

Notes for Large Scale Drone Perception

Henrik Skov Midtby

2022-03-02 09:03:51

Contents

1	Color segmentation	3
2	Camera settings	8
3	Pumpkin counting miniproject	11
4	Hints	13

Introduction

These notes are written for the Large Scale Drone Perception course taught by Elzbieta Pastucha and Henrik Skov Midthiby at the University of Southern Denmark.

The notes provides an overview of the topics discussed in class. To get more detailed knowledge please refer to the references in the text.

Best regards,
Henrik

Additional literature

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 describes in section 1.2. In section 1.3 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 (RGB) 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.

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.

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. The nonlinear gamma correction can be problematic when doing calculations with colors³.

1.1.2 Hue, Saturation and Value / Lightning

On 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 vary similar way. In the HSV color space a color is described in terms of the following three values⁴

- Hue
- Saturation

¹Web: [RGB color model](#)

²Web: [Gamma correction](#)

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

⁴Web: [HSL and HSV](#)

- Value

Hue describe 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, is the saturation is low, the color is a shade of gray and if it is high the color is clear.

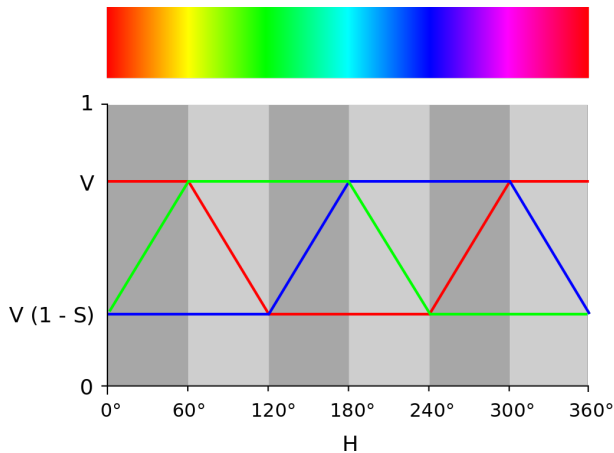


Figure 1.1: Visual description of how to convert between HSV and RGB color representations. *From wikipedia.*

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 equals 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.

1.1.4 OK LAB

OK LAB is a brand 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 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>


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

⁵Web: [CIELAB color space](#)

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

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

$$200 < R < 255$$

$$149 < G < 209$$

$$21 < B < 81$$

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

⁷Web: [Mahalanobis distance](#)

1.3 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. full red) in an image editing program like Gimp. Then the location of the annotated pixels can be determined with `inRange` 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 weight 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. Then the average pixel value and the covariance matrix can 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 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]))
```

1.4 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](#)

1.5 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
- Run the command
`pipenv install`

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 1.5.1

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 1.5.2 hint

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

Exercise 1.5.3 hint

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

Exercise 1.5.4 hint

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

Exercise 1.5.5 hint

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

Exercise 1.5.6 hint

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

Exercise 1.5.7 hint

Download the image [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).

Load the image and generate a pixel mask (bw image) of where the image contains yellow pixels. Save

the pixel mask to a file. Experiment with using different colorspace.

Exercise 1.5.8 hint

Load the image from exercise 1.5.7. 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 1.5.9 hint hint

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

Exercise 1.5.10

Try to segment the image from exercise 1.5.7 by calculating the euclidean distance to the BGR color (187,180,178).

1.6 Counting brightly colored objects

These exercises will be based on some sample images of colored plastic balls on a grass field. 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

- Clone the exercise git repository that you have been given access to on <https://gitlab.sdu.dk>
- Enter the subdirectory
`02_counting_bright_objects/input`
- Run the command
`make download_images`

The code snippet shown in figure 1.2 might be handy for debugging the segmentation exercises.

Exercise 1.6.1 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 1.6.2 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 1.6.3 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

```

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

the color channels ($V - S \geq 0.2$). Which of the three images are easiest to segment?

Exercise 1.6.4 [hint](#)

Segment the images in the CieLAB 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 1.6.5 [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 1.6.6 [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 determine can affect the quality of the acquired images and is therefore import 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.

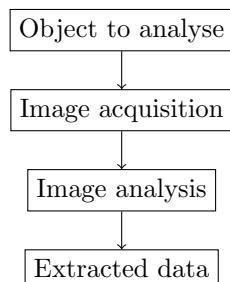


Figure 2.1: Image processing pipeline

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.

¹Web: [Active pixel sensor](#)

²Web: [What camera measure](#)

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

- the aperture, which limits the amount of light that reaches the imaging sensor
- the shutter time, in which the sensor counts photons
- the ISO sensor sensitivity, which determines the gain of the sensor, how many photons can be received before the sensor is saturated?

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 it 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](#)

2.1.1 Camera settings when capturing video

Usually try to achieve an open shutter for 50% of the time when capturing video. If much lower, the motion blur does not match what we will expect and the frames would look choppy⁶.

³Web: [A Comprehensive Beginner's Guide to Aperture, Shutter Speed, and ISO](#)

⁴Video: [Aperture, Shutter Speed, ISO, & Light Explained ... \(14 min\)](#)

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

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

2.1.2 Adjusting camera settings on a DJI drone

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.

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 sensor 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:

$$x_{\text{distorted}} = x \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (2.2)$$

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

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

$$x_{\text{distorted}} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \quad (2.4)$$

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

In most cases the three parameters (k_1 , k_2 and k_3) modelling the radial distortions is enough to describe the distortions. In 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](#)

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.

$$s \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \cdot [R | \mathbf{t}] \cdot \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} \quad (2.6)$$

2.7 Collection of data

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

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

plane, Boeing, introduced preflight checklists to ensure that the plane was configured correctly prior to takeoff ⁸.

When operating UAVs checklists are used to ensure the safety and success of the mission that is to be conducted.

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](#)

You have acquired a set of images with severe motion blur and 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](#)

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

Chapter 3

Pumpkin counting miniproject

In this mini-project, you will work in groups of up to three students. The project deals with how to estimate the number of pumpkins in a set of UAV images from a single pumpkin field. To estimate this number, you will need to apply several of the methods that we have discussed in the class.

We expect you to hand in a report that describes how you have dealt with the exercises listed below. Please include the following elements in the report:

- a description of the overall goal of the project
- a description of what has been done as part of each exercise
- references to sources of information
- source code for reproducing the results
- example input data
- example output data
- comments on the obtained results

The report should be handed in before Thursday the 17th of March at midnight (23.59) using IT's Learning.

The dataset that should be used for the project can be downloaded from this link: <https://nextcloud.sdu.dk/index.php/s/L8J4iw7FMCTzS2K>

The project is divided into four parts 1) colour based segmentation, 2) counting of objects in a single image, 3) generation of orthomosaics and 4) finally counting of pumpkins in the entire field. The three first parts can be solved more or less independently whereas the fourth part can only be completed when all the other elements are in place.

Ela will be present to help with questions related to the project in the scheduled class session on March 4th.

3.1 Colour based segmentation

This part consists of using colour based information to segment individual images from a pumpkin field. The segmented images should end up having a black background with smaller white objects on top.

Exercise 3.1.1

[hint](#)

Annotate some pumpkins in a test image and extract information about the average pumpkin colour in the annotated pixels. Calculate both mean value and standard variation. Use the following two colour spaces: RGB and CieLAB. Finally try to visualise the distribution of colour values.

Exercise 3.1.2

Segment the orange pumpkins from the background using color information. Experiment with the following segmentation methods

1. `inRange` with RGB values
2. `inRange` with CieLAB values
3. Distance in RGB space to a reference colour

Exercise 3.1.3

Choose one segmentation method to use for the rest of the mini-project.

3.2 Counting objects

This part is about counting objects in segmented images and then to generate some visual output that will help you to debug the programs.

Exercise 3.2.1

[hint](#)

Count the number of orange blobs in the segmented image.

Exercise 3.2.2

[hint](#)

Filter the segmented image to remove noise.

Exercise 3.2.3

Count the number of orange blobs in the filtered image.

Exercise 3.2.4

[hint](#) [hint](#)

Mark the located pumpkins in the input image. This step is for debugging purposes and to convince others that you have counted the pumpkins accurately.

3.3 Generate an orthomosaic

This part deals with orthorectifying multiple images of the same field into a single cartometric product.

Choose proper settings for all below processes, taking into consideration the available computing resources

Exercise 3.3.1

Load data into Metashape.

Exercise 3.3.2

Perform bundle adjustment (align photos) and check results

Exercise 3.3.3

Perform dense reconstruction

Exercise 3.3.4

Create digital elevation model

Exercise 3.3.5

Create orthomosaic

Exercise 3.3.6

Limit orthomosaic to pumpkin field

3.4 Count in orthomosaic

Use the python package `rasterio` to perform operations on the orthomosaic using a tile based approach.

Exercise 3.4.1

[hint](#) [hint](#) [hint](#)

Create code that only loads parts of the orthomosaic.

Exercise 3.4.2

[hint](#)

Design tile placement incl. overlaps.

Exercise 3.4.3

Count pumpkins in each tile.

Exercise 3.4.4

Deal with pumpkins in the overlap, so they are only counted once.

Exercise 3.4.5

Determine amount of pumpkins in the entire field.

3.5 Endnotes

Reflect on the conducted work in this miniproject.

Exercise 3.5.1

Determine GSD and size of the image field. What is the average number of pumpkins per area?

Exercise 3.5.2

Reflect on whether the developed system is ready to help a farmer with the task of estimating the number of pumpkins in a field.

Chapter 4

Hints

First hint to 1.5.2 [Back to exercise 1.5.2](#)

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

First hint to 1.5.3 [Back to exercise 1.5.3](#)

Look at [numpy indexing](#).

First hint to 1.5.4 [Back to exercise 1.5.4](#)

Look at [numpy indexing](#).

First hint to 1.5.5 [Back to exercise 1.5.5](#)

Look at `cv2.cvtColor`

First hint to 1.5.6 [Back to exercise 1.5.6](#)

Look at `cv2.minMaxLoc`

First hint to 1.5.7 [Back to exercise 1.5.7](#)

Look at `cv2.inRange`.

First hint to 1.5.8 [Back to exercise 1.5.8](#)

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 1.5.9 [Back to exercise 1.5.9](#)

Look at the `plt.hist` from matplotlib.

First hint to 1.6.1 [Back to exercise 1.6.1](#)

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

First hint to 1.6.2 [Back to exercise 1.6.2](#)

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

First hint to 1.6.3 [Back to exercise 1.6.3](#)

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

First hint to 1.6.4 [Back to exercise 1.6.4](#)

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

First hint to 1.6.5 [Back to exercise 1.6.5](#)

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

First hint to 1.6.6 [Back to exercise 1.6.6](#)

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](#)

Information from section 1.3 might be handy.

First hint to 3.2.1 [Back to exercise 3.2.1](#)

The `findContours` method can help.

First hint to 3.2.2 [Back to exercise 3.2.2](#)

Consider to apply Gaussian blur or a median filter.

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

I prefer to draw circles on top of each detected pumpkin.

First hint to 3.4.1 [Back to exercise 3.4.1](#)

```
# Hint 1: to get image resolution without
# loading it into memory:
import rasterio
```

```
filename = "example.tif"
with rasterio.open(filename) as src:
    columns = src.width
    rows = src.height
```

First hint to 3.4.2 [Back to exercise 3.4.2](#)

How do you deal with pumpkins on the edge between two tiles?

Second hint to 1.5.9 [Back to exercise 1.5.9](#)

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

Second hint to 1.6.1 [Back to exercise 1.6.1](#)

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 1.6.6 [Back to exercise 1.6.6](#)

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

Second hint to 3.2.4 [Back to exercise 3.2.4](#)
The methods `circle` and `moments` are helpful. See
`moments` in action [here](#).

Second hint to 3.4.1 [Back to exercise 3.4.1](#)

```
# Hint 2: to load part of an image:
import rasterio
from rasterio.windows import Window

filename = "example.tif"
with rasterio.open(filename) as src:
    # tile ulc - upper left corner,
    # lower left corner... and so on.
    window_location = Window.from_slices(
```

```
(tile.ulc[0], tile.lrc[0]),
(tile.ulc[1], tile.lrc[1]))
img = src.read(window=window_location)
```

Third hint to 3.4.1 [Back to exercise 3.4.1](#)

Hint 3: loaded image has a different shape
than opencv image, so...

```
temp = img.transpose(1, 2, 0)
t2 = cv2.split(temp)
img_cv = cv2.merge([t2[2], t2[1], t2[0]])
```

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>.
- 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.
- 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).
- 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).
- 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](https://doi.org/10.1109/ICCV.2003.1238663).
- 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](https://doi.org/10.1109/ISMAR.2002.1115078).