Computer Vision I
Assignment 1

Prof. Stefan Roth
Jan-Martin Steitz
Jannik Schmitt

TECHNISCHE UNIVERSITÄT DARMSTADT

09/11/2020

This assignment is due on November 22nd, 2020 at 23:59.

Group work and grading policy
You are required to work on each assignment in groups of ***two***. We reserve the right to merge singleton groups randomly for subsequent assignments.

Programming exercises
For the programming exercises you will be asked to hand in Python code. Please use the provided environment as we will use this environment to test your solution. Use comments in the source code to clarify the functionality of your implementation in sufficient detail. Even if the computed results are incorrect due to a minor bug, you may still earn partial credit if your reasoning is valid.
You *must* adhere to the naming scheme for functions and files included with each problem. Do *not* alter function files and do *not* change the given function signatures. If you feel that there is a mistake in the assignments, or you find the task ambiguous, contact us on Moodle.

Multiple choice questions
There may be multiple choice questions in the assignments, which are implemented via a function or a class method. We provide a detailed explanation *in the source code* about the data type and format of the return value. Please, read the instructions carefully, because the format may vary depending on the question.

Files you need
The data and source code skeleton and the PDF with the assignment tasks will be made available on Moodle.

Handing in
Please, upload your solutions in the corresponding section on Moodle. Each problem task will specify the files to be included in your submission. Please include only the files specified in the problem descriptions and do *not* upload data – such as images – provided by us. You only need to submit one solution per group. Should you have troubles accessing Moodle, get in touch with us as soon as possible. Upload all your solution files as a single `.zip` or `.tar.gz` file. *We do not accept other file formats!*

Late submissions
We will accept late submissions, but you will lose *20% of the total reachable points* for every day of delay. Note that even 15 minute delays will be counted as being one day late! After the assignment solution has been discussed in class, you may no longer hand in.

Other remarks
Your grade will depend on two factors. Firstly, it will be determined by the correctness of your answer. Secondly, it will depend on the clarity of presenting your results and a good writing style. It is your task to find a way to *explain clearly how* you solved the problems. You can still get partial credit even if you did not complete the task.
We encourage interaction about class-related topics both in-class and on Moodle. However, you should not share solutions with other groups, and **everything you hand in must be your own work**. You are also not allowed to copy material from the web without acknowledgment. You must **acknowledge any source of information that you used to solve the homework** (i.e. books other than the course books, papers, etc.). Using acknowledgments will *not* affect your grade, but failing to do so is a clear violation of academic ethics. Note that the university as well as the department is very serious about plagiarism. For more details please see `https://plagiarism.org`.

Problem 1: Getting to know Python (5 Points)

In this task you will set up `Python-3.8` on your system using `Miniconda` and learn how to install additional packages. Packages that you will often use are: (i) `Pillow` for loading and saving images; (ii) `NumPy` and `SciPy` for scientific computing; and (iii) `Matplotlib` for plotting and visualisation and as an alternative for loading and saving images.

Note: In contrast to the `Anaconda` distribution, `Miniconda` supplies only the package management system. This allows to set up minimalistic environments when the disk space is limited (e.g., pool PCs offered by ISP). If you have `Anaconda` already installed and would like to use it instead, the instructions below should also work.

Setup `Miniconda`:

- Install `Miniconda` following the instructions at `https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html`.
- To test if the installation was successful, command `conda list` should return the list of installed packages.
- We will now create a new `conda` environment with the name `cv1`. We assume that the file `requirements.txt` with the dependencies we provide is in the current directory, so simply execute

```
conda env create -f requirements.txt -n cv1
source activate cv1
```

  Here, the second command activates environment `cv1`.

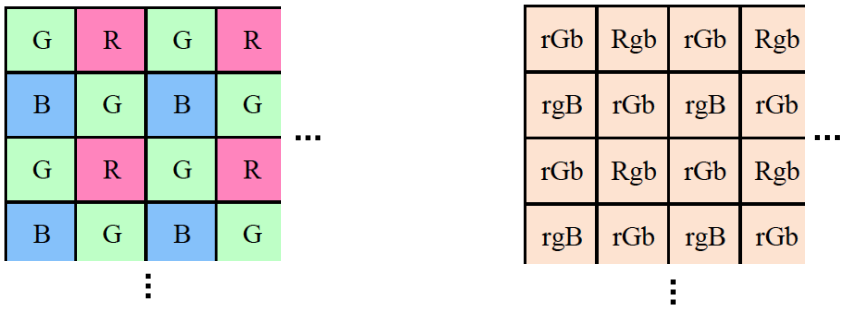
Tasks: We are now ready to execute some Python code! You can use the entry-point `main.py` to import the functions from `problem1.py` to test your code. As a warm-up, complete the following tasks in `problem1.py`.

- Load image `data/a1p1.png` using the `load_image` function we provide in `main.py`.
- Implement `display_image` to show the image using `matplotlib`.
- Implement `save_as_npy` to save the image (a numpy array) to a binary `.npy` file (use `numpy.save()`).
- Implement `load_npy` to load the previously saved file. Check that it is the same image with `display_image`.
- Implement `mirror_horizontal` to horizontally mirror (flip) the image, i.e., the resulting image should revert the pixel order in the horizontal direction.
- Implement `display_images` to display the original and the mirrored (flipped) image from the previous step in *one* plot.

Submission: Please only include the file `problem1.py` in your submission.

Problem 2: Bayer Interpolation (10 Points)

Although today's digital cameras output RGB color pictures or videos, most of them do not have three separate RGB sensing chips but only one single chip based on a *color filter array* (CFA), where alternating sensor sites are covered by different color filters. The Bayer pattern (see Fig. 1) is the most commonly used pattern. In this problem the task is to construct a RGB color image from the data in the Bayer pattern through interpolating the missing color values.



(a) color filter array layout.

(b) interpolated pixel values; with unknown values shown as lower case.

Figure 1: Bayer RGB pattern.

Your task is to write Python code that uses *bilinear interpolation* to restore all missing RGB color values (Fig. 1(b)) from the image data in the Bayer pattern (Fig. 1(a)). Load the image data saved in the Bayer pattern from `bayerdata.npy` and show two images:

1. Display the RGB color image directly transformed from the Bayer pattern, where missing color values should be filled with 0.
2. Display the interpolated full RGB color image.

Tasks:

- Implement the data loading in the function `loadbayer`.
- Separate the color channels filling missing values with zero in the function `separatechannels`.
- Implement bilinear interpolation for the missing values in the function `interpolate`. You should use `ndimage.convolve` from `scipy` package and find appropriate 3×3 kernels. For boundary pixels that might not be (bi)linearly interpolated, please take a look at the `mode` argument of `convolve` and select an appropriate option to make the color of each pixel look consistent with its neighbours.

Notes and Tips: The skeleton is given in `problem2.py`. Please, do not change the function signatures, that is the number of variables the function has as the arguments and the return values. If you are not familiar with (bilinear) interpolation, you can find a tutorial here:
`http://www.cambridgeincolour.com/tutorials/image-interpolation.htm`.

Submission: Please only include the file `problem2.py` in your submission.

Problem 3: Projective Transformation (10 Points)

In this exercise a camera is simulated to allow a specific visualization of a 3D scene as a 2D image. First, transformations are performed on 3D points based on the following order:

- translation by $[6.7,\ -10,\ 4.2]^\top$;
- rotation by -45 degrees around the y-axis;
- rotation by 120 degrees around the x-axis;
- rotation by -10 degrees around the z-axis.

Then a central projection K onto the xy-plane, with principal point $[9;\ -7]$, focal length 8 and square pixels, is applied. We finally obtain 2D points in the image plane.

Tasks: You will start with the obtained 2D points and some other information to restore the original 3D coordinates of all points.

- Implement the functions `cart2hom` and `hom2cart`, which convert an arbitrary matrix of 2D or 3D points to homogeneous coordinates and back.
- All the operations (transformations plus projection) are equal to one single perspective projection. However, doing the computation in separate steps makes debugging easier. Compute the rotation matrices around the individual axes and the translation. Then, compute the camera intrinsics matrix K in `getcentralprojection` and use the obtained matrix to compute the full projection matrix P in `getfullprojection` (all matrices in homogeneous coordinates).
- Load all the 2D points from the file `obj2d.npy` with the function `loadpoints`. The point coordinates are given as column vectors. Show all the points in a 2D plot.
- In the file `zs.npy` you can find the z-coordinates of all transformed 3D points right before the projection is applied. Load them with `loadz` and compute the coordinates (namely, x and y) of these transformed 3D points in `invertprojection`.
- Perform the inverse transformations to get the original 3D coordinates of all points in `inverttransformation`. Show these points in a 3D plot in `displaypoints3d`.
- Use the obtained 3D points, recompute the projected 2D points using the camera C in `projectpoints` and show them in a new figure. Do you get the same 2D points as given?
- Change the order of the transformations (translation and rotation). Check if they are commutative. Adjust the return value of `p3multiplechoice` accordingly.

Submission: Please only include the file `problem3.py` in your submission.

Problem 4: Image Filtering and Edge Detection (15 Points)

In this problem you will get to know how to compute image derivatives and create a simple edge detector. For testing use image `a1p4.png`.

Tasks:

- Implement `createfilters` that creates a 3×3 x-derivative filter that computes the derivative using central differences (i.e., $\frac{f(x+h;y)-f(x-h;y)}{2h}$ where $h = 1$) and at the same time smooths in y-direction with a 3×1 Gaussian filter of $\sigma = 0.9$. In other terms, you obtain the full 3×3 filter by combining a 1×3 with a 3×1 filter. Also, create a corresponding y-derivative filter. Make sure that the filters are correctly normalized.
- Make use of the created filters in `filterimage` to compute the x-derivatives ∂x and y-derivatives ∂y of the input image from above. You can use the `scipy` method `ndimage.convolve` to do the filtering. Note: Make sure that the sign of the derivatives is correct.
- Compute and display the gradient magnitude $\sqrt{\partial_x^2 + \partial_y^2}$. Detect edges by thresholding the gradient magnitude in `detectedges`, i.e., check if the gradient magnitude at each pixel exceeds a certain value.
- Implement *non-maximum suppression* in `nonmaxsupp` to thin the obtained edges. For every edge candidate pixel, consider the grid direction (out of 4 directions) that is “most orthogonal" to the edge. If one of the two neighbors in this direction has a larger gradient magnitude, remove the central pixel from the edge map.

  The initial threshold is set to 1.0, however, this is quite an arbitrary choice. Experiment with various threshold values and choose one that shows the “important" image edges. Briefly explain with a comment in the code how you found this threshold and why you chose it.

Submission: Please only include the file `problem4.py` in your submission.