ECE661: Homework 5

Fall 2022
Due Date: 11:59pm, Oct 12, 2022

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace.

1 Introduction

In this homework, you will learn to implement a fully automated approach for robust homography estimation and subsequently refine it using Nonlinear Least-Squares minimization approaches such as Levenberg-Marquardt (LM) algorithm.

In Homework 4, you used your own Harris corner detector and SIFT or SURF implementation from OpenCV to obtain the feature descriptors, and SSD and NCC to establish the correspondences. You most probably noticed some false correspondences in your output. In this homework you will use RANdom SAmpling Consensus (RANSAC) algorithm (ref. Lecture 12) for outlier rejection and a linear least squares algorithm (ref. Lecture 11) to obtain an initial homography estimation. The subsequent LM-based refinement (ref. Lecture 13) gives more robust homography estimation.

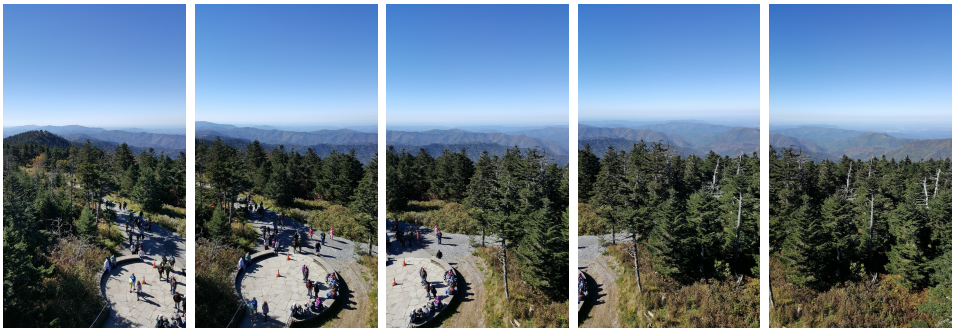
You will implement the above steps to generate a *panoramic* view by "stitching" together five (or more) overlapping views of the same scene. This is also known as *image mosaicing*. As mentioned earlier, you will generate this panoramic view without any manual annotations. You will also capture your own five or more images as explained in section 3.

2 Theory Questions

For this homework you have two theoretical questions.

1. Conceptually speaking, how do we differentiate between the inliers and the outliers when using RANSAC for solving the homography estimation problem using the interest points extracted from two different photos of the same scene?

2. As you will see in Lecture 13, the Gradient-Descent (GD) is a reliable method for minimizing a cost function, but it can be excruciatingly slow. At the other extreme, we have the much faster Gauss-Newton



(a) Image 0 (b) Image 1 (c) Image 2 (d) Image 3 (e) Image 4

Figure 1: The five input images. Note that since the scene contains many repetitive structures, e.g. trees, a relatively large number of outliers are expected. Still, if done correctly, you will be able to find a reasonable set of inliers for homography estimation owing to the power of RANSAC.

(GN) method but it can be numerically unstable. Explain in your own words how the Levenberg-Marquardt (LM) algorithm combines the best of GD and GN to give us a method that is reasonably fast and numerically stable at the same time.

3 Programming Tasks

3.1 Task 1

For this homework you will need five or more ($n \geq 5$) images of the same scene. You must implement your own Linear Least-Squares and RANSAC algorithms. Use the following steps to produce a panorama consisting of the five images from fig. 1.

1. Let's assume that your input images are numbered like $1, 2, \cdots, n$. Note that, there should be significant overlap in any two successive views. Your code should be able to estimate homographies between views, 1 and 2, 2 and 3, $\cdots$, $n-1$ and n. After estimating and refining all the homographies, you will project all the views onto a common reference frame, e.g., middle image.

2. Similar to your Homework 4, extract some interest points and their descriptors, and find the initial set of correspondences in a pair of images. You can use your own Harris corner detector and SSD or

NCC you implemented in Homework 4. Alternatively, you can use SIFT or SURF and a force matcher that directly compares the feature vectors of interest points.

3. As you learned in Lecture 12, the set of correspondences you obtain from the previous step will have both inliers and outliers.

4. The key steps for automatically estimating homographies involve (a) Outlier rejection using the RANSAC algorithm, (b) Homography estimation using a Linear Least-Squares method (ref. Lecture 11) with the help of inliers obtained from the previous step, and (c) Homography refinement using a Nonlinear Least-Squares approach (ref. Lecture 13).

5. Use all the pairwise homographies to project all the views on to a fixed common frame. Note that you should apply appropriate scaling and interpolation strategies to avoid a very large sized image for output and to smooth out the visual artifacts, respectively.

3.2 Task 2

Repeat Task 1 with your own set of five or more images.

3.3 Notes

Linear Least-Squares Minimization: In Lecture 11, you learned two approaches for estimating homographies with Linear Least-Squares minimization – using (a) Homogeneous and (b) Inhomogeneous equations. The first one is relatively easier to implement than the latter. In summary, a set of correspondences gives you an over-determined system of homogeneous equations, i.e., $\mathbf{Ax} = \mathbf{0}$. The SVD of $\mathbf{A}$ would give you $\mathbf{A} = \mathbf{UDV}^T$. The required solution is the last column vector of $\mathbf{V}$ which will be the eigenvector of $\mathbf{A}^T\mathbf{A}$ corresponding to the smallest singluar value in $\mathbf{D}$.
RANSAC: The key idea behind RANSAC algorithm is that starting with a pair of images and an initial set of correspondences, you randomly pick a few correspondences, say five or six at a time. This randomly picked set of point would give you an estimate of homography using least squares method explained earlier. Use this homography estimation to project all the initial correspondences from the domain to range image. Now count the number of inliers (use your answer to the first theoretical question to decide how to find these "inliers"). Keep the homography that returns the largest number of inliers.

LM for Homography Refinement: For this step, you have a choice either (a) use any open-source implementation (e.g. scipy, levmar) for LM algorithm or (b) implement your own LM-algorithm for homography refinement. For more detailed explanation of LM algorithm refer to [3]. Prof. Kak has his own open-source python module of LM-algorithm, you can find the two available versions of the module by visiting [1] and [2].

4 Extra Credits (30 pts)

To earn the extra credits, you will need to complete the following two steps:

1. Implement your own Levenberg–Marquardt algorithm. You are free to use routines in `scipy` and `numpy` to help with your linear algebraic needs. However, using directly `scipy.optimize.least_squares` is not allowed.

2. Quantitatively demonstrate the effectiveness of your LM implementation by showing the geometric errors before and after applying your non-linear least squares solver. In the notations of Lecture 13 (page 1), the geometric error E for a set of matched keypoints is defined as:

$$E(\overrightarrow{p}) = \|\overrightarrow{X} - \overrightarrow{f}(\overrightarrow{p})\|^2. \tag{1}$$

   Note that you should use the full set of inliers for calculating the geometric error.

3. Tabulate the reductions in geometric error by your LM algorithm on at least three pairs of images that you used for Task 1 and 2.

5 Submission Instructions

Include a typed report explaining how did you solve the given programming tasks.

1. Turn in a zipped file, it should include (a) a typed self-contained pdf report with source code and results and (b) source code files **(only .py files are accepted — convert your .ipynb files to .py)**. Rename your .zip file as hw5_<First Name><Last Name>.zip and follow the same file naming convention for your pdf report too.

2. Your pdf must include a description of

- Your answers to the theoretical questions in section 2.
- A clear description of your implementation of (a) RANSAC algorithm, (b) Least-Squares method, and (c) the LM Algorithm
- Extracted correspondences between the sets of adjacent images. Similar to Homework 4 draw lines to show your correspondences.
- Show the sets of outliers and inliers for at least two pairs of images.
- Optimal set of parameters that produced good results.
- The final output mosaic or panoramic view obtained after stitching your input images.
- Your source code. Make sure that your source code files are adequately commented and cleaned up.

3. To help better provide feedbacks to you, make sure to **number your figures**.

4. The sample solutions from previous years are for reference only. **Your code and final report must be your own work.**

References

[1] NonlinearLeastSquares 1.1.1, . URL https://pypi.org/project/NonlinearLeastSquares/1.1.1/.

[2] NonlinearLeastSquares 2.0.0, . URL https://pypi.org/project/NonlinearLeastSquares/.

[3] Levmar. URL https://users.ics.forth.gr/~lourakis/levmar/levmar.pdf.