Proposed solutions for LABexc2-ELE510-2023

September 11, 2023

1 ELE510 Image Processing with robot vision: LAB, Exercise 2, Image Formation.

Purpose: *To learn about the image formation process, i.e. how images are projected from the scene to the image plane.*

The theory for this exercise can be found in chapter 2 and 3 of the text book [1]. Supplementary information can found in chapter 1, 2 and 3 in the compendium [2]. See also the following documentations for help: - OpenCV - numpy - matplotlib

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 frst. 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

a) What is the meaning of the abbreviation PSF? What does the PSF specify?

Point Spread Function. It specifies the shape that a point will take on the image plane.

b) Use the imaging model shown in Figure 1. The camera has a lens with focal length $f = 40$mm and in the image plane a CCD sensor of size 10mm $\times$ 10mm. The total number of pixels is 5000×5000.

At a distance of $z_w = 0.5$m from the camera center, what will be the camera's resolution in pixels per millimeter?

Figure 1: Perspective projection caused by a pinhole camera. Figure 2.23 in [2].

We can try to find the spatial coordinates (x_w, y_w) corresponding to the height and width of the physical area covered by the image plane at depth $z_w = 1$m. Thereafter, find how many lines (pixels) per mm the camera will resolve at this depth:

$$x = f\frac{x_w}{z_w} \rightarrow x_w = x\frac{z_w}{f} = 10\frac{500}{40} = 125mm \tag{1}$$

$$y = f\frac{y_w}{z_w} \rightarrow y_w = x\frac{z_w}{f} = 10\frac{500}{40} = 125mm \tag{2}$$

$$r_x = r_y = \frac{5000}{125} = 40 \text{ pixels per mm at a distance of 1 m from the camera} \tag{3}$$

Alternative, we can find the distance between the world points D and between pixels d, and use similar triangles to get the answer:

$$\frac{D}{z_w} = \frac{d}{f} \rightarrow D = z_w\frac{d}{f} \tag{4}$$

$$d = \frac{10mm}{5000pixels} = 2\mu m/pixel \tag{5}$$

$$D = 0.5m\frac{2\mu m/pixel}{40mm} = \frac{1}{40}mm/pixel \rightarrow \text{This means 40 pixels per mm at a distance of 1 m from the camera} \tag{6}$$

c) Explain how a Bayer filter works. What is the alternative to using this type of filter in image acquisition?

One costefficient way to produce a colorimage is to let each sensor (at the pixel location) capturing either the green, the red or the blue spectrum. A Bayer filter mosaic is a colorfilter array for arranging RGB color filters on a square grid of photosensors (CCD or CMOS).

A Bayer filter blocks all but the green light for alternating pixels throughout the sensor in a checkerboard pattern. Green is then sensed by half the pixels, with the remaining pixels sensing blue or red in alternating rows. The raw output is a Bayern pattern image, where each pixel only has one color. A demosaicing method is used to interpolate so that the resulting image has RGB at all pixel poisitions.

d) Briefly explain the following concepts: Sampling, Quantization, Gamma Compression.

Sampling is the process of converting the continuous irradiance function into a discrete function defined only over the rectangular lattice of integer (x, y) coordinates.

Quantization is the process of assigning a discrete gray level to every pixel in order to represent its value in digital form.

Gamma Compression is the process of transforming the linear, physical light intensity into a perceptually uniform quantity, so that the pixel values in a digital image are not directly proportional to the amount of light collected by the sensor.

1.2 Problem 2

Assume we have captured an image with a digital camera. The image covers an area in the scene of size 1.024m $\times$ 0.768m (the camera has been pointed towards a wall such that the distance is approximately constant over the whole image plane, *weak perspective*). The camera has 4096 pixels horizontally, and 3072 pixels vertically. The active region on the CCD-chip is 8mm $\times$ 6mm. We define the spatial coordinates (x_w, y_w) such that the origin is at the center of the optical axis, x-axis horizontally and y-axis vertically upwards. The image indexes (x, y) is starting in the upper left corner. The solutions to this problem can be found from simple geometric considerations. Make a sketch of the situation and answer the following questions:

a) What is the size of each sensor (one pixel) on the CCD-chip?

b) What is the scaling coefficient between the image plane (CCD-chip) and the scene? What is the scaling coefficient between the scene coordinates and the pixels of the image?

1.2.1 a)

If the total number of pixels is 4096×3072, the size of each pixel for a 0.008×0.006m action region on the chip is computed as:

$$d_x = \frac{0.008 \text{ m}}{4096 \text{ pixels}} \approx 1.95\mu\text{m / pixel} \tag{7}$$

$$d_y = \frac{0.006 \text{ m}}{3072 \text{ pixels}} \approx 1.95\mu\text{m / pixel} \tag{8}$$

1.2.2 b)

The CCD-chip is represented by the (x, y) coordinates and the scene by the camera coordinates (x_w, y_w). Even without the knowledge of f we can use the information from the Field of View (FOV) to find the scaling factors.

$$x = a_x x_w \rightarrow a_x = \frac{x}{x_w} = \frac{0.008}{1.024} \approx 0.0078 \tag{9}$$

$$y = a_y y_w \rightarrow a_y = \frac{y}{y_w} = \frac{0.006}{0.768} \approx 0.0078 \tag{10}$$

The scaling factors between pixels and the scene coordinates are:

$$s_x = s_y = \frac{a_y}{d_y} = \frac{a_x}{d_x} = \frac{0.0078}{0.00195 \times 10^{-3}} = 4000 \text{ pixels per m} \tag{11}$$

1.3 Problem 3

Translation from the scene to a camera sensor can be done using a transformation matrix, T.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = T \begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix} \tag{12}$$

where

$$T = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{13}$$

$ _x$ and α_y are the scaling factors for their corresponding axes.

Write a function in Python that computes the image points using the transformation matrix, using the parameters from Problem 2. Let the input to the function be a set of K scene points, given by a $2 \times K$ matrix, and the output the resulting image points also given by a $2 \times K$ matrix. The parameters defining the image sensor and field of view from the camera center to the wall can also be given as input parameters. For simplicity, let the optical axis (x_0, y_0) meet the image plane at the middle point (in pixels).

Test the function for the following input points given as a matrix:

$$\mathbf{P}_{in} = \begin{bmatrix} 0.512 & -0.512 & -0.512 & 0.512 & 0 & 0.35 & 0.35 & 0.3 & 0.7 \\ 0.384 & 0.384 & -0.384 & -0.384 & 0 & 0.15 & -0.15 & -0.5 & 0 \end{bmatrix} \tag{14}$$

Comment on the results, especially notice the two last points!

[1]:
```
# Import the packages that are useful inside the definition of the␣
 ↪weakPerspective function
import math
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
"""
Function that takes in input:
- FOV: field of view,
- sensorsize: size of the sensor,
- n_pixels: camera pixels,
- p_scene: K input points (2xK matrix)

and return the resulting image points given the 2xK matrix
"""
def weakPerspective(FOV, sensorsize, n_pixels, p_scene):

    x_0 = np.ceil(n_pixels[0]/2)
    y_0 = np.ceil(n_pixels[1]/2)

    # Scaling factor (between image plane (CCD-chip) and scene)
    a = np.array(sensorsize)/np.array(FOV)

    # Pixel dimension
    d = np.array(sensorsize)/np.array(n_pixels)

    # Scaling factor (between scene coordinates and image indexes)
    s = a/d
```

```
    T = np.array([[s[0], 0, x_0], [0, s[1], y_0], [0, 0, 1]])

    K = p_scene.shape[1]

    # Just take the 2 first row from the multiplication
    return np.matmul(T, np.concatenate((p_scene, np.ones((1,K))), axis=0))[0:2]
```

[6]:
```
# The above function is then called using the following parameters:

# Parameters
FOV = [1.024, 0.768]
sensorsize = [0.008, 0.006]
n_pixels = [4096, 3072]
p_scene_x = [0.512, -0.512, -0.512, 0.512, 0, 0.35, 0.35, 0.3, 0.7]
p_scene_y = [0.384, 0.384, -0.384, -0.384, 0, 0.15, -0.15, -0.5, 0]
```

[7]:
```
####
# This cell is locked; it can be only be executed to see the results.
####
# Input data:
p_scene = np.array([p_scene_x, p_scene_y])

# Call to the weakPerspective() function
pimage = weakPerspective(FOV, sensorsize, n_pixels, p_scene)

# Result:
print(pimage)
```

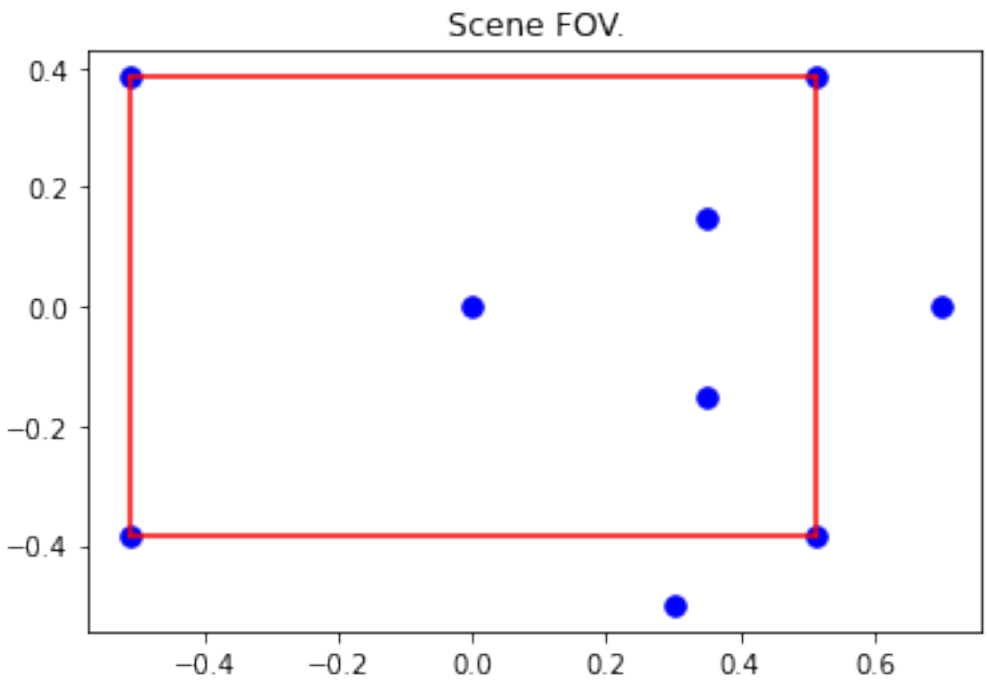
```
[[4096.    0.    0. 4096. 2048. 3448. 3448. 3248. 4848.]
 [3072. 3072.    0.    0. 1536. 2136.  936. -464. 1536.]]
```

The first 5 points corresponds to the 4 corners and the center pixel (x0,y0) and are in accordance with the results from problem 2. The two next points are within the FOV (Field of View), (2136,3448) and (936,3448), (both in column 3448). The two last points are outside the FOV, give image points outside the image sensor, and will not be seen in the image

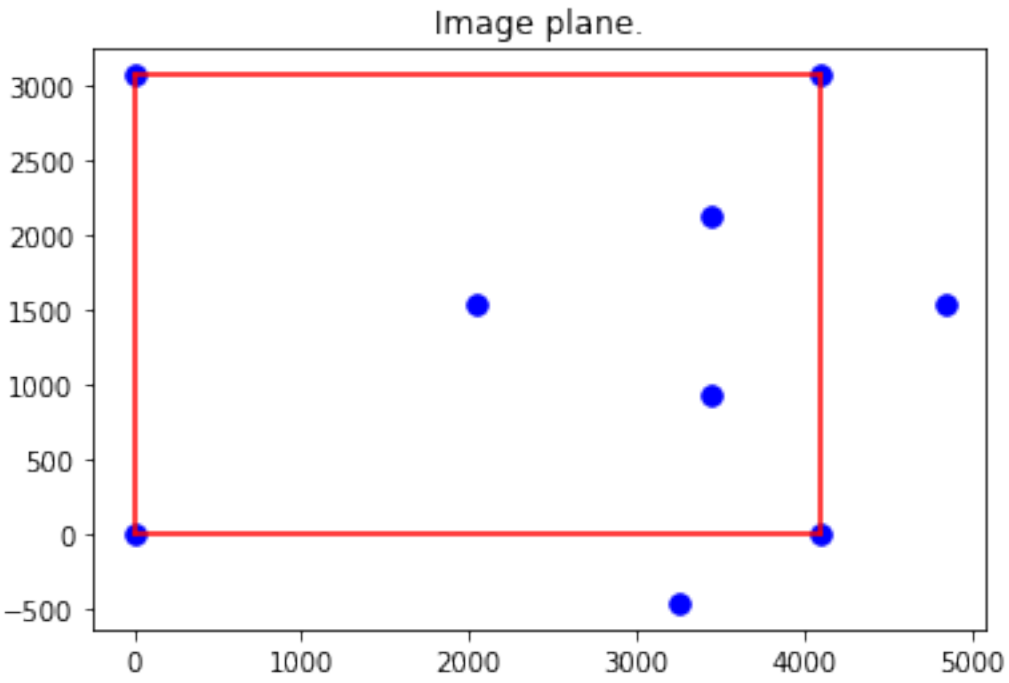
[ ]:
```
print(
"""\n
The FOV of the scene is the red rectangle while the input points are in blue.
The four corners of the FOV are (clockwise, from top-left):
(-0.512,0.384), (0.512,0.384), (0.512,-0.384), (-0.512,-0.384).
"""
)

plt.title("Scene FOV.")
```

```
plt.plot(p_scene_x, p_scene_y, marker='.', linestyle='None', markersize=15,␣
 ↪color='blue')
plt.plot([FOV[0]/2, FOV[0]/2], [FOV[1]/2,-FOV[1]/2], color='red')
plt.plot([FOV[0]/2,-FOV[0]/2], [-FOV[1]/2,-FOV[1]/2], color='red')
plt.plot([-FOV[0]/2,-FOV[0]/2], [-FOV[1]/2,FOV[1]/2], color='red')
plt.plot([FOV[0]/2,-FOV[0]/2], [FOV[1]/2,FOV[1]/2], color='red')
plt.show()

print("""
And the resulting scene image coordinates points (pimage) can be seen in the␣
 ↪same way
inside the image plane.
The four corners of the image plane are (clockwise, from top-left):
(0,3072), (4096,3072), (4096,0), (0,0)
""")

plt.title("Image plane.")
plt.plot(pimage[0], pimage[1], marker='.', linestyle='None', markersize=15,␣
 ↪color='blue')
plt.plot([0, n_pixels[0]], [0,0], color='red')
plt.plot([n_pixels[0], n_pixels[0]], [0, n_pixels[1]], color='red')
plt.plot([0, n_pixels[0]], [n_pixels[1], n_pixels[1]], color='red')
plt.plot([0,0], [0, n_pixels[1]], color='red')
plt.show()
```

The FOV of the scene is the red rectangle while the input points are in blue.
The four corners of the FOV are (cloackwise, from top-left):
(-0.512,0.384), (0.512,0.384), (0.512,-0.384), (-0.512,-0.384).



And the resulting scene image coordinates points (pimage) can be seen in the same way
inside the image plane.
The four corners of the image plane are (cloackwise, from top-left):
(0,3072), (4096,3072), (4096,0), (0,0)



1.3.1 Delivery (dead line) on CANVAS: 15-09-2023 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

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.