

# Proposed solutions for LABexc7-ELE510-2023

October 20, 2023

## 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 reconstruction by triangulation.*

The theory for this exercise can be found in chapter 13 of the text book [1] and in chapter 4 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:

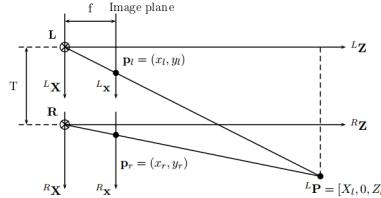
```
![image name]("image_path")  

```

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.  $\mathbf{L}$  and  $\mathbf{R}$  denotes the focal point of the Left and Right camera respectively.  $\mathbf{P}$  is a point in the 3D world, and  $\mathbf{p}$  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  $T = 5\text{ cm}$  and the focal length  $f = 5\text{ cm}$ .

- a) Consider the scene point  ${}^L\mathbf{P}_w = [0.05\text{m}, 0, 10\text{m}]^T$ . Suppose that due to various errors, the image coordinate  $x_l$  is 2% **bigger** than its true value, while the image coordinate  $x_r$  is perfect. What is the error in depth  $z_w$ , in millimeters (round up to three decimals)?
- b) An image of resolution  $500 \times 500$  pixels is seen by the Left and Right cameras. The image sensor size is  $10\text{mm} \times 10\text{mm}$ . Let the disparity in the image coordinates be 25 pixels. Using the same focal point and baseline, what is the depth of the image compare to the cameras?
- c) Can you explain with your own words the stereo ordering constraint? What is the definition of forbidden zone in this scenario?

We have

$$\begin{aligned} x_l &= f \cdot \frac{{}^Lx_w}{{}^Lz_w} = 0.05\text{m} \cdot \frac{0.05\text{m}}{10\text{m}} = 0.00025\text{m} \\ x_r &= f \cdot \frac{{}^Lx_w - T}{{}^Lz_w} = 0.05\text{m} \cdot \frac{0.05\text{m} - 0.05\text{m}}{10\text{m}} = 0 \end{aligned}$$

- a) An error of +2% in  $x_l$  gives:

$$x_{l,error} = 0.00025\text{m} \cdot 1.02 = 0.000255\text{m}$$

${}^Lz_{w,error}$  is then given by:

$$\begin{aligned} {}^Lz_{w,error} &= {}^Lx_w \cdot \frac{f}{x_l} \\ &= 0.05\text{m} \cdot \frac{0.05\text{m}}{0.000255\text{m}} = 9.804\text{m} \end{aligned}$$

The error is then:

$$\Delta {}^Lz_{w,error} = 0.196\text{m} = 196\text{mm}$$

b)

Transform the  $d$  from pixel to mm:

$$\begin{aligned} d &= \frac{25 \text{ pixel}}{\frac{500 \text{ pixel}}{10\text{mm}}} \\ d &= \frac{25 \text{ pixel}}{50 \text{ pixel/mm}} = 0.5\text{mm} \end{aligned}$$

Then, you use the disparity to find the depth ( $Z$ ):

$$Z = \frac{fT}{d} = \frac{50\text{mm} \cdot 50\text{mm}}{0.5\text{mm}} = 5000\text{mm} = 5\text{m}$$

c) When a point in a continuous surface is viewed by both cameras, it is not physically possible for another point on the same surface to also be visible in both cameras if it lies within the region defined by two lines passing through the centers of projection and the point. This hourglass-shaped region is called the forbidden zone, and denying matches in the forbidden zone is known as the ordering constraint.

## 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 compute stereo correspondence using the block matching algorithm.

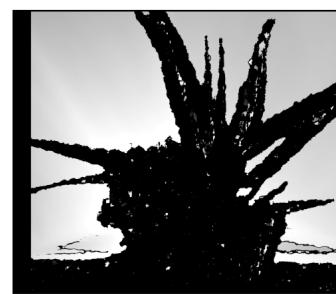
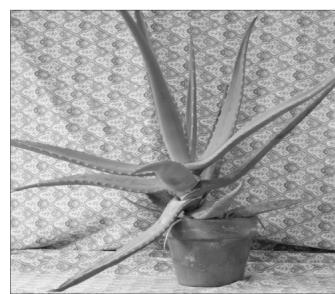
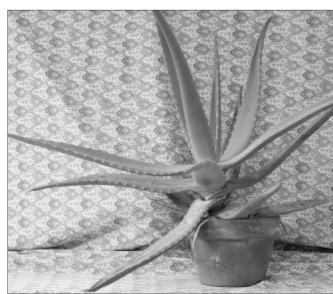
Find the disparity map between the following images: `./images/aloeL.jpg` and `./image/aloeR.jpg`.

```
[1]: import cv2, os
import numpy as np
import matplotlib.pyplot as plt

left = cv2.imread(os.path.join(os.getcwd(), 'images/aloeL.jpg'), cv2.
    ↪IMREAD_GRAYSCALE)
right = cv2.imread(os.path.join(os.getcwd(), 'images/aloeR.jpg'), cv2.
    ↪IMREAD_GRAYSCALE)

stereo = cv2.StereoBM_create(numDisparities=0, blockSize=21)
disparity = stereo.compute(left,right)

plt.figure(figsize=(30,30))
plt.subplot(131), plt.imshow(left, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.subplot(132), plt.imshow(right, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.imshow(disparity,'gray'), plt.xticks([]), plt.yticks([])
plt.show()
```

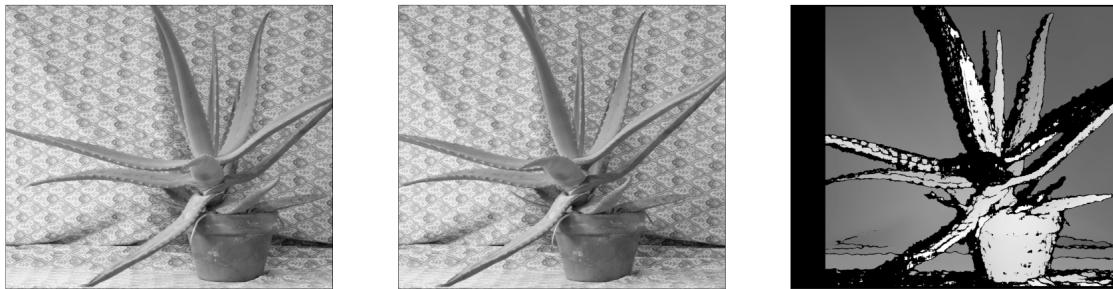


b) What happens if you increase the `numDisparities` parameter in the `cv2.StereoBM_create()`? And if you change the `blockSize` parameter?

Increasing the `numDisparities` parameter, the algorithm will search in a broader range of disparities, which increases the computation time significantly.

```
[2]: stereo = cv2.StereoBM_create(numDisparities=128, blockSize=21)
disparity = stereo.compute(left,right)

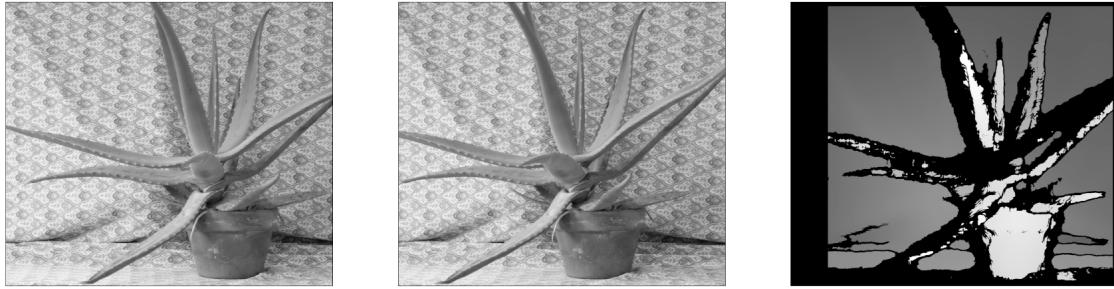
plt.figure(figsize=(30,30))
plt.subplot(131), plt.imshow(left, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.subplot(132), plt.imshow(right, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.imshow(disparity,'gray'), plt.xticks([]), plt.yticks([])
plt.show()
```



If you increase the `blockSize` parameter implies smoother, though less accurate disparity map. On the other hand, if you decrease it, the block size gives more detailed disparity map, but there is higher chance for algorithm to find a wrong correspondence.

```
[3]: stereo = cv2.StereoBM_create(numDisparities=128, blockSize=41)
disparity = stereo.compute(left,right)

plt.figure(figsize=(30,30))
plt.subplot(131), plt.imshow(left, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.subplot(132), plt.imshow(right, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.imshow(disparity,'gray'), plt.xticks([]), plt.yticks([])
plt.show()
```

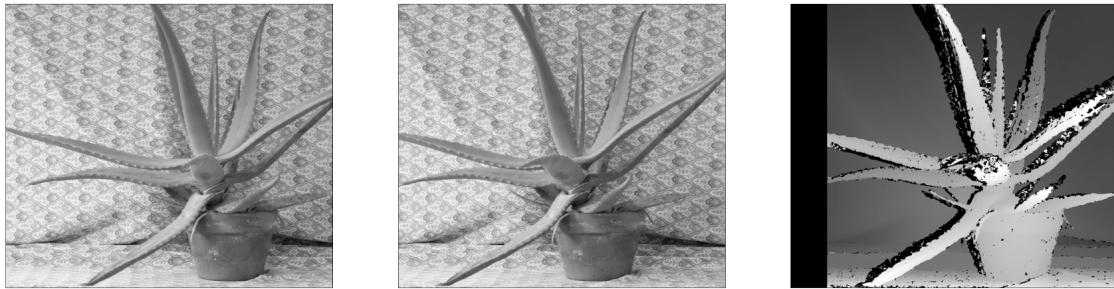


An alternative to the classic BM algorithm is to use the modified version developed by Heiko Hirschmuller. Article: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4359315>

The function is given by OpenCV: `cv2.StereoSGBM_create()` ([Documentation](#))

```
[4]: stereo = cv2.StereoSGBM_create(minDisparity=16, numDisparities=128, blockSize=11)
disparity = stereo.compute(left,right)

plt.figure(figsize=(30,30))
plt.subplot(131), plt.imshow(left, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.subplot(132), plt.imshow(right, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.imshow(disparity, 'gray'), plt.xticks([]), plt.yticks([])
plt.show()
```



### 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

**Hint:** use `glob('~/images/left???.jpg')` to get the entire list of the images to process.

- 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 coefficients (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](#)).

```
[5]: # 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, cv2.IMREAD_GRAYSCALE)

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

    if not found: return None

    return corners.reshape(-1, 2)
```

```
[6]: # b)
# Function to calibrate the camera.
# Return the camera matrix and the distortion coefficients (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
    rms, camera_matrix, dist_coefs, _, _ = cv2.calibrateCamera(obj_points,
                                                               img_points, img_shape, cameraMatrix=None, distCoeffs=None)

    print("overall RMS re-projection error:", rms)
    print("camera matrix:\n", camera_matrix)
    print("distortion coefficients: ", dist_coefs.ravel())

    return camera_matrix, dist_coefs
```

```
[7]: # 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
    newcameramtx, roi = cv2.
    ↪getOptimalNewCameraMatrix(cameraMatrix=camera_matrix,
                                distCoeffs=dist_coefs,
                                imageSize=(w, h),
                                alpha=1,
                                newImgSize=(w, h))

    # Transforms an image to compensate for lens distortion using the camera
    # matrix,
    # the distortion coefficients and the camera matrix of the distorted image.
    dst = cv2.undistort(src=img,
                        cameraMatrix=camera_matrix,
                        distCoeffs=dist_coefs,
                        dst=None,
                        newCameraMatrix=newcameramtx)

    # roi = output rectangle that outlines all-good-pixels region in the
    # undistorted image.
    x, y, w, h = roi

    plt.figure(figsize=(30,30))
    plt.subplot(121),plt.imshow(cv2.rectangle(dst,(x,y),(w,h),(255,0,0),2),
    ↪"gray"), plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(img, "gray"), plt.xticks([]), plt.yticks([])
```

```
[8]: from glob import glob

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_col) = 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)

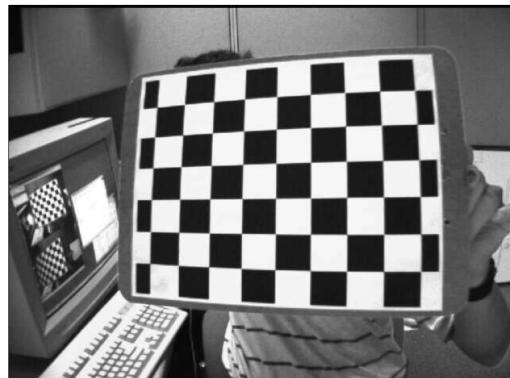
#### b)
# Get the camera matrix and the distortion coefficients (radial & tangential).
img_shape = cv2.imread(img_names[0], cv2.IMREAD_GRAYSCALE).shape[:2]
camera_matrix, dist_coefs = calibrateTheCamera(obj_points, img_points, img_shape)

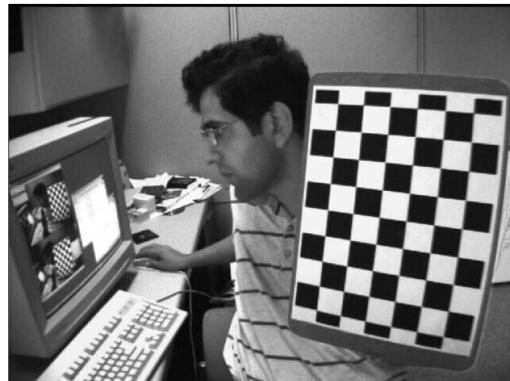
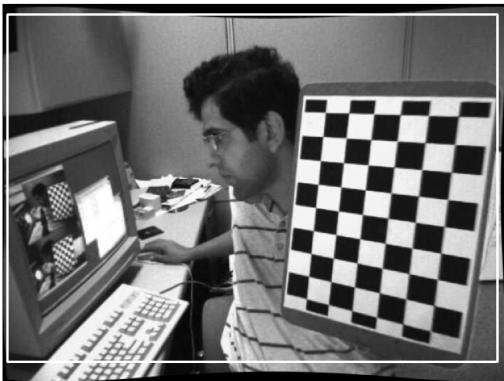
#### c)
# Undistort the images and plot them.
for filename in img_names:
    undistortImage(filename, camera_matrix, dist_coefs)

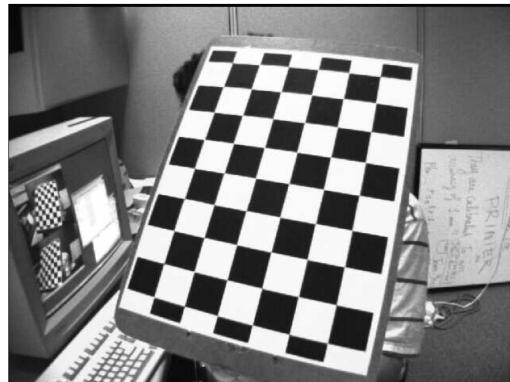
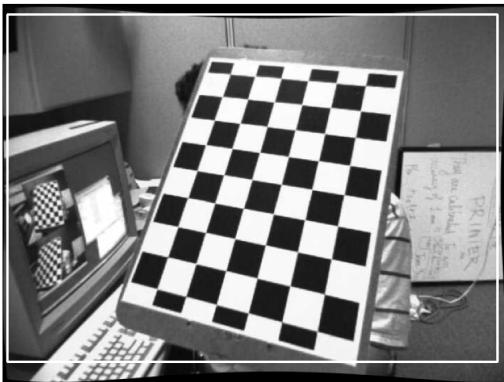
```

overall RMS re-projection error: 0.38117154733679093  
camera matrix:  
[[531.14749874 0. 341.82112771]  
 [ 0. 531.4320961 235.03969878]  
 [ 0. 0. 1. ]]  
distortion coefficients: [-2.74391331e-01 -3.07977600e-02 8.95663893e-04  
-2.21525979e-04  
2.76812366e-01]











## 1.4 Contact

### 1.4.1 Course teacher

Professor Kjersti Engan, room E-431, E-mail: [kjersti.engan@uis.no](mailto:kjersti.engan@uis.no)

#### **1.4.2 Teaching assistant**

Saul Fuster Navarro, room E-401 E-mail: saul.fusternavarro@uis.no

Jorge Garcia Torres Fernandez, room E-401 E-mail: jorge.garcia-torres@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.