Computer Vision I, SS13 - Homework Assignment 1 **May 08, 2013**

Prof. Stefan Roth
Qi Gao
Thorsten Franzel
Stephan Richter

This assignment is due on May 20th, 2013 at 23:59.

Group work and grading policy

You are allowed to work on each assignment in groups of ***two*** people. It is up to you to form groups, but please note that the group assignments cannot change after the first homework assignment. Grading will be done in conjunction with personal interviews after you submit your solutions. Each group has to sign up for an interview slot and both group members must be present. Everyone will be graded individually based on the group's implementation and the personal knowledge, i.e., everyone must be able to explain all of the code to us.

The interview will take place in our Linux lab (Room 346 of S3|05); each group will be assigned a 10-minute slot. Right after each assignment submission deadline, you can find available slots and sign up for one in the spreadsheet `http://goo.gl/6ISNV`.

Programming exercises

For the programming exercises you will be asked to hand in Matlab code. If you used any other tool to write your code, say Octave, it is your responsibility to make sure that the code also works on Matlab, which is what we will use for grading. You need to comment your code in sufficient detail so that it will be easily clear to us what each part of your code does.

Your Matlab code should display your results so that we can judge if your code works from the results alone. Of course, we will still look at the code. If your code displays results in multiple stages, please insert appropriate `pause` commands between the stages so that we can step through the code. Please be sure to name each file according to the naming scheme included with each problem.

Pen & paper exercises

You might also have some theoretical exercises to do in the assignments. In this case we would greatly appreciate if you could typeset the theoretical part of your solution (ideally with LaTeX) and submit it as a PDF along with the rest of your solution. If you are not sufficiently familiar with a mathematical typesetting system such as LaTeX, you can also hand in a handwritten solution for this part. Please write neatly and legibly. You can hand in your handwritten solutions with the teaching assistants Thorsten Franzel and Stephan Richter in room 320 of S3|05.

Files you need

All the data you will need for problems will be available at the course web page
`http://www.gris.tu-darmstadt.de/teaching/courses/ss13/cv1`, next to the link for this exercise sheet.

Handing in

UPDATE: Please upload your solutions to Moodle. Students who are not able to access Moodle email their solutions to `cv1staff@gris.informatik.tu-...`**.** You only need to submit one solution per group. You find your group number at `http://goo.gl/7M8tu`.

We would appreciate if you sent all your solution files (the writeup, `.m` scripts and functions, and other required files) as a single `.zip` or `.tar.gz` file following the naming convention groupXX-assignmentY.EXT, where XX stands for your group number, Y for the assignment number, and EXT for the file extension. **Please note that we cannot accept file formats other than the ones specified!**

Late handins

We will accept late handins, but we will take *10 points* off for every day that you are late. Note that even 1 minute late will be counted as being one day late! After the exercise has been discussed in class, you can no longer hand in.

Other remarks

Your grade will depend on various factors. Of course it will be determined by the correctness of your answer. But it will also depend on a clear presentation of your results and good writing style. It is your task to find a way to *explain clearly how* you solved the problems. Note that you will get grades for the solution, not for the result. If you get stuck, try to explain why and describe the problems you encountered – you can get partial credit even if you did not complete the task. So please hand in enough information for us to understand what you did, what things you tried, and how it worked! We will provide skeleton code for every assignment. Please use the provided interfaces as it allows us to better understand what you did.

We encourage interaction about class-related topics both within and outside of class. 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 use material from the web. You are required to **acknowledge any source of information that you used to solve the homework** (i.e. books other than the course books, papers, etc.). Acknowledgements will *not* affect your grade. Thus, there is no reason why you would not acknowledge sources properly. Not acknowledging a source that you have used, on the other hand, 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 `http://www.informatik.tu-darmstadt.de/index.php?id=202` and `http://plagiarism.org`.

G	R	G	R	
B	G	B	G	…
G	R	G	R	
B	G	B	G	
⋮				

(a) color filter array layout.

rGb	Rgb	rGb	Rgb	
rgB	rGb	rgB	rGb	…
rGb	Rgb	rGb	Rgb	
rgB	rGb	rgB	rGb	
⋮				

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

Figure 1: Bayer RGB pattern.

Problem 1 - Warm-Up / 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 figure 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.

Your task is to write Matlab 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.mat` and show two images: *(1)* the RGB color image directly transformed from the Bayer pattern (missing color values should be filled with 0) and *(2)* the interpolated full RGB color image.

The skeleton is given in `problem1.m`. Implement the data loading in `load_bayer.m`, separate the color channels filling missing values with zero in `separate_bayer.m`, and implement the interpolation in `debayer.m`.

You can NOT use the Matlab built-in function `interp2` but have to write your own code to implement bilinear interpolation. For boundary pixels that might not be (bi)linearly interpolated, please find any other ways (*e.g.*, take the values of the nearest neighbours) to make the color of each pixel look consistent with its neighbours. If you are not familiar with (bilinear) interpolation, you can find a tutorial here: `http://www.cambridgeincolour.com/tutorials/image-interpolation.htm`.

Bonus points: if you successfully solve the problem using vectorization (without using any loop, you can get another 2 points as a bonus.)

Problem 2 - Projective Transformation (15 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 $[-27.1, -2.9, -3.2]^{\mathrm{T}}$;
- rotation by 135 degrees around the y-axis; (Note that Matlab uses radians, not degrees, as angular unit.)
- rotation by -30 degrees around the x-axis;
- rotation by 90 degrees around the z-axis.

Then a central projection onto the xy-plane, with principal point $[8; -10]$, focal length 8 and square pixels, is applied. The 2D points in the image plane are thus obtained.

Tasks:

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

- Write two functions `cart2hom.m` and `hom2cart.m`, which convert an arbitrary matrix of 2D-points or 3D-points to homogeneous coordinates and back.

- All the operations (transformations plus projection) are equal to one single perspective projection. Compute and output the corresponding camera matrix P in homogeneous coordinates.
- Load all the 2D points from the file `obj_2d.mat`. The point coordinates are given as column vectors. Show all the points in a 2D plot.
- In the file `zs.mat` you can find the z-coordinates of all transformed 3D points right before the projection is applied. Compute the coordinates (namely, x and y) of these transformed 3D points.
- Perform the inverse transformations to get the original 3D coordinates of all points. Show these points in a 3D plot. (`plot3` in Matlab may be useful, with which you can adjust the viewpoint of the 3D plot using the rotation icon of the figure window.)
- Use the obtained 3D points, recompute the projected 2D points using the camera P 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.

Problem 3 - Image Filtering and Edge Detection (15 points)

In this problem you will get to know how to use Matlab to compute image derivatives and create a simple edge detector. For testing use image `a1p3.png`.

Tasks:

- Create 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 for us $h = 1$) and at the same time smoothes in y-direction with a 3×1 Gaussian filter of $\sigma = 1.0$. 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 and show both as matrices on the Matlab prompt. Make sure that the filters are correctly normalized.
- Use these filters to compute the x- and y-derivative of the input image from above. Use `imfilter` to do the filtering. Set the options for `imfilter` so that the filter output has the same size as the input image, and use the boundary handling option that replicates the pixels on the image edge. Display both results. Note: Make sure that the sign of the derivatives is correct (see corr vs conv option).
- Compute and display the gradient magnitude $\sqrt{\partial_x^2 + \partial_y^2}$. Detect edges by thresholding the gradient magnitude, *i.e.*, check if the gradient magnitude at each pixel exceeds a certain value.
- Use *non-maximum suppression* to thinnen 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 neighbours in this direction has a larger gradient magnitude, remove the central pixel from the edge map. In the end, be sure to binarize these edge maps. Experiment with various threshold values from above step 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.
- *Bonus points:* if you successfully solve the problem of non-maximum suppression without using any `for` loop, you can get another 2 points as a bonus.