs Global Shutter](#)

2.1.2 Camera settings when capturing video

Special consideration should be taken when capturing video. It can be desirable to have some amount of motion blur in the recorded video, which ideally should match the used frame rate. It is recommended to use shutter speed of 50% of the time between adjacent frames when capturing video. Eg. when capturing video with 30 fps, the shutter speed should be around $1 / 60$. To achieve a suitable exposure, it is sometimes needed to mount a *neutral density* filter in front of the camera, to reduce the amount of light that enters the camera. If much lower, the motion blur does not match what we will expect and the frames would look choppy⁶.

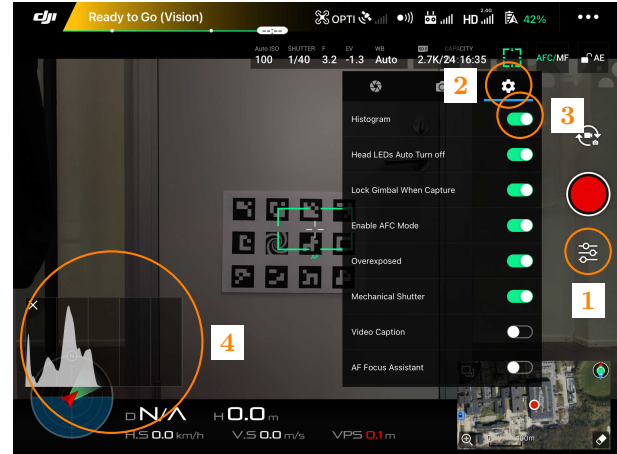


Figure 2.3: How to enable the histogram in DJI Go 4. Open camera settings (1), gear (2), enable histogram (3) and the histogram is now visible (4).

2.1.3 Adjusting camera settings on a DJI drone

When capturing images with a DJI drone, it is possible to adjust the camera settings while the drone is in the air. To get an idea about the exposure with the current camera settings, it is helpful to enable the histogram in DJI 4 Go (see figure 2.3).

It is also possible to control the camera settings (see figure 2.4). The camera modes that are available are *Auto*, *Aperture priority*, *Shutter priority* and *Manual*. Usually you can get decent results by setting the camera on auto and then adjust the exposure. Most of my flights are conducted with the exposure set to $EV = -1.3$.

- Web: [DJI Mavic Pro Basic Exposure Settings HELP!!](#)

2.2 Optical filters

Linear polarization can be used to remove reflections from vertical (e.g. a window) or horizontal surfaces (e.g. a sea surface)⁷.

A band pass filter can be used to only allow a certain part of the electromagnetic spectrum to reach the sensor. High pass and low pass filters also exist.

5. Video: [Why Do Cameras Do This? \(Rolling Shutter Explained\) \(7 min\)](#)

6. Web: [Frame rate vs. shutter speed, setting the record straight](#)

7. Web: [Polarizing filter \(photography\)](#)

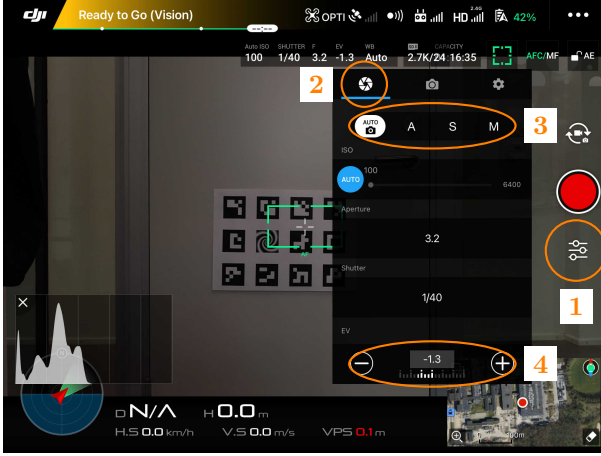


Figure 2.4: Finding the camera settings in DJI Go 4. Settings (1), camera (2), camera mode (3), exposure (4).

2.3 Pinhole camera model

The pinhole camera model is used for mapping 3D world coordinates to 2D image coordinates as follows:

$$s \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \cdot [R | \mathbf{t}] \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad (2.1)$$

The pinhole model maps the 3D world coordinate (given in homogeneous coordinates) $(X, Y, Z, 1)^T$ to 2D image coordinates (also in homogeneous coordinates). To compute the mapping, the location and orientation of the camera is used. This is specified as a position vector \mathbf{t} and rotation matrix R of the optical centre of the camera. Some parameters related to the optical system is encoded K matrix. The included parameters cover the focal length in the x and y -directions (f_x and f_y) and the location of the axis of the optical system (c_x and c_y) as well as the parameter describing the skew γ between the x and y axes of the camera sensor. On modern cameras the skew parameter can usually be set to zero.

$$K = \begin{pmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

- Web: [Geometry of image formation](#)

2.4 Lens distortion

The pinhole model introduced in 2.3 makes the assumption that all light that reaches the camera sen-

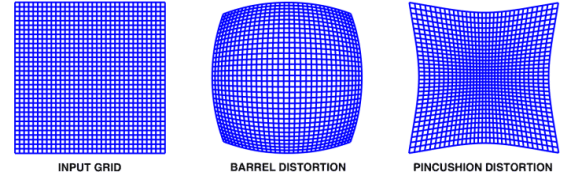


Figure 2.5: Examples of radial lens distortions. From (Sadekar 2023).

sor passes through a single point – the pin hole. However this is usually replaced with an optical system, consisting of one or more lenses. Such an optical system will introduce distortions to the formed image. These distortions can be divided into two groups

1. radial distortions
2. tangential distortions

The radial distortions are introduced by the optical system, and can be modelled with the equations:

$$r = \sqrt{(x - x_c)^2 + (y - y_c)^2} \quad (2.2)$$

$$x_{\text{distorted}} = x \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (2.3)$$

$$y_{\text{distorted}} = y \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (2.4)$$

where r is the distance to the optical axis of the system. The tangential distortions can similarly be modelled with the equations.

$$x_{\text{distorted}} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \quad (2.5)$$

$$y_{\text{distorted}} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \quad (2.6)$$

In most cases the three parameters (k_1 , k_2 and k_3) modelling the radial distortions is enough to describe the distortions. If a few cases it could make sense to use additional parameters if the distortion is hard to model.

- Web: [Camera calibration With OpenCV](#)
- Web: [Understanding lens distortion](#)
- Web: [Camera Calibration and 3D Reconstruction – Detailed description](#)

2.5 Camera calibration

Camera calibration is the process of estimating both the parameters in the pinhole camera model and the parameters in the lens distortion model.

- Web: [Camera calibration using OpenCV](#)
- Web: [Camera calibration with large chessboards](#)

2.6 Projecting observations on to the ground plane

If we look at objects at a known altitude (ie. the sea surface with $Z = 0$) compared to the camera and also know the camera position and orientation, it is possible to calculate the 3D world coordinates of the point we are looking at. So given the image coordinates u and v , the task is to estimate the X and Y parts of the real world coordinates, provided this equation.

$$s \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \cdot [R | \mathbf{t}] \cdot \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} \quad (2.7)$$

2.7 Collection of data

There is a lot of things to consider prior to going out on a mission to acquire data with a drone. To ensure that the mission can be completed successfully, checklists can be used to ensure that you take care of things in the right order. When operating UAVs checklists are used to ensure the safety and success of the mission that is to be conducted.

During prototype tests with a B17 bomber in 1935, the pilots failed to configure the plane properly, which caused the plane to crash. To avoid similar problems in the future the company behind the plane, Boeing, introduced preflight checklists to ensure that the plane was configured correctly prior to takeoff ⁸.

If you want to read more about checklists and their adoption in aviation this source could be of interest Web: [Not flying by the book: Slow adoption of checklists and procedures in WW2 aviation](#)

Exercise 2.7.1

Discuss what actions you would take before taking off with a multirotor UAV. Then make a pre flight checklist for a multirotor UAV.

Exercise 2.7.2

Discuss what actions you would take after landing with a multirotor UAV. Then make a post flight checklist for a multirotor UAV.

Exercise 2.7.3

Collect images with different exposures

- Under exposed images

- Well exposed images
- Over exposed images

Collect images with different orientations of the camera

- Change yaw or pitch

Exercise 2.7.4

Take a look at some of the acquired images (eg. DJI_0001.JPG and DJI_0177.JPG). Extract information from the image about the position of the UAV during image capture and the orientation of the camera. Try to verify the information by locating known objects in the images.

Exercise 2.7.5

[hint](#)

A set of images with severe motion blur have been acquired and you have to redo the flight. Which camera settings would you modify for the flight?

Exercise 2.7.6

Take the image DJI_0177.JPG and find information about the camera orientation in the exif data that is embedded in the image. Modify the camera projection script (provided as matlab code) to match the used camera orientation.

Exercise 2.7.7

[hint](#)

Project the image from exercise 2.7.6 down on the ground plane.

Exercise 2.7.8

Take multiple images and project them down onto the same ground plane. The images should be taken with the UAV in the same position but with differences in the gimbal orientation.

2.8 Loose ends

- Web: [Quick D: Static helicopter blades](#)
- Web: [Understanding lenses, aperture, focal length and fisheye](#)
- Web: [CD / Shutter Speed](#)

8. Web: [Checklist born – B-17 Bomber pre-flight checklist](#)

Chapter 3

Shape based recognition

The exercises described in this section are in the `02_shape_based_object_classification` directory.

3.1 Feature extraction

Exercise 3.1.1 [hint](#)

Open the file `input/shapes.png`, convert it to grayscale and extract contours of the image using the Canny edge detector. Draw the extracted contours on top of a black image.

Exercise 3.1.2 [hint](#)

Continuation of exercise 3.1.1. For all of the extracted contours calculate the length of the perimeter and the area enclosed by the contour. Explain why some of the calculated areas are negative.

Exercise 3.1.3 [hint](#)

Continuation of exercise 3.1.2. For each of the detected contours classify whether it is a circle or not by looking at the ratio

$$\frac{\text{perimeter}^2}{4\pi\text{area}}$$

Exercise 3.1.4 [hint](#)

Continuation of exercise 3.1.2. Extract central moments and then calculate the seven moments of Hu for each of the detected contours.

3.2 Support vector machine classification

Exercise 3.2.1

Go through this example of using a support vector machine from the opencv documentation: <https://docs.opencv2.org/3.4/d1/d73/>

[tutorial_introduction_to_svm.html](#). Explain what each line does in the example. Identify the parts of the code that should be changed to handle more than two object classes.

Exercise 3.2.2

For each of the images `input/numbers/1.png`, `input/numbers/2.png`, `input/numbers/3.png`, ... `input/numbers/9.png`, extract all contours and calculate the contour moments as well as the seven moments of Hu. The calculated numbers should be shown on the screen with a suitable number of significant digits.

Exercise 3.2.3

This exercise is a continuation of exercise 3.2.2. For each of the detected contours save the following information in a suitable data structure:

- object class (1, 2, 3, ..., 9)
- area of object
- perimeter of object
- the mu12 central image moment

Exercise 3.2.4 [hint](#) [hint](#)

Train a support vector machine classifier with the features extracted in exercise 3.2.3. Use the classifier to recognize digits in the sudoku `input/sudoku01bw.png`

Chapter 11

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.

First hint to 2.7.5 [Back to exercise 2.7.5](#)

Exposure time should be decreased while ISO or aperture increased to compensate for the reduction in light on the sensor.

First hint to 2.7.7 [Back to exercise 2.7.7](#)

OpenCV hint: Look at the functions: `getPerspectiveTransform` and `warpPerspective`.

First hint to 3.1.1 [Back to exercise 3.1.1](#)

These methods should be used in the solution

```
cv2.cvtColor
cv2.Canny
cv2.findContours
cv2.drawContours
```

First hint to 3.1.2 [Back to exercise 3.1.2](#)

These methods should be used in the solution

```
cv2.arcLength
cv2.contourArea
```

First hint to 3.1.3 [Back to exercise 3.1.3](#)

Consider what this ratio is for different basic shapes (eg. a triangle, a square and a circle).

First hint to 3.1.4 [Back to exercise 3.1.4](#)

The following methods will help you with the solution

`cv2.moments`
`cv2.HuMoments`

First hint to 3.2.4 [Back to exercise 3.2.4](#)

What is the range of each of the numeric feature descriptors.

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 `xml.dom.minidom` is also handy.

Second hint to 3.2.4 [Back to exercise 3.2.4](#)

The performance of the classifier can be quite sensitive to large differences in the scale of features.

Bibliography

- Aanaes, 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](https://doi.org/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](https://arxiv.org/abs/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](https://doi.org/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.
- 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](https://doi.org/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](https://arxiv.org/abs/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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. Frönz, 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](#).