ECE 4554 / 5554: Computer Vision: Homework 1

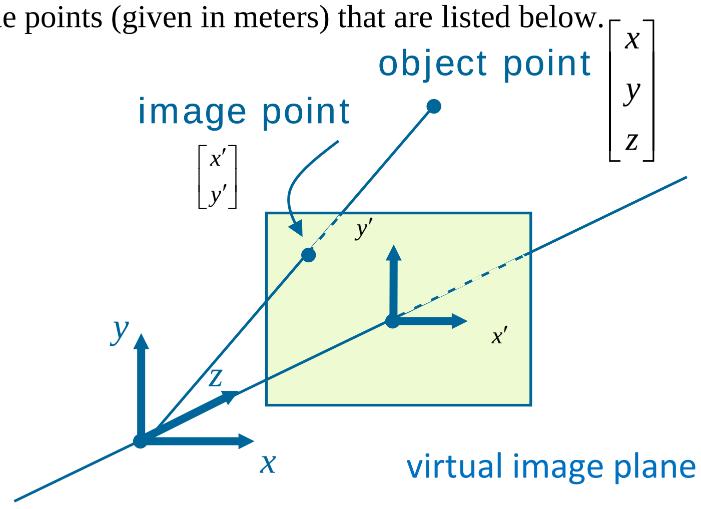
Fall 2020

Instructions

- The assignment is due at Canvas before midnight on Sept. 16. As described in the syllabus, late submissions are allowed at the cost of 1 token per 24-hour period. A submission of only a minute after midnight will cost an additional token.
- Each problem is worth 10 points. Notice that one of the problems is required for 5554 students, but is optional (extra-credit) for 4554 students.
- Prepare an answer sheet that contains all of your written answers in a single file named `LastName_FirstName_HW1.pdf`. (Use your own name, of course.) Handwritten solutions are permitted, but they must be easily legible to the grader.
- Also create a zip file named `LastName_FirstName_HW1.zip`. Inside this zip file place your (pdf) answer sheet along with all of the Python source files and input/output files that are described below. Please do not create subdirectories within the main directory. This zip file is the only file that you should submit to Canvas.
- For the Python implementation problems, try to ensure that the grader can run your code "out of the box". For example, do not encode absolute path names for input files; provide relative path names, assuming that the input files reside in the same directory as the source files.
- If any plots are required, include them in your answer sheet (the pdf file). Also, your Python code must display the plots when executed.
- After you have submitted to Canvas, it is your responsibility to download the zip file that you submitted and verify that it is correct and complete. *The files that you submit to Canvas are the files that will be graded.*

Problem 1. Consider a camera with a focal length of $f = 16$ mm. For this problem, assume that the coordinates of three-dimensional (3D) scene points are specified relative to a left-handed coordinate system that is centered at the point of projection, as shown in the figure below. The z axis is aligned with the optical axis of the camera. The virtual image plane is perpendicular to the z axis, and is a distance f from the point of projection. Assuming ideal perspective projection, find the image locations (in metric units, relative to the x'-y' coordinate system shown) that correspond to the three-dimensional scene points (given in meters) that are listed below.

a) $(x, y, z) = (2, 3, 4)$
b) $(x, y, z) = (4, 6, 8)$
c) $(x, y, z) = (-1, -2, 8)$
d) $(x, y, z) = (0, 0, 4)$


object point
$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$
image point
$\begin{bmatrix} x' \\ y' \end{bmatrix}$
y'
x'
y
z
x
virtual image plane

e) You may assume, without proof, that a 3D line segment projects onto a 2D line segment in the image. Consider the 3D line segment that extends from point (0, 0, 4) to (2, 3, 4). Find the length of the 2D image of this line segment.
f) Consider the 3D line segment that extends from point (0, 0, 4) to (4, 6, 8). Find the length of the 2D image of this line segment.

Problem 2. Consider a camera with focal length f that is positioned above a flat table. The camera is aimed vertically downward to observe the table, so that the image plane is parallel to the table. The distance between the table and the camera's point of projection is z_c.

On the table is a short object, such as a nail, that appears as a line segment in the image. Assuming ideal perspective projection, show that the length of the nail as it appears in the image does not depend on the nail's location on the table. (Assume that the nail is always entirely within the camera's field of view.)

Problem 3. Consider two 2-dimensional kernels: $g = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $h = \begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix}$. Let I represent a 2D image, and let "*" represent 2D convolution. Solve for a new kernel f that will satisfy the following relationship:

$$(I * g) * h = I * f$$

Problem 4. (For <u>5554</u> students, this problem is <u>required</u>. For <u>4554</u> students, this problem is optional and can be submitted for <u>extra credit</u>.) Notice that it is possible to represent an arbitrary line in 3 dimensions using the following parametric form:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} t + \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix},$$

In this expression, $[x_0, y_0, z_0]^T$ is any reference point on the line, $[a,b,c]^T$ is a vector that indicates the direction of the line, and parameter t is an independent scalar variable. (To visualize this, consider $[x_0, y_0, z_0]^T$ and $[a,b,c]^T$ to be constants. Then $[x,y,z]^T$ represents a point on the line that depends on the choice of variable t. This representation is equivalent to vector addition in 3D space, with changes in t tracing different locations $[x,y,z]^T$ along the line.)

In general, lines that are parallel in 3D do not appear parallel in a 2D image. Instead, lines that are parallel in 3 dimensions will appear to converge at a single point in the image, known as the *vanishing point*. Assume perspective projection, and assume that the virtual image plane lies at $z = f$. (As usual, the virtual image plane is perpendicular to the z axis.) Suppose that you are given two lines in 3D that are parallel to the line defined in the equation shown above. Show that you can write an equation for the location of the vanishing point in the image plane for these two 3D lines, in terms of the constants a, b, c, and the focal length f.

Problem 5. The goal of this problem (and the next one) is to become familiar with Python, NumPy, and OpenCV.

a) Write a brief description of the result of each of the following Python commands. Try to guess the result before running the commands interactively. If needed, check the document that is posted at Canvas, *An Introduction to NumPy and SciPy*, or the online API documentation at https://docs.scipy.org/doc/. Include your written description in your answer sheet, but do not submit a screenshot of the result of typing these commands.

```
> import numpy as np
> a = np.array([[1,2,3], [4,5,6], [7,8,9]])
> b = a[2,:]
> c = a.reshape(-1)
> f = np.random.randn(5,1)
> g = f[f>0]
> x = np.zeros(10)+0.5
> y = 0.5*np.ones(len(x))
> z = x + y
> a = np.arange(1,100)
> b = a[::-1]
> c = np.random.permutation(10)
```

b) Write a few lines of code to do each of the following. Copy and paste your code into your answer sheet. (You do not need to place the code into a separate source file.)

1) Let `y` be the vector: `y = np.array([1, 2, 3, 4, 5, 6])`. Use the `reshape` command to form a new matrix `z` that looks like this: `[[1,2],[3,4],[5,6]]`

2) Use `np.max` and `np.where` to set `x` to the maximum value that occurs in `z` (from the previous part of this problem), and set `r` to the row location and `c` to the column location where that maximum value occurs. Remember that Python uses 0-indexing.

3) Let `v` be the vector: `v = np.array([1, 8, 8, 2, 1, 3, 9, 8])`. To a new variable `x`, assign the number of 1's that are present in the vector `v`.

4) Use `np.random.randint` and create an array containing the roll of a six-sided die over `N` trials. (You can pick a convenient value for `N`.)

c) Create any 100 × 100 matrix `A` (without all elements being identical). Save `A` in a file called `inputP5A.npy` and submit it with your solutions. Write a script that loads `inputP5A.npy` and performs each of the following actions on `A`. Name your script `P5.py`. Paste this script into your answer sheet, and also submit the script file.

1) Plot all of the intensities that are contained in `A`, sorted in decreasing value. Provide the plot in your answer sheet. (Note, in this case we don't care about the 2D structure of `A`; we only want a sorted list of all intensities.)

2) Display a histogram of `A`'s intensities with 20 bins. Provide the histogram in your answer sheet.

3) Create a new matrix `X` that consists of the lower left quadrant of `A`. Display `X` as an image in your answer sheet using `matplotlib.pyplot.imshow` with no interpolation. Save `X` in a file called `outputP5X.npy` and submit the file.

4) Create a new matrix `Y`, which is the same as `A`, but with `A`'s mean intensity value subtracted from each pixel. Display `Y` as an image in your answer sheet using

`matplotlib.pyplot.imshow`. Save `Y` in a file called `outputP5Y.npy` and submit the file.

5) Create a new matrix `Z` that represents a color image that is the same size as `A`, but with 3 channels to represent Red, Green, and Blue values. Set the values in `Z` to be red (i.e., R = 255, G = 0, B = 0) wherever the intensity in `A` is greater than a threshold `t =` the average intensity in `A`, and black everywhere else. Display `Z` as an image in your answer sheet using `matplotlib.pyplot.imshow`. Be careful with type-casting. Save `Z` as `outputP5Z.png` and submit the file. (Before submitting, look at `outputP5Z.png` in a standard image viewer to make sure it looks right.)

Problem 6. Write and test a Python/OpenCV program that will perform Gaussian smoothing. Name your script `P6.py`. Paste this script into your answer sheet, and also submit the source file.

Your program should do the following:

- Load an image in grayscale format. The image is provided to you in file `inputP6.jpg`, and you should hard-code this file name into your program.
- Apply a Gaussian smoothing filter by implementing a 2D kernel operation, similar to the code given to you in lecture. Your filter should use $\sigma = 1.414$ and an appropriate kernel size. (See the notes below. Your code must access pixel values directly, possibly using `for` loops.)
- Write the resulting image to a file named `outputP6.png`. Paste this output image into your answer sheet, and also submit the image file.

Notes:

- For this problem, do not use any OpenCV functions other than for loading and saving image files. During the filtering operation, your code must access pixel values directly. (Yes, I know that OpenCV has a built-in Gaussian filter but you cannot use it for this problem.)
- You must also calculate the kernel coefficients yourself. You can use basic Python/NumPy math functions if you wish, but not any special OpenCV functions to create a kernel.
- Your output image should be the same size (rows and columns) as the input image. You may use any technique to assign pixel values near the image border.