Computer Vision II - Homework Assignment 2

Stefan Roth, Simon Kiefhaber, Krishnakant Singh
Visual Inference Lab, TU Darmstadt

May 31, 2024

This homework is due on Jun 14th, 2024 at 17:00.
Please read the instructions carefully!

General remarks

Your grade not only depends on the correctness of your answer, but also on clear presentation of your results and a good writing style. It is your responsibility to find a way to *explain clearly how* you solve the problems. Note that we will assess your complete solution and not exclusively the results you present to us. You can get partial credit even if you have not completed the task, but addressed some of its challenges. Hence, please hand in enough information demonstrating your effort: what you have tried and how your final solution works.

Every group has to submit its own original solution. We encourage interaction about class-related topics both within and outside of class. However, you are not allowed to share solutions with your classmates, and *everything you hand in must be your own work*. Also, you are not allowed to just copy material from the web. You are required to *acknowledge any source of information you use to solve the homework* (*e.g.* books other than the course books, papers, websites). Acknowledgements will *not* affect your grade. However, not acknowledging a source you used is a clear violation of academic ethics. Note that both the university and the department take any cases of plagiarism very seriously. For more details, see the department guidelines about plagiarism and `http://plagiarism.org`.

Programming exercises

For the programming exercises you will be asked to hand in Python code. Please make sure that your code runs with **Python 3.6 or higher** (and **PyTorch 1.8** or higher in later assignments). In order for us to be able to grade the programming assignments properly, insert your code in the provided function definitions. Feel free to experiment with your code in the main() function. However, make sure that your final submission can execute the code originally provided in the main() function. Additionally, comment your code in sufficient detail, so that it is clear what each part of your code does. Sufficient detail does not mean that you should comment every line of code (that defeats the purpose), nor does it mean that you should comment 20 lines of code using only a single sentence. In your final submission, please do not use pop-up windows for visualising intermediate results or ask for user input, unless instructed in the assignment task. Of course, you are welcome, in fact even encouraged, to use visualisation while developing your solution. Please be sure to carefully read the comments in the provided code skeleton, as these provide important details on the function's semantics (*e.g.* what arguments it expects and what results it should return), as well as useful tips on implementation. And finally, please make sure that you included your name and email in the code.

Files you need

All the data you will need for the assignments will be made available over Moodle.

What to hand in

Your hand-in should contain a PDF file for any non-programming ("pen & paper") tasks in the assignment. You are encouraged to typeset your solution, but we will also accept scans or photos of your handwritten solution as long as the image quality does not inhibit its readability. For the programming parts, you must not submit any images of your results; your code should be able to generate these instead. Please hand in your completed version of `.py` scripts provided with the assignment. Make sure your code actually executes and that all functions follow the provided definitions, *i.e.* accept the specified input format and return the instructed output with correct types and dimensions.

Handing in

Please upload your solution files as a single `.zip` or `.tgz` file to the corresponding assignment on Moodle. **Please note that we will not accept file formats other than the ones specified!** Your archive should include your write-up (`.pdf`) as well as your code (`.py`). If *and only if* you have problems with your upload, you may send it to `cv2staff@visinf.tu-darmstadt.de`.

Late Handins

We will accept late hand-ins, but we will deduct 20% of the total reachable points for every day that you are late. Note that even 15 minutes late will be counted as being one day late! After the exercise has been discussed in class, you can no longer hand in. If you are not able to make the deadline, *e.g.* due to medical reasons, you need to contact us *before* the deadline. We might waive the late penalty in such a case.

Code Interviews

After your submission, we may invite you to give a code interview. In the interview you need to be able to explain your written solution as well as your submitted code to us.

Problem 1 – Markov random fields with Gaussian potentials 6 points

In class we have seen how to model a disparity prior using Markov random fields (MRFs). In particular we use a pairwise MRF which is a simple yet effective tool for this task. In this model we represent pixels by nodes in a graphical model and connect every node with its 4 nearest neighbors (left, right, top, bottom). Edges between horizontal and vertical neighbors induce a factor in the probability density which describes the compatibility of the neighboring pixels. For a disparity map $\boldsymbol{x}$ of height M and width N we write such a prior model of disparities as

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{i=1}^{M-1} \prod_{j=1}^{N} f_V(x_{i+1,j} - x_{i,j}) \cdot \prod_{i=1}^{M} \prod_{j=1}^{N-1} f_H(x_{i,j+1} - x_{i,j}).$$

To model the horizontal and vertical potentials f_H and f_V we use Gaussian distributions:

$$f_{H/V}(d) = e^{-\frac{(d-\mu_{H/V})^2}{2\sigma_{H/V}^2}}.$$

For simplicity, we assume that the parameters of the horizontal and vertical potentials are the same, *i.e.* $\sigma_H = \sigma_V = \sigma$ and $\mu_H = \mu_V = \mu$. Since we will not need the normalization term $1/Z$, we will ignore it in the following, *i.e.* you can assume that it is 1.

Tasks:

- Implement the function

  `mrf_log_prior(x, mu, sigma)`,

  that computes the log of the unnormalized MRF prior density with Gaussian potentials for a disparity map `x`. Make sure that your implementation contains terms for *all* horizontal and vertical potentials. Please implement the calculation of the log of the Gaussian potential in the function

  `log_gaussian(x, mu, sigma)`

  2 points

- Implement the function

  `random_disparity(disparity_size)`

  to create a random "noise map" of the same dimensions as the Tsukuba disparity map by drawing uniform random numbers in $[0, 15]$ independently for each pixel. Use the function `np.random.randint()`.

  1 point

- Also implement the function

  `constant_disparity(disparity_size, a)`

  to create a "constant map" with same dimensions taking value a everywhere.

  1 point

- Compare the values of the log-prior density for these three disparity maps and comment on what the relative values say about how well this MRF model captures the properties of the visual world. How does increasing σ of the Gaussian potentials affect the values of the log-prior density and why is this the case? How does reducing the range of the noise map, e.g. to [0,4], affect the values of the log-prior density and why is this the case?

2 points

Problem 2 – Stereo with gradient-based optimization 17 points

In this problem we will put things together and implement an algorithm for stereo estimation using gradient-based MAP estimation. We use the model of the previous problem consisting of *(a)* a pairwise MRF prior over disparity values with Gaussian potentials; and *(b)* our stereo likelihood with Gaussian potentials.

In this and previous assignments you have so far implemented code for evaluating the (unnormalized) log-probability of a disparity map given two input images. For the gradient-based optimization we will now also compute the gradients.

Tasks:

- Implement the function

  `[value, grad] = log_gaussian(x, μ, σ)`

  that calculates the value and the gradient (w.r.t. `x`) of the log-density of the Gaussian distribution. Write the function such that it can also operate on matrix-valued inputs for which it should calculate the gradient element-wise. Here, we want to be more flexible and use given parameters μ and σ.

  1 point

- Now implement the MRF log-prior. Write a function

  `[value, grad] = stereo_log_prior(x, μ, σ)`

  that computes the value and gradient of the log-prior density w.r.t. every disparity value in `x`. Make use of the previously implemented function `log_gaussian`. Here, μ and σ are the distribution parameters of this log-density. Note that the output gradient `grad` should have the same dimensions as the input disparity map.

  2 points

- In the following, we will need to apply the disparity to an image to obtain its shifted version. In contrast to Assignment 1, however, the disparity values can now be continuous. Implement this operation in

  `shifted_i1 = shift_interpolated_disparity(im1, d)`

  where `i1` is the image and `d` is the disparity. The value of the (x, y) coordinate in the output should correspond to the now interpolated (linearly or cubicly) value of the input image at location $(x - d(x, y), y)$. *Hint:* You may find `scipy.interpolate` useful.

  2 points

- Now implement the stereo log-likelihood. Write a function

  `[value, grad] = stereo_log_likelihood(x, im0, im1, μ, σ)`

  that computes the value and gradient w.r.t. every disparity value given the input images $\boldsymbol{I}^0$ and $\boldsymbol{I}^1$. Similarly to the previous task, make use of `log_gaussian` with parameters μ and σ, and your `shift_interpolated_disparity` function.

  2 points

- We now combine the stereo prior and the likelihood the log-posterior. Implement

  `[value, grad] = stereo_log_posterior(d, im0, im1, μ, σ, α)`

  that computes the value and gradient of the stereo posterior. Here, we will use hyperparameter α to trade off the likelihood and the prior:

  $$\log p(d|\mathbf{I}_0, \mathbf{I}_1) \sim \log p(\mathbf{I}_0|\mathbf{I}_1, d) + \alpha \log p(d). \tag{1}$$

  2 points

- With the log-posterior and its gradient now available, we can now estimate the disparity map. To that end, implement a function

  `x = stereo(d0, im0, im1, μ, σ, α, method)`

  that takes an initial guess of the disparity map d_0 and two input images $\boldsymbol{I}^0$ and $\boldsymbol{I}^1$. It should optimize the log-posterior to a local optimum by a gradient-based method. The last parameter `method` specifies the optimization algorithm (set in function `optim_method`). Please, use `minimize()` from `scipy.optimize` and pass the method name to its `method` argument. `minimize()` supports a number of optimization methods, but only a few are suitable for our problem with regard to the computational time, the memory footprint and the quality of the solution. Set the optimization algorithm of your choice in function `optim_method()`. *Hint:* Look into the family of quasi-Newton methods and experiment with `minimize()` first on a toy example.

  3 points

Evaluate your algorithm with the supplied stereo image pair `i0.png` and `i1.png` with ground truth `gt.png`. You can set $\mu = 0$ and $\sigma = 1.2$, but feel free to experiment with α.

- Experiment in function `main()` of the script `problem2` that loads the images and invokes your stereo method. After your algorithm has finished, visualise the ground truth disparity, your estimated disparity map, and the differences between both (not graded).
  Pen & Paper. Compare different initializations and report your observations when you: *(i)* initialize the disparity map with from the provided ground truth; *(ii)* initialize the disparity map with a constant value, *i.e.* $\boldsymbol{x} \equiv 8$; *(iii)* when you initialize with uniform noise in $[0, 14]$. How sensitive is your optimization method w.r.t. the initializations in terms of the attained local optimum of the log-posterior? How does α qualitatively affect your results?

  2 points

- Now implement coarse-to-fine estimation in

  `ds = coarse2fine(d0, im0, im1, μ, σ, α, num_levels)`

  For this, use subsequently downsampled versions of each input image and run the algorithm first on the coarsest scale. For the next iteration initialize with an upscaled version of the current estimate of the disparity map (do not forget to scale the disparity values after upsampling). The function returns a list of size `num_levels` which contains your disparity estimates (in the finest to coarset order).

  3 points