Proposed solutions for LABexc2-ELE510-2021

September 20, 2021

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?

Proposed answer (a): The point spread function (PSF) describes the response of an imaging system to a point source. This is more generally known as a systems impulse response. The Point spread function (PSF) specifies the shape that a point will take on the image pane.

The term "impulse response" is common in signal processing, and if the system is a filter (rather than the imaging system).

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 $8\text{mm} \times 8\text{mm}$. The total number of pixels is 4000×4000. How many lines per mm will this camera resolve at a distance of $z_w = 1$m from the camera center?

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

Proposed answer (b): The answer is found by using similar triangles and finding the distance between the world points corresponding to the distance between pixels.

$$d = \frac{8mm}{4000pixels} = 2\mu m/pixel(line)$$

$$\frac{D}{1m} = \frac{d}{f} = \frac{d}{40mm}$$

Which gives

$$D = \frac{2 \cdot 10^{-6}}{40 \cdot 10^{-3}} \cdot 1m = \frac{1}{20} mm/pixel(line)$$

This means 20 lines per mm

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

Proposed answer (c): 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 alternative to using this type of filter is to use three sensors to capture the image, one for each color channel.

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 × 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 2048 pixels horizontally, and 1536 pixels vertically. The active region on the CCD-chip is* $10mm \times 7.5mm$. *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. For simplicity let the optical axis meet the image plane at* $(x_0 = 1024, y_0 = 768)$. *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?*

Proposed answer (a): *The size of each pixel is computed as,*

$$d_x = \frac{10mm}{2048pixel} \cdot 1pixel = 4.9\mu m$$
$$d_y = \frac{7.5mm}{1536pixel} \cdot 1pixel = 4.9\mu m$$

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 image indexes?*

Proposed answer (b): *The CCD-chip is represented by the x-y-coordinates and the scene by the camera coordinates X-Y. Even without the knowledge of f we can use the information from the Field of View (FOV) to find the scaling factors.*

$$s_x = \frac{x}{X} = \frac{10 \cdot 10^{-3}m}{1.024m} = 9.7656 \cdot 10^{-3}$$
$$s_y = \frac{y}{Y} = \frac{7.5 \cdot 10^{-3}m}{0.768m} = 9.7656 \cdot 10^{-3}$$

The scaling factors between pixels and the scene coordinates are:

$$s = \frac{s_x}{d_x} = \frac{s_y}{d_y} = \frac{10 \cdot 10^{-3}m}{1.024m} \cdot \frac{2048pixel}{10 \cdot 10^{-3}m} = 2000\, pixels\ per m$$

Thus, the scaling factor is $\alpha_x = \alpha_y = 2000$ *if you calculate it in meters. Or, if you use millimeters:*

$$s = \frac{s_x}{d_x} = \frac{s_y}{d_y} = \frac{10mm}{1024mm} \cdot \frac{2048pixel}{10mm} = 2\, pixels\ per mm$$

Thus, the scaling factor is $\alpha_x = \alpha_y = 2$.

1.3 Problem 3

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

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

where

$$T = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

α_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.*

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

$$\mathbf{P}_{in} = \begin{bmatrix} 0.512 & -0.512 & -0.512 & 0.512 & 0 & 0.3 & 0.3 & 0.3 & 0.6 \\ 0.384 & 0.384 & -0.384 & -0.384 & 0 & 0.2 & -0.2 & -0.4 & 0 \end{bmatrix}. \tag{3}$$

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):
    W, H = FOV
    w, h = sensorsize
    N, M = n_pixels

    dx = w/N; dy = h/M # pixel size
    sx = w/W; sy = h/H # scaling factor, metric units
    s1 = sx/dx; s2 = sy/dy # scaling factor for pixels
    x0 = math.ceil(N/2); y0 = math.ceil(M/2) # center pixel

    # Then we get the transformation matrix
    T = np.array([[s1, 0, x0], [0, s2, y0], [0, 0, 1]])

    K = p_scene.shape[1] # number of scene points
    P3 = np.ones((1,K))
    P = np.append(p_scene,P3,axis=0) # homogeneous coordinates

    p=np.matmul(T,P)
    pimage = p[0:2, :] # the third row of the homogeneous coordinates are␣
↪removed
    return pimage
```

[3]:
```
# The above function is then called using the following parameters:

# Parameters
```

```
FOV = (1.024, 0.768)
sensorsize = (10e-3, 7.5e-3)
n_pixels = (2048, 1536)
p_scene_x = [0.512, -0.512, -0.512, 0.512, 0, 0.3, 0.3, 0.3, 0.6]
p_scene_y = [0.384, 0.384, -0.384, -0.384, 0, 0.2, -0.2, -0.4, 0]
```

[4]:
```
# Input data:
p_scene = np.array([p_scene_x, p_scene_y])

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

print(pimage)
```

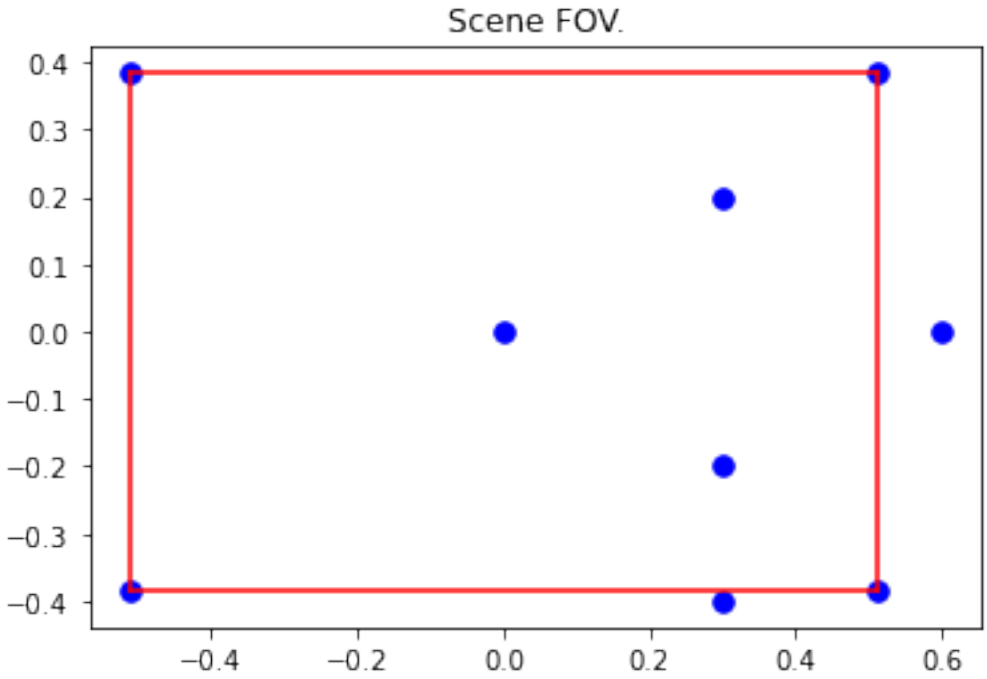
```
[[2048.    0.    0. 2048. 1024. 1624. 1624. 1624. 2224.]
 [1536. 1536.    0.    0.  768. 1168.  368.  -32.  768.]]
```

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), (1168,1624) and (368,1624), (both in column 1624). The two last points are outside the FOV, give image points outside the image sensor, and will not be seen in the image

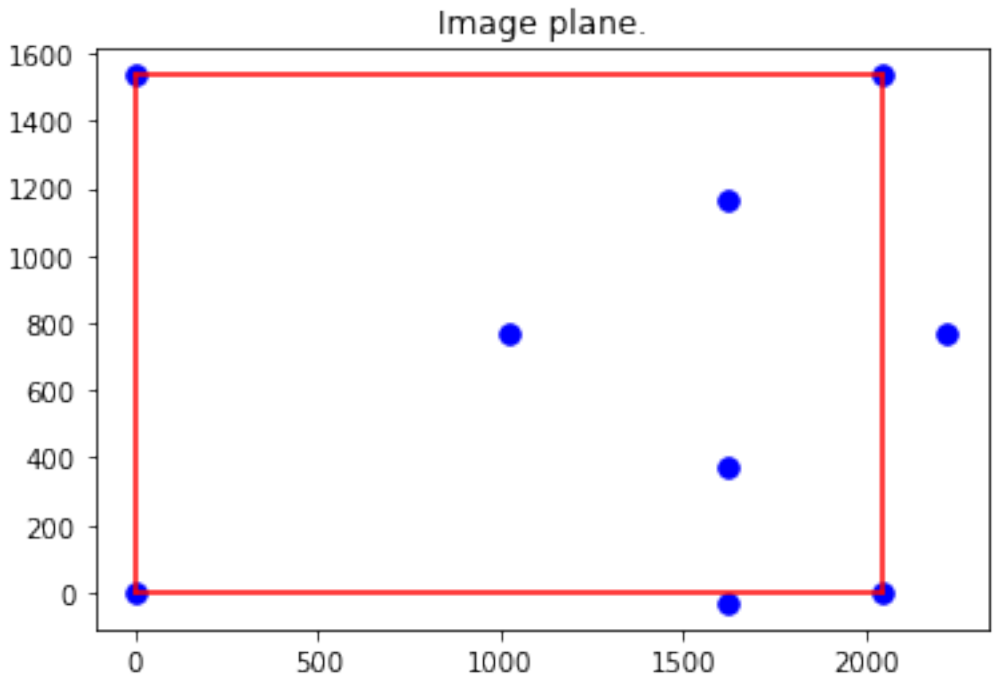
[5]:
```
print(
"""\n
The FOV of the scene is the red rectangle while the input points are in blue.
The four corners of the FOV are (cloackwise, from top-left):
(-0.512,0.384), (0.512,0.384), (0.512,-0.384), (-0.512,-0.384).
"""
)

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 (cloackwise, from top-left):
(0,1536), (2048,1536), (2048,0), (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):
(0,1536), (2048,1536), (2048,0), (0,0)



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.