CS59300CVD Assignment 3

Due date: Sunday, Nov. 3, 11:59 PM

This assignment was adapted from Svetlana Lazebnik's course on computer vision.

Homework Policy: Each student should write up **their solutions independently, no copying of any form is allowed.** If you have used online resources to help with answering the questions, but have come up with your own answers please properly reference the resources. You are allowed to use online resources as long as the answer is your own and properly acknowledge it. Please refer to the course website for policy on academic honesty. Finally, You MUST USE the provided LaTeXtemplate to typeset the report.

1 Assignment Details

1.1 Part 1: Image Stitching

Objective: In this assignment you will implement image stitching to create a panorama image. The assignment is designed to cover the following concepts: key-point detection, key-point descriptor, fitting, and RANSAC.

Method description:

1. Load both images and convert them to grayscale.
2. **Detect feature points in both images.** We extract corner key points using the provided Harris corner detector, see provided code in the `helpers.py`. Note, the helpers assume numpy arrays, hence functions like `kornia.utils.tensor_to_image`, and `kornia.utils.image_to_tensor` may be useful.
3. **Extract key-point descriptors.** We will consider the following descriptors:

   (a) Extract local neighborhoods around every keypoint in both images, and form descriptors simply by "flattening" the pixel values in each neighborhood to one-dimensional vectors. Subtract the mean and normalize the vector to have unit-norm. Experiment with different neighborhood sizes to see which one works the best. **Hint:** You should revisit the median filter code covered in the lecture.

   (b) We will also extract descriptors using a pre-trained HyNet. HyNet takes in a 32×32 patch and outputs a feature vector of size 128. Use this 128 dimension vector as the descriptor. Please refer to the Kornia's documentation for details.
4. Compute ℓ_2 distances between every descriptor in one image and every descriptor in the other image. In Pytorch, the function `torch.cdist` may be useful.
5. Select putative matches based on the matrix of pairwise descriptor distances obtained above. You can select all pairs whose descriptor distances are below a specified threshold, or select the top few hundred descriptor pairs with the smallest pairwise distances. In PyTorch, functions `torch.argsort`, and `torch.unravel_index` may be useful.
6. Implement RANSAC to estimate a homography mapping one image onto the other. Report the number of inliers and the average residual for the inliers (squared distance between the

point coordinates in one image and the transformed coordinates of the matching point in the other image). Also, display the locations of inlier matches in both images by using the provided `plot_inlier_matches`.

A very simple RANSAC implementation is sufficient. Use four matches to initialize the homography in each iteration. You should output a single transformation that gets the most inliers in the course of all the iterations. For the various RANSAC parameters (number of iterations, inlier threshold), play around with a few "reasonable" values and pick the ones that work best. Refer to the lecture on RANSAC for details.

Next, you will implement homography fitting (homogeneous least squares). As described in the lecture, the solution to the homogeneous least squares system $\boldsymbol{AX} = 0$ is obtained from the SVD of A by the singular vector corresponding to the smallest singular value. In PyTorch, functions `torch.svd` may be useful.

7. Warp one image onto the other using the estimated transformation. Create a new image big enough to hold the panorama and composite the two images into it. You can composite by averaging the pixel values where the two images overlap, or by using the pixel values from one of the images. In Python, `skimage.transform.ProjectiveTransform` and `skimage.transform.warp` may be useful.

1.2 Part 2: Shape from shading

Objective: The goal of this part is to implement shape from shading as described in the lecture; Also see Section 2.2.4 of Forsyth & Ponce 2nd edition).

Method description:

1. The data consists of 64 images for each of the four subjects from the Yale Face database. The light source directions are encoded in the file names. We have provided utilities to load the input data and display the output. Your task will be to implement the functions preprocess, `photometric_stereo` and `get_surface` in the starter code, as explained below.

2. For each subject (subdirectory in croppedyale), load the images and light source directions. The function `LoadFaceImages` returns the images for the 64 light source directions and an ambient image (i.e., an image taken with all the light sources turned off). The `LoadFaceImages` function is completed and provided to you in the starter code.

3. Preprocess the data: subtract the ambient image from each image in the light source stack, set any negative values to zero, and rescale the resulting intensities to between 0 and 1. Complete the `preprocess` function.

4. Estimate the albedo and surface normals. For this, you need to fill in code in `photometric_stereo`, which is a function taking as input the image stack corresponding to the different light source directions and the matrix of the light source directions, and returning an albedo image and surface normal estimates. The latter should be stored in a three-dimensional matrix. That is, if your original image dimensions are $h \times w$, the surface normal matrix should be $h \times w \times 3$, where the third dimension corresponds to the x-, y-, and z-components of the normals. To solve for the albedo and the normals, you will need to set up a linear system. To get the least-squares solution of a linear system, the `torch.linalg.lstsq` function may be helpful. Complete the `photometric_stereo` function.

5. If you directly implement the formulation from the lecture, you will have to loop over every image pixel and separately solve a linear system in each iteration. Do not do this! Instead, formulate the problem such that you can get all the solutions at once. This is done by stacking the unknown g vectors for every pixel into a $3 \times N$ matrix, where N denotes the number of pixels. You will be able to get all the solutions with a single call to the solver. You will most likely need to reshape your data in various ways before and after solving the linear system.

6. Compute the surface height map by integration. More precisely, instead of continuous integration of the partial derivatives over a path, you will simply be summing their discrete values. Your code implementing the integration should go in the `get_surface` function. As stated in the slide, to get the best results, you should compute integrals over multiple paths and average the results. Complete the `get_surface` function.

7. You should implement the following variants of integration: Integrating first the rows, then the columns. That is, your path first goes along the same row as the pixel along the top, and then goes vertically down to the pixel. It is possible to implement this without nested loops using the `torch.cumsum` function. Integrating first along the columns, then the rows. Average of the first two options. Average of multiple random paths. For this, it is fine to use nested loops. You should empirically determine the number of paths needed.

8. Display the results using functions `display_output` and `plot_surface_normals`. Feel free to adjust the viewing angle to have a better view.

2 What to include in the report? (10 points)

2.1 Part 1

- Describe your solution, including any interesting parameters or implementation choices for feature extraction, putative matching, RANSAC, etc. (1 point)
- For each of the descriptors, report the number of homography inliers and the average residual for the inliers (squared distance between the point coordinates in one image and the transformed coordinates of the matching point in the other image). Also, display the locations of inlier matches in both images. (3 point)
- Show the final result of your stitching from both descriptors. (1 point)

2.2 Part 2

- Briefly describe your implemented solution, focusing especially on the more "non-trivial" or interesting parts of the solution. What implementation choices did you make, and how did they affect the quality of the result and the speed of computation? What are some artifacts and/or limitations of your implementation, and what are possible reasons for them? (1 point)
- Discuss the differences between the different integration methods you have implemented. Specifically, you should choose one subject, display the outputs for all of a-d (be sure to choose viewpoints that make the differences especially visible), and discuss which method produces the best results and why. You should also compare the running times of the different approaches. For the remaining subjects, it is sufficient to simply show the output of your best method, and it is not necessary to give running times. (1 points)

- For every subject, display your estimated albedo maps and screenshots of height maps (use `display_output` and `plot_surface_normals`). When inserting results images into your report, you should resize/compress them appropriately to keep the file size manageable – but make sure that the correctness and quality of your output can be clearly and easily judged. For the 3D screenshots, be sure to choose a viewpoint that makes the structure as clear as possible (and/or feel free to include screenshots from multiple viewpoints). You will not receive credit for any results you have obtained, but failed to include directly in the report PDF file. (2 point)
- Discuss how the Yale Face data violates the assumptions of the shape-from-shading method covered in the slides. What features of the data can contribute to errors in the results? Feel free to include specific input images to illustrate your points. Choose one subject and attempt to select a subset of all viewpoints that better match the method's assumptions. Show your results for that subset and discuss whether you were able to get any improvement over a reconstruction computed from all the viewpoints. (1 point)

While code is not explicitly graded. Poorly commented or written code will result in points deducted from the previous three parts. For example, if the grader cannot understand what was implemented.

What to include in the code?

The code should contain a README.md file that describes how to run and reproduce your result. This includes the packages installed and their versions. We expected adequately commented code. To help with readability, please follow a style guide and use a linter, e.g., PEP8 is usually a good choice.

How to submit?

- You should have received an e-mail with the link to access Gradescope. If you haven't, let the TAs know.
- For your pdf file, use the naming convention `username_hw#.pdf`. For example, your TA with username *alice* would name her pdf file for HW3 as `alice_hw3.pdf`.
- Use the provided LaTeX template to typeset the assignment by editing the tex files under `student_response/`.
- After uploading your submission to Gradescope, mark each page to identify which question is answered on the page. (Gradescope will facilitate this.)
- For your code, use the naming convention `username_hw#.zip`. The code can be submitted via Gradescope under **assignment3 programming**.