LABexc2-ELE510-2021

September 8, 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?
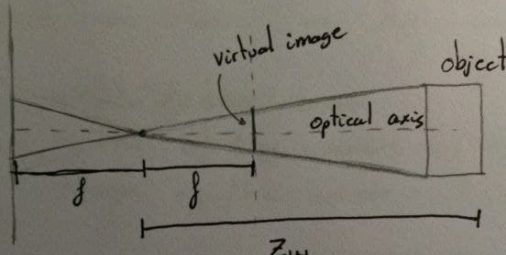
PSF means Point Spread Function and it specifies the shape that a point will take on the image plane.

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 8mm $\times$ 8mm. 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].
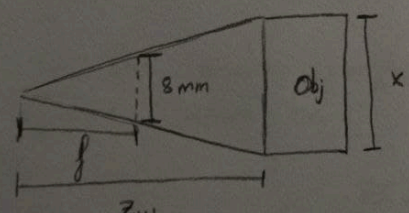
Solution to the exercise:

b)

virtual image
object
optical axis
Sensor size = 8mm × 8mm
$f = 40$mm
Pixels = 4000 × 4000
$Zw = 1m$
Zw

How many lines/mm will the camera resolve?

Using simple geometry we can obtain the size of the object

8mm
Obj
x
Sensor height
Zw

Using triangle similarity we know that

$$\frac{8mm}{x} = \frac{40mm}{1000mm} \Rightarrow X = 200mm$$

$$\left(\frac{f}{Zw}\right)$$

The camera has 4000 rows of pixels (or lines of pixels) and the object height is 200mm, that means we have 20 lines per mm:

$$\frac{4000 \, pixels}{200 \, mm} = \boxed{20 \, lines/mm}$$

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

The bayer color filter is formed by 3 raws, each has a primary color, but with in-between pixels missing, to obtain the full color information we use debayering algorithms which interpolates the missing information from neighboring pixels and provide estimation.

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 × 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?

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?

Solution to the section a)

a) Image size = 1,024 m × 0,768m

Camera resolution = 2048 × 1536

Active region on the chip = 10mm × 7,5mm

Knowing that the active region on the chip is 10mm × 7,5mm and we have 2048 × 1536 pixels on the camera...

width of the sensor = $\dfrac{10mm}{2048\,pixels} \approx 4,88\mu m$

Height of the sensor = $\dfrac{7,5m}{1536\,pixels} \approx 4,88\mu m$

Each sensor is $\boxed{4,88\mu m \times 4,88\mu m}$

Solution to the section b)

b) We need the scaling coefficient of each axis between the scene and the image plane:

$$\alpha_x = \frac{10mm}{1024mm} = \boxed{9,76 \cdot 10^{-3}}$$

$$\alpha_y = \frac{7,5mm}{768mm} = \boxed{9,76 \cdot 10^{-3}}$$

But the scaling coefficient between the scene coordinates and the image indexes (pixels) is:

$$\alpha_x = \frac{9,76 \cdot 10^{-3}}{4,88 \cdot 10^{-3} mm} = \boxed{2}$$

$$\alpha_y = \frac{9,76 \cdot 10^{-3}}{4,88 \cdot 10^{-3} mm} = \boxed{2}$$

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!

4

```
[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
```

```
[23]: """
      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):

          # Get an array of the same size of p_scene
          p_out = np.zeros(np.shape(p_scene))

          ## Get the scaling coefficient for the x axis
          width = sensorsize*n_pixels[0]    # Width of the active region

          coef_x = ((width / FOV[0]) / sensorsize)

          ## Get the scaling coefficient for the y axis
          heigth = sensorsize*n_pixels[1]   # Heigth of the active region

          coef_y = ((heigth / FOV[1]) / sensorsize)

          T = np.array([[coef_x, 0, n_pixels[0]/2], [0, coef_y, n_pixels[1]/2], [0,
      ↪0, 1]])

          row = np.ones(np.shape(p_scene)[1])
          p_scene = np.append(p_scene, [row], axis=0)

          p_out = np.matmul(T, p_scene)

          return p_out[0:2, :]
```

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

      # Parameters
      FOV = [1.024, 0.768]
      sensorsize = 4.88*10**-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]
```

```
[25]:  ####
       # 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)
```

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

We can see that the last to points are out of the image plane, we can notice this cause the y coordinate of the 8th point is -32 (this value is less than 0 so it is out of the FOV) and the x coordinate of the last point is 2224 (this value is greater than 2048, also out of the FOV). We could solve this in our function, if one of the coordinates is out of the FOV, we could throw an exception or control it somehow (printing a message or ignoring it for example.)

1.3.1 Delivery (dead line) on CANVAS: 12-09-2021 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.