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.

You have to fill out code everywhere it is indicated with `...`

The code section under `######## a)` is answering subproblem a) etc.

Problem 1

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

The abbreviation "PSF" stands for "Point Spread Function." The Point Spread Function is a fundamental concept in the field of image processing and is particularly important when dealing

with image degradation and the modeling of image formation processes.

The PSF specifies how a single point of light or a point source in the real world gets spread out or blurred in the resulting image. It describes the response of an ideal optical system to a single point source. In simpler terms, it tells us how a tiny, infinitely small point of light will appear in the image after going through the imaging system. The PSF helps us understand the blurring or smearing of objects in an image due to factors such as lens imperfections, diffraction, motion, and defocus. By convolving the PSF with the ideal object (which would be a point source in this case), we can simulate how that object will appear in the final image. Understanding the PSF is essential for deblurring or deconvolving images. By knowing the PSF, one can attempt to reverse the blurring process and restore the original, sharper image. In various computer vision applications, like medical imaging, knowledge of the PSF is crucial for improving the quality of images and making accurate measurements. PSF is used to characterize the imaging system itself. Different optical systems will have different PSFs, and understanding the PSF can help in system design and evaluation.

b) Use the imaging model shown in Figure 1. The camera has a lens with focal length $f = 40\text{mm}$ and in the image plane a CCD sensor of size $10\text{mm} \times 10\text{mm}$. The total number of pixels is 5000×5000. At a distance of $z_w = 0.5\text{m}$ from the camera center, what will be the camera's resolution in pixels per millimeter?

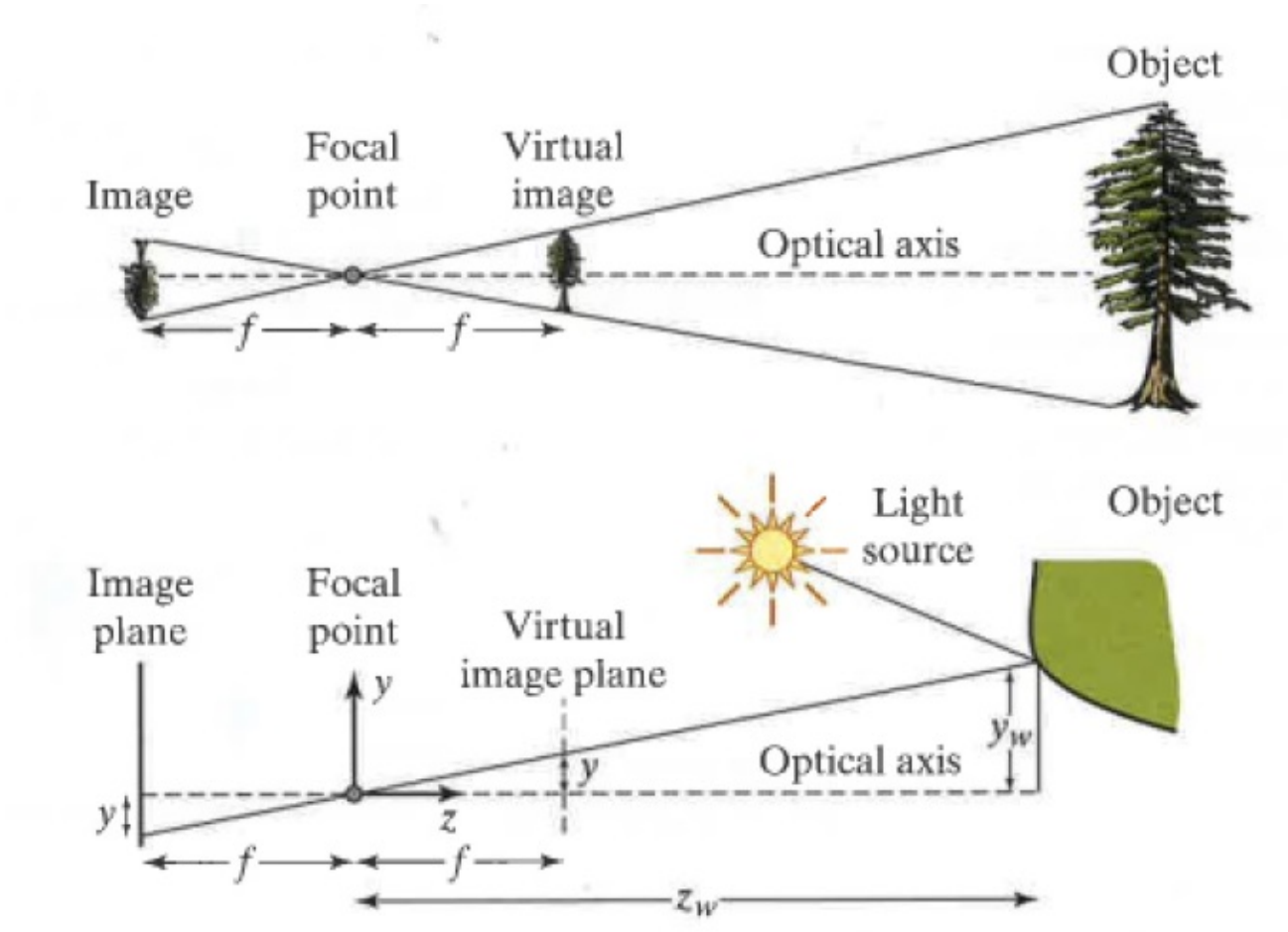


Figure 1: Perspective projection caused by a pinhole camera. Figure 2.23 in [2].

The first thing that we need to do is to find out the yw the real dimension in the real world of the image plane. We can do that by using the formula:

$$y = f \cdot \frac{y_w}{z_w}$$

where y is the dimension in the image plane, f is the focal length, y_w is the dimension in the real world and z_w is the distance from the camera center. By some simple algebra we can find that:

$$y_w = z_w \cdot \frac{y}{f}$$

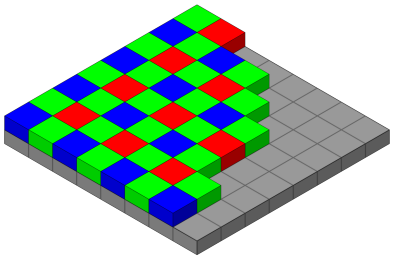
Now we can find the resolution in pixels per millimeter by dividing the number of pixels by the dimension in the real world as follows.

In [ ]:
```
f=40
pixels=5000
sensor=10
zw=500

yw=sensor*zw/f

print("The height in the real world is",yw,"mm")

res = pixels/yw

print("The camera resolution is",res,"pixels/mm")
```

```
The height in the real world is 125.0 mm
The camera resolution is 40.0 pixels/mm
```

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

A Bayer filter is a color filter array (CFA) used in many digital image sensors, particularly in most digital cameras. Its primary function is to capture color information by selectively filtering the incoming light on a pixel-by-pixel basis. The Bayer filter consists of a repeating pattern of color filters, typically arranged in a 2x2 or 4x4 grid, with each pixel containing either a red, green, or blue filter. The most common Bayer filter arrangement is the RGGB pattern, where red and blue filters alternate in the rows and green filters are interleaved.



When light passes through the Bayer filter, each pixel on the image sensor records the intensity of a single color channel. Red pixels only capture red light, blue pixels capture blue light, and green pixels capture green light. To create a full-color image, the missing color information for each pixel needs to be estimated because each pixel only records one color. Interpolation techniques are used to estimate the values of the missing color channels based on the neighboring pixels. These techniques use information from the surrounding pixels with different color filters to estimate the missing colors. Once interpolation is performed, a full-color image is reconstructed from the grayscale image with interpolated color channels.

The alternative to using a Bayer filter in image acquisition is to use three separate image sensors, with a prism that allows to separate the color and projected on three different sensors. This approach is called CMOS or complementary metal-oxide semiconductor. Instead of relying on a single sensor with a Bayer filter to capture color information, these systems use three separate

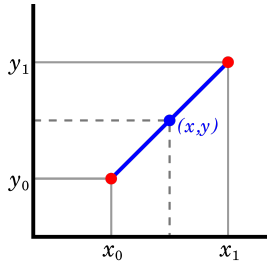
sensors, one each for red, green, and blue channels. Each sensor directly captures the intensity of its respective color, eliminating the need for interpolation.

This type of sensor can be more expensive and bigger but has some advantages over Bayer filter-based sensors: can capture color information with high accuracy since there is no need for interpolation, avoid artifacts caused by interpolation errors, and can potentially offer better low-light performance and higher sensitivity.

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

Sampling:

Sampling refers to the process of converting a continuous signal, such as an analog audio or image signal, into a discrete signal.It involves capturing discrete samples of an image at specific points or pixels. One simple example of interpolation is linear interpolation which estimates values between sampled data points by assuming a straight-line relationship between them.



Quantization:

Quantization is the process of mapping continuous values to a finite set of discrete values. For example, in an 8-bit grayscale image, there are 256 possible discrete intensity levels ranging from 0 (black) to 255 (white). Quantization can introduce errors and reduce the precision of the signal but is necessary for representing continuous data in a digital format. In color images, quantization is applied independently to each color channel (e.g., red, green, and blue).

Gamma Compression:

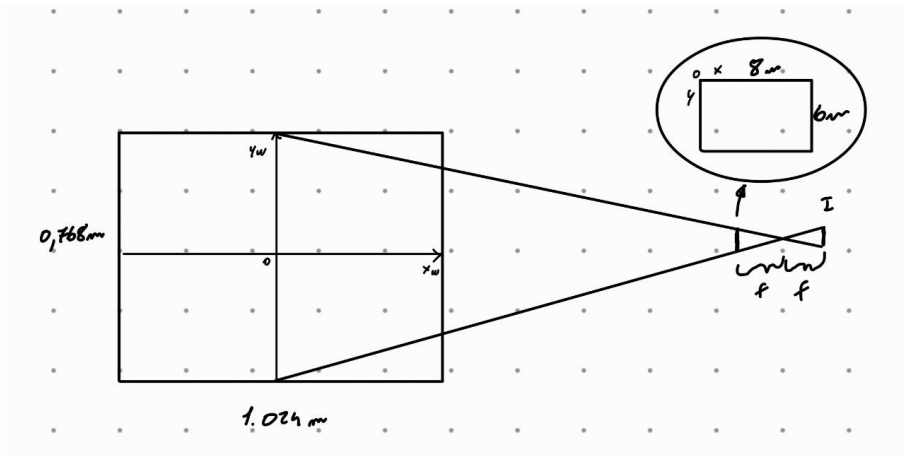
Gamma compression, also known as gamma correction or gamma adjustment, is a nonlinear operation applied to the pixel values in an image to compensate for the nonlinear response of many display devices, such as monitors. It aims to ensure that the displayed image appears visually correct and has the desired contrast. The process involves raising the pixel values to a specific exponent (gamma value), typically in the range of 2.2 to 2.4 for standard displays. The gamma correction helps to correct for the nonlinear relationship between pixel values and perceived brightness, resulting in more accurate and visually pleasing images when viewed on these displays. It is especially important for accurately reproducing grayscale and color tones in images. These concepts are fundamental in digital image processing and play a crucial role in capturing, representing, and displaying digital images accurately and effectively.

Problem 2

Assume we have captured an image with a digital camera. The image covers an area in the scene of size $1.024\mathrm{m} \times 0.768\mathrm{m}$ (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 $8\mathrm{mm} \times 6\mathrm{mm}$.

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:

Sketch of the situation (not to scale):



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

To find the size of each sensor (one pixel) on the CCD-chip, we need to calculate the dimensions of the sensor area divided by the number of pixels in each dimension.

Given information:

- CCD-chip size: $8\text{mm} \times 6\text{mm}$.
- Number of pixels horizontally: 4096.
- Number of pixels vertically: 3072.

Let's calculate the size of one pixel by dividing the sensor size by the number of pixels we will find that all measures are stable in both directions

In [ ]:
```
#given information
h_scene_size = 1.024*1000
v_scene_size = 0.768*1000
h_pixel = 4096
v_pixel = 3072
h_size = 8
v_size = 6
```

In [ ]:
```
h_pix_size = h_size/h_pixel

print("The horizontal pixel size is",h_pix_size,"mm")

v_pix_size = v_size/v_pixel

print("The vertical pixel size is",v_pix_size,"mm")
```

```
The horizontal pixel size is 0.001953125 mm
The vertical pixel size is 0.001953125 mm
```

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?

To find the scaling coefficients, we need to relate the sizes of the scene, the CCD-chip, and the number of pixels.

Given information:

In [ ]:
```
#Scaling coefficient between the image plane (CCD-chip) and the scene:
h_scale = h_size/h_scene_size

print("The horizontal scaling coefficient is",h_scale)

#similarly for the vertical scaling coefficient:
v_scale = v_size/v_scene_size

print("The vertical scaling coefficient is",v_scale)
```

```
The horizontal scaling coefficient is 0.0078125
The vertical scaling coefficient is 0.0078125
```

Scaling coefficient between the scene coordinates and the pixels of the image. This coefficient tells us how much distance in the scene corresponds to one pixel in the image.

In [ ]:
```
#Scaling coefficient between scene coordinates and image plane coordinates:
h_scale = h_scene_size/h_pixel

print("The horizontal scaling coefficient is",h_scale)

#similarly for the vertical scaling coefficient:
v_scale = v_scene_size/v_pixel

print("The vertical scaling coefficient is",v_scale)
```

```
The horizontal scaling coefficient is 0.25
The vertical scaling coefficient is 0.25
```

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. 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:

$$\mathbf{P}_{in} = \begin{bmatrix} 0.512 & -0.512 & -0.512 & 0.512 & 0 & 0.35 & 0.35 & 0.3 & 0.7 \\ 0.384 & 0.384 & -0.384 & -0.384 & 0 & 0.15 & -0.15 & -0.5 & 0 \end{bmatrix} \tag{3}$$

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

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

In [ ]:
```
"""
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):
    alpha_x = sensorsize[0] / (2 * np.tan(FOV[0] / 2))
    alpha_y = sensorsize[1] / (2 * np.tan(FOV[1] / 2))
    x_0 = n_pixels[0] / 2
    y_0 = n_pixels[1] / 2

    # Creating the transformation matrix T
    T = np.array([[alpha_x, 0, x_0],
                  [0, alpha_y, y_0],
                  [0, 0, 1]])

    # Applying the transformation to the scene points
    p_image = np.dot(T, np.vstack((p_scene, np.ones((1, p_scene.shape[1])))))

    # Extracting the 2D image points
    p_image = p_image[:2, :]

    return p_image
```

In [ ]:
```
FOV = (2 * np.arctan((8 / 2) / (1024)), 2 * np.arctan((6 / 2) / (768)))
sensorsize = (8, 6)
n_pixels = (4096, 3072)

p_scene_x = np.array([0.512, -0.512, -0.512, 0.512, 0, 0.35, 0.35, 0.3, 0.7])
p_scene_y = np.array([0.384, 0.384, -0.384, -0.384, 0, 0.15, -0.15, -0.5, 0])
```

In [ ]:
```
####
# 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 casted to int
print (np.round(pimage).astype(int))
```

```
[[2572 1524 1524 2572 2048 2406 2406 2355 2765]
 [1831 1831 1241 1241 1536 1651 1421 1152 1536]]
```

These results demonstrate how the weak perspective transformation affects the positions of the scene points in the image plane. The transormation projects the points from the wall in the real world to the image plane. The results show how the points are mapped to the image plane and how their positions change in the image. The results also illustrate the effects of perspective, where points closer to the camera center appear larger in the image, and points farther away appear smaller.

Imagine you're taking a photo with your smartphone. If you point your phone's camera at an object that's closer to you, it will appear larger in the picture. If you point it to the right, that object will appear on the right side of the photo. If you tilt your phone down, the object will appear lower in the picture. These principles of perspective apply to how cameras capture scenes, and the results illustrate these effects mathematically.

In the last two points we can see that for example the Eighth Point (0.3, -0.5) maps to approximately (2355, 1152). This point has a negative y-coordinate, indicating that it is located below the center of the image.

The Ninth Point (0.7, 0) maps to approximately (2765, 1536) transposed located to the right of the center of the image.

Delivery (dead line) on CANVAS: 15-09-2023 at 23:59

Contact

Course teacher

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

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

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.