LABexc7-ELE510-2021

October 18, 2021

1 ELE510 Image Processing with robot vision: LAB, Exercise 7, Stereo Vision and Camera Calibration.

Purpose: *To learn about imaging with two cameras, stereo, and reconstrution by triangulation.*

The theory for this exercise can be found in chapter 13 of the text book [1] and in appendix C in the compendium [2]. See also the following documentations for help: - OpenCV - numpy - matplotlib - scipy

IMPORTANT: Read the text carefully before starting the work. In many cases it is necessary to do some preparations before you start the work on the computer. Read necessary theory and answer the theoretical part first. The theoretical and experimental part should be solved individually. The notebook must be approved by the lecturer or his assistant.

Approval:

The current notebook should be submitted on CANVAS as a single pdf file.

`To export the notebook in a pdf format, goes to File -> Download as -> PDF via LaTeX (.pdf).`

Note regarding the notebook: The theoretical questions can be answered directly on the notebook using a *Markdown* cell and LaTex commands (if relevant). In alternative, you can attach a scan (or an image) of the answer directly in the cell.

Possible ways to insert an image in the markdown cell:

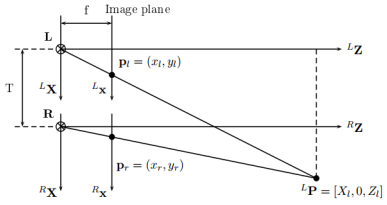
```
![image name]("image_path")
```

```
<img src="image_path" alt="Alt text" title="Title text" />
```

Under you will find parts of the solution that is already programmed.

```
<p>You have to fill out code everywhere it is indicated with `...`</p>
<p>The code section under `######## a)` is answering subproblem a) etc.</p>
```

1.1 Problem 1 (Correspondence problem)



Assume that we have a simple stereo system as shown in the figure. **L** and **R** denotes the focal point of the Left and Right camera respectively. **P** is a point in the 3D world, and **p** in the 2D image plane. ${}^{L}\mathbf{P}_w$ denotes a world point with reference to the focal point of the Left camera. The baseline (line between the two optical centers) is $T = 10\,\text{cm}$ and the focal length $f = 2\,\text{cm}$.

a) Consider the scene point ${}^{L}\mathbf{P}_w = [0.1\text{m}, 0, 10\text{m}]^T$. Suppose that due to various errors, the image coordinate x_l is 1 percent **smaller** than its true value, while the image coordinate x_r is perfect. What is the error in depth z_w, in millimeters?

b) An image of resolution 1500×1500 pixels is seen by the Left and Right cameras. The image sensor size is 20mm $\times$ 20mm. Let the disparity in the image coordinates be up to 75 pixels. Using the same focal point and baseline, what is the depth of the image compare to the cameras?

c) What are the typical application for the correspondence problem?

Section a)

To find the error we need not only our value of x_l but also the its true value, so we calculate it as follows:

$$x_l = f \cdot \frac{X_l}{Z_l} \implies x_l = 0.02 \cdot \frac{0.1}{10} = 2 \cdot 10^{-4} m \tag{1}$$

Given an error of 1% of our true value, we get that our current value is $0.99 \cdot x_l = 1.98 \cdot 10^{-4} m$. We can calculate our current Z_l with this value:

$$Z_l = f \cdot \frac{X_l}{x_l} \implies Z_l = 0.02 \cdot \frac{0.1}{1.98 \cdot 10^{-4}} = 10.1 m \tag{2}$$

We can get the error as follows:

$$\Delta Z_l = |Z_l^+ - Z_l^-| \implies \Delta Z_l = |10 - 10.1| = 0.1 m \tag{3}$$

We have an error of 100mm in depth.

Section b)

Disparity is inversely proportional to depth:

$$d = \frac{f \cdot T}{Z_l} \implies Z_l = \frac{f \cdot T}{d} \tag{4}$$

First, we need to convert the disparity pixels into a real worl metric. We have $20mm^2$ sensors and 1500x1500 pixels of resolution, this means:

$$\frac{20mm}{1500pix} = 0.01\bar{3}mm \tag{5}$$

We have 75 pixels of disparity, if we convert them to metric units we get the following:

$$75pix \cdot 0.01\bar{3}mm = 1mm \implies d = 0.001m \tag{6}$$

Replacing in our first equation we get the depth Z_l:

$$Z_l = \frac{0.02 \cdot 0.1}{0.001} = 2m \tag{7}$$

The image would have a depth up to 2 meters.

Section c)

Estimation of 3D world points (very useful in computer vision): - Machines that can grab objects thanks to cameras - Virtual reality for games (Xbox kinect) - High quality in photographies

1.2 Problem 2 (Block Matching)

The simplest algorithm to compute dense correspondence between a pair of stereo images is **block matching**. Block matching is an *area-based* approach that relies upon a statistical correlation between local intensity regions.

For each pixel (x,y) in the left image, the right image is searched for the best match among all possible disparities $0 \leq d \leq d_{\max}$.

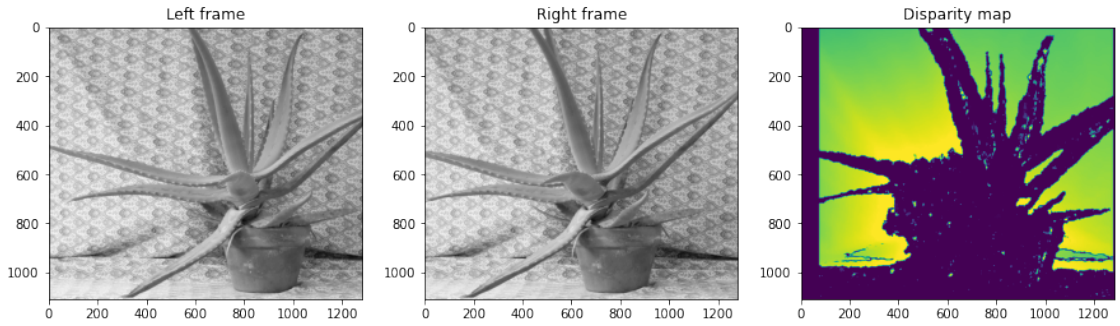
a) Use the function `cv2.StereoBM_create(numDisparities=0, blockSize=21)` (Documentation) (Class Documentation) to computing stereo correspondence using the block matching algorithm.

Find the disparity map between the following images: **./images/aloeL.jpg** and **./image/aloeR.jpg**.

[1]:
```
import cv2
import matplotlib.pyplot as plt
```

[2]:
```
# Read images
imL = cv2.imread('./images/aloeL.jpg', cv2.IMREAD_GRAYSCALE)
imR = cv2.imread('./images/aloeR.jpg', cv2.IMREAD_GRAYSCALE)

# Create the object of the class that will compute the stereo correspondence
stereoBM = cv2.StereoBM_create(numDisparities=0, blockSize=21)

# Compute the disparity map
dispMap = stereoBM.compute(imL, imR)

# Plot the results
plt.figure(figsize=(15,15))
plt.subplot(131)
plt.title('Left frame')
plt.imshow(imL, cmap='gray')
plt.subplot(132)
plt.title('Right frame')
```

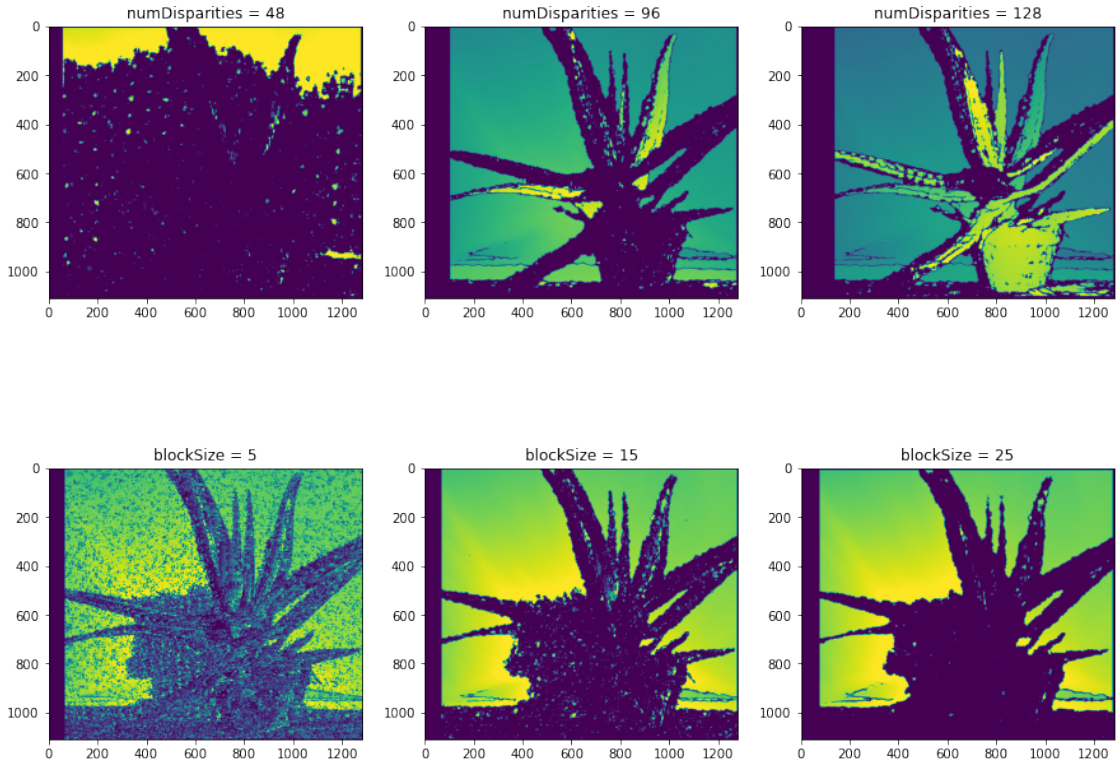
```
plt.imshow(imR, cmap='gray')
plt.subplot(133)
plt.title('Disparity map')
plt.imshow(dispMap)
plt.show()
```



b) What happens if you increase the `numDisparities` parameter in the `cv2.StereoBM_create()`? And if you change the `blockSize` parameter?

[3]:
```
# StereoBM for different disparity numbers
stereoBM_disp1 = cv2.StereoBM_create(numDisparities=48, blockSize=21)
stereoBM_disp2 = cv2.StereoBM_create(numDisparities=96, blockSize=21)
stereoBM_disp3 = cv2.StereoBM_create(numDisparities=128, blockSize=21)

# StereoBM for different blockSize
stereoBM_bsize1 = cv2.StereoBM_create(numDisparities=0, blockSize=5)
stereoBM_bsize2 = cv2.StereoBM_create(numDisparities=0, blockSize=15)
stereoBM_bsize3 = cv2.StereoBM_create(numDisparities=0, blockSize=25)

# Compute the disparity map
dispMap_disp1 = stereoBM_disp1.compute(imL, imR)
dispMap_disp2 = stereoBM_disp2.compute(imL, imR)
dispMap_disp3 = stereoBM_disp3.compute(imL, imR)
dispMap_bsize1 = stereoBM_bsize1.compute(imL, imR)
dispMap_bsize2 = stereoBM_bsize2.compute(imL, imR)
dispMap_bsize3 = stereoBM_bsize3.compute(imL, imR)


# Plot the results
plt.figure(figsize=(15,15))
plt.title('Results with different disparity number')
plt.subplot(131)
plt.title('numDisparities = 48')
plt.imshow(dispMap_disp1)
plt.subplot(132)
```

```
plt.title('numDisparities = 96')
plt.imshow(dispMap_disp2)
plt.subplot(133)
plt.title('numDisparities = 128')
plt.imshow(dispMap_disp3)
plt.show()

plt.figure(figsize=(15,15))
plt.title('Results with different block size')
plt.subplot(131)
plt.title('blockSize = 5')
plt.imshow(dispMap_bsize1)
plt.subplot(132)
plt.title('blockSize = 15')
plt.imshow(dispMap_bsize2)
plt.subplot(133)
plt.title('blockSize = 25')
plt.imshow(dispMap_bsize3)
plt.show()
```



numDisparity: "For each pixel the algorithm will find the best disparity from 0 (default minimum disparity) to numDisparities", this means that the more disparities you have, the more probability you have of finding a better disparity, but also the more inefficient your algorithm is.

blockSize: On the one hand, a smaller block size means we get more details in our disparity map, however, it also means there are more probabilities of getting wrong correspondence. On the other hand the more the block size is, implies we get a smoother, but also less accurate disparity map.

1.3 Problem 3 (Camera calibration)

Calibrate the camera using a set of checkerboard images (you can find them in *./images/left??.jpg*), where `??` indicates the index of the image

Click here for an optional hint

Use the following lines to get the entire list of the images to process:

```
from glob import glob

img_names = glob('./images/left??.jpg')
```

a) Use the checkerboard images to find the feature points using the openCV `cv2.findChessboardCorners()` function (Documentation).

Normally, we have talked about camera calibration as a method to know the intrinsic parameters of the camera, here we want to use the camera matrix and the relative distortion coefficients to undistort the previous images. For a detailed explanation of distortion, read section 13.4.9 of the text book [1].

b) Calibrate the camera using the feature points discovered in **a)** and find the relative camera matrix and distortion coefficients using `cv2.calibrateCamera()` function (Documentation).

P.S.:
By default, you should find 5 distortion coeffiencients (3 radial distortion coeff. (k_1, k_2, k_3) and 2 tangential coeff. (p_1, p_2)); these values are used later to find a new camera matrix and to undistort the images.

c) Using the camera matrix and distortion coefficients, transform the images to compensate any kind of distortion using `cv2.getOptimalNewCameraMatrix()` (Documentation) and `cv2.undistort()` (Documentation).

[4]:
```
# a)
# Function to find the feature points using cv2.findChessboardCorners(...)
# If the function finds the corners, return them, otherwise return None
def findCorners(filename, pattern_size):

    img = cv2.imread(filename)

    found, corners = cv2.findChessboardCorners(img, pattern_size)

    if not found: return None
    return corners.reshape(-1, 2)
```

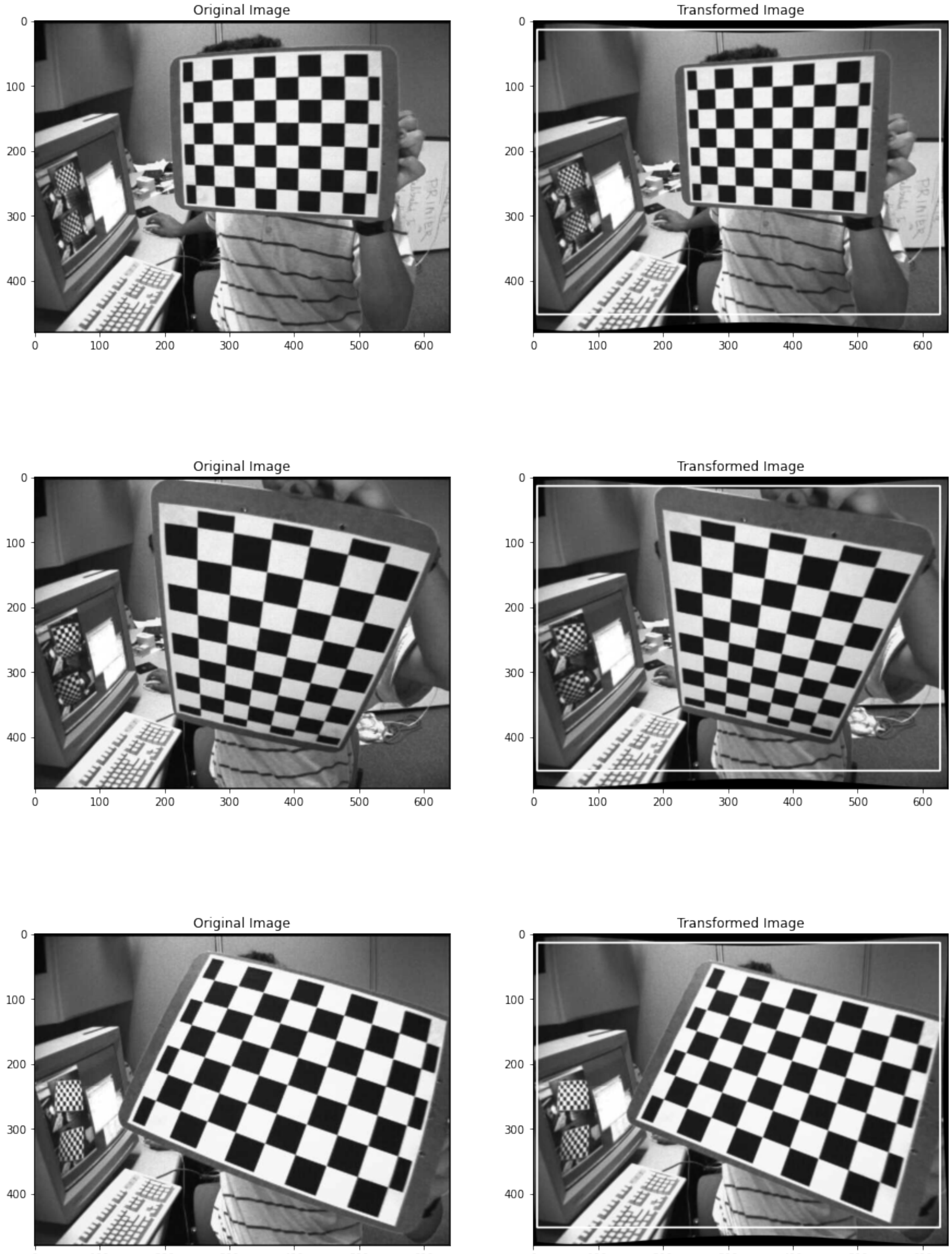
[5]:
```
# b)
# Function to calibrate the camera.
```

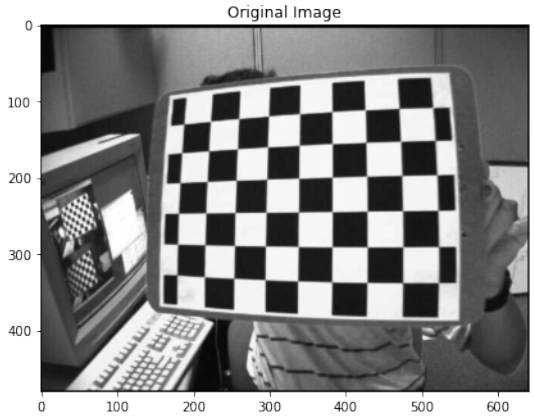
```
# Return the camera matrix and the distortion coeffiecients (radial &␣
↪tangential)
def calibrateTheCamera(obj_points, img_points, img_shape):

    # The function estimates the intrinsic camera parameters and extrinsic␣
↪parameters for each of the views
    _, cameraMatrix, distCoeffs, _, _ = cv2.calibrateCamera(obj_points,␣
↪img_points, img_shape,
                                                            cameraMatrix=None,␣
↪distCoeffs=None)

    return cameraMatrix, distCoeffs
```

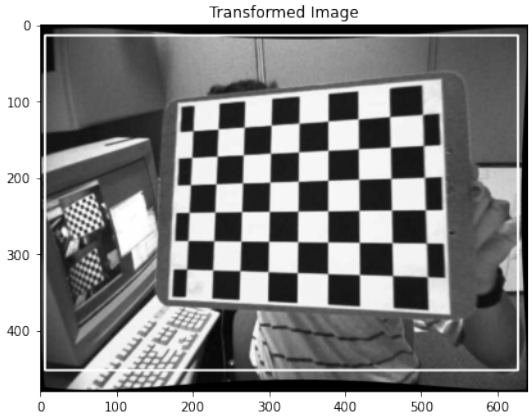
[6]:
```
# c)
# Function that undistort the images using cv2.getOptimalNewCameraMatrix(...)␣
↪and cv2.undistort(...)
# Plot the new undistorted images.
def undistortImage(filename, camera_matrix, dist_coefs):

    img = cv2.imread(filename,0)
    h, w = img.shape[:2]

    # Returns the new camera intrinsic matrix based on the camera matrix and␣
↪the distortion coefficients
    new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(camera_matrix,␣
↪dist_coefs, imageSize=(w, h),
                                                           alpha=1)

    # Transforms an image to compensate for lens distortion using the camera␣
↪matrix,
    # the distortion coefficients and the camera matrix of the distorted image.
    img_transformed = cv2.undistort(img, camera_matrix, dist_coefs, None,␣
↪new_camera_matrix)

    x, y, w, h = roi

    plt.figure(figsize=(15,15))
    plt.subplot(121)
    plt.title('Original Image')
    plt.imshow(img, cmap='gray')
    plt.subplot(122)
    plt.title('Transformed Image')
    plt.imshow(cv2.rectangle(img_transformed, (x,y), (w,h), (255,255,255), 2),␣
↪cmap='gray')
    plt.show()
```

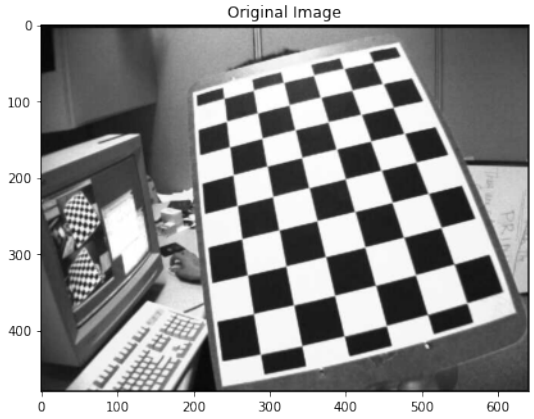
```
    return
```

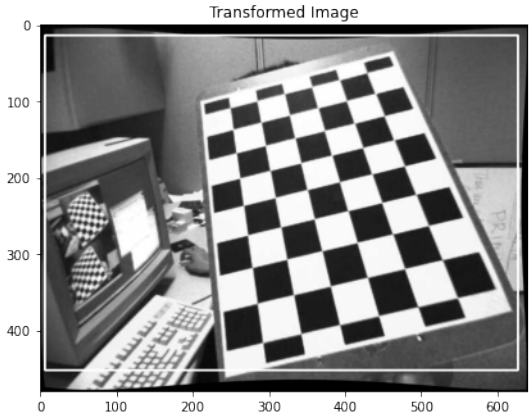
[7]:
```
from glob import glob
import numpy as np
import cv2
import matplotlib.pyplot as plt

obj_points = []
img_points = []
img_names = glob('./images/left??.jpg')


# From the documentation of cv2.findChessboardCorners:
# patternSize - Number of inner corners per a chessboard row and column
#( patternSize = cvSize(points_per_row,points_per_colum) = cvSize(columns,rows)␣
 ↪).
pattern_size = (9,6)

# Defining the world coordinates for 3D points
pattern_points = np.zeros((np.prod(pattern_size), 3), np.float32)
pattern_points[:, :2] = np.indices(pattern_size).T.reshape(-1, 2)
pattern_points *= 1

#### a)
# Find feature points with checkerboard images.
chessboards = [findCorners(filename, pattern_size) for filename in img_names]
for corners in [chessboard for chessboard in chessboards if chessboard is not␣
 ↪None]:
    img_points.append(corners)
    obj_points.append(pattern_points)

#### b)
# Get the camera matrix and the distortion coeffiecients (radial & tangential).
img_shape = cv2.imread(img_names[0], cv2.IMREAD_GRAYSCALE).shape[:2]
camera_matrix, dist_coefs = calibrateTheCamera(obj_points, img_points,␣
 ↪img_shape)

#### c)
# Undistort the images and plot them.
for filename in img_names:
    undistortImage(filename, camera_matrix, dist_coefs)
```

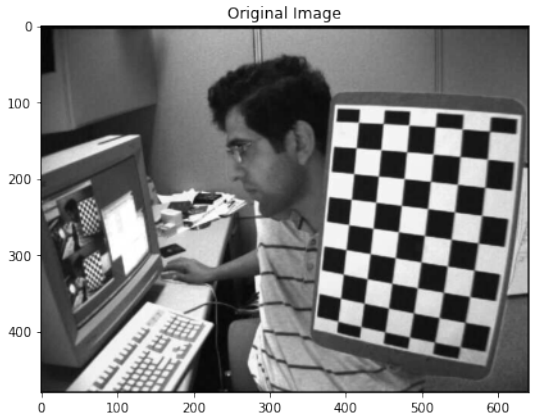

Original Image
Transformed Image
Original Image
Transformed Image
Original Image
Transformed Image

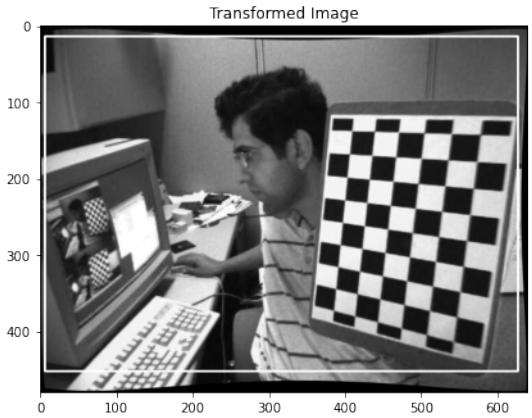

Original Image

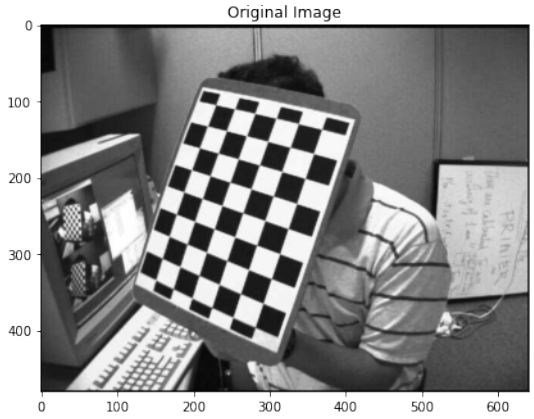

Transformed Image

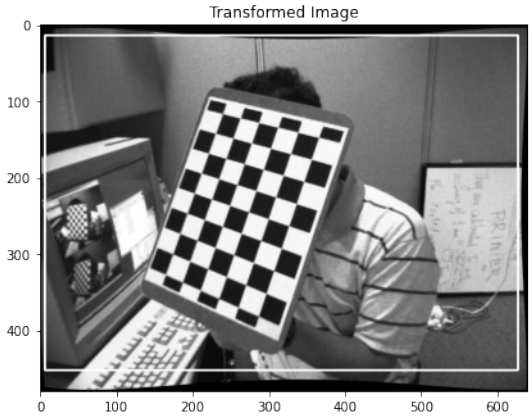

Original Image

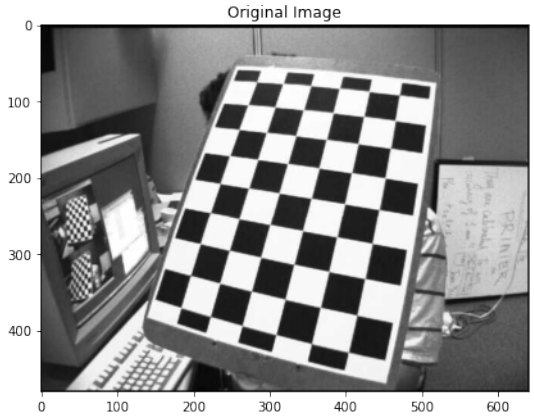

Transformed Image

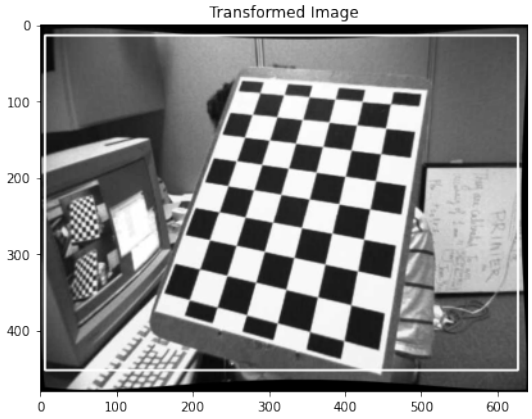

Original Image

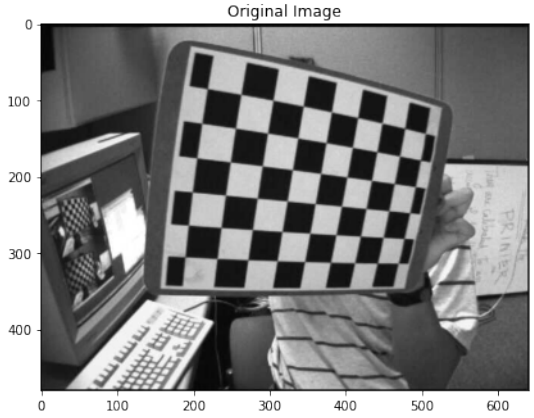

Transformed Image

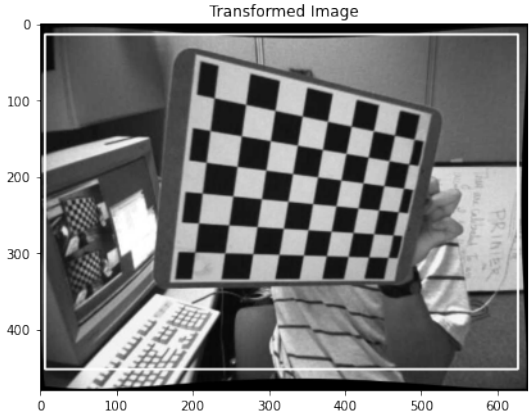

Original Image

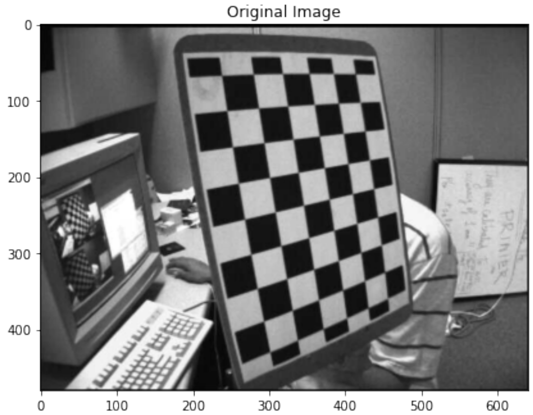
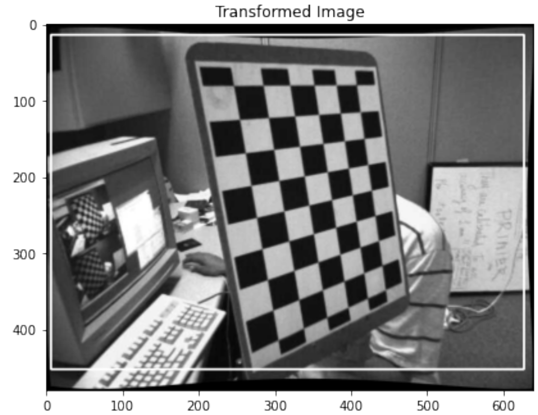
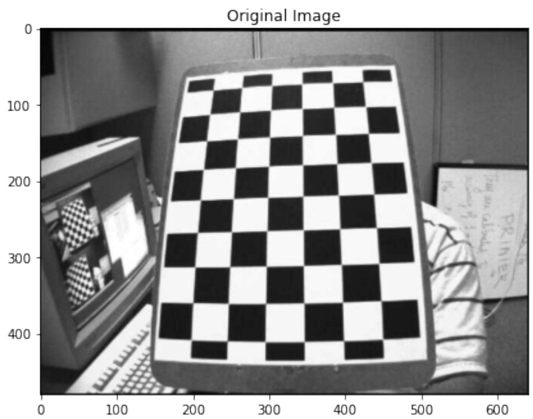
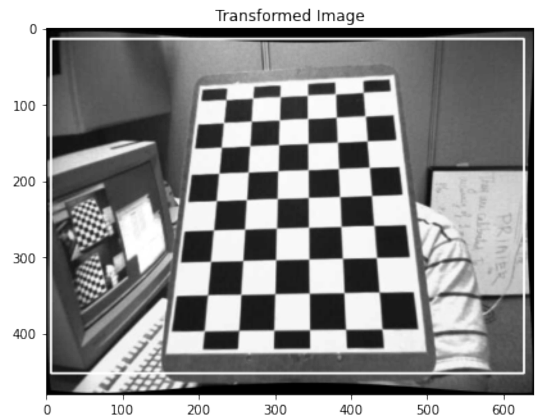
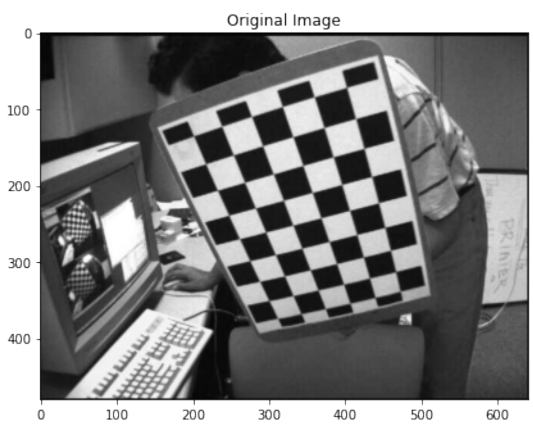
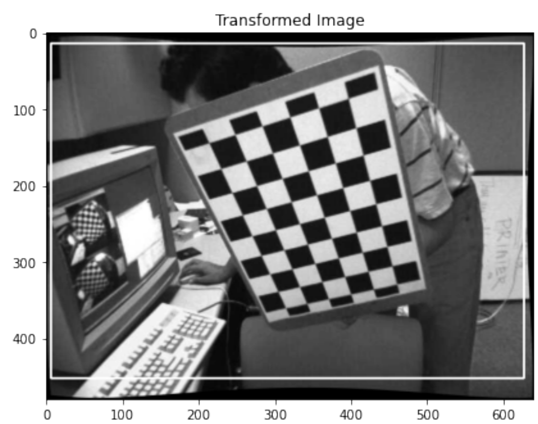

Transformed Image

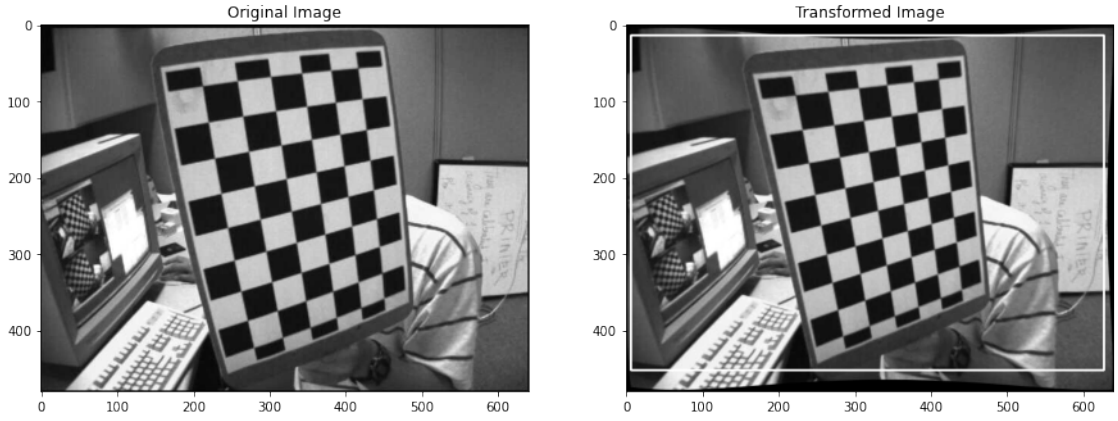

Original Image


Transformed Image


Original Image


Transformed Image


Original Image

Transformed Image

Original Image

Transformed Image

Original Image

Transformed Image



Notice in the transformed image we have marked with a rectangle the part of the image tha is not under effects of distorsion. It's easy to see the differents between both images.

1.3.1 Delivery (dead line) on CANVAS: 24.10.2021 at 23:59

1.4 Contact

1.4.1 Course teacher

Professor Kjersti Engan, room E-431, E-mail: kjersti.engan@uis.no

1.4.2 Teaching assistant

Tomasetti Luca, room E-401 E-mail: luca.tomasetti@uis.no

1.5 References

[1] S. Birchfeld, Image Processing and Analysis. Cengage Learning, 2016.

[2] I. Austvoll, "Machine/robot vision part I," University of Stavanger, 2018. Compendium, CANVAS.