ETH Zürich
Computer Science Department

Prof. Marc Pollefeys
Prof. Siyu Tang
Prof. Vittorio Ferrari
HS 20

Computer Vision - Exercise 7
Structure from Motion

Hand-out: 19-11-2020
Hand-in: 28-11-2020
denys.rozumnyi@inf.ethz.ch

Objective:

In this exercise you will put the different parts from the previous exercises together and create a very simple structure from motion pipeline. This time you should try to reuse your code as much as possible. You are also allowed to use code for the parts we already implemented from other sources (Peter Kovesi's homepage is a good source).

7.1 Feature extraction and initialization with epipolar geometry (40%)

To start the pipeline the first thing to do is to extract features in every image. Then select two images that have a large baseline but at the same time a big overlap so that there are enough feature matches. The larger the baseline, the better the triangulation of the feature points.

- In the code framework 5 images are included. For this step you should take the first and the last image of the sequence. Extract SIFT features and match them.
- Run either 8-point or 5-point RANSAC to compute the inlier-set and then compute the essential matrix. Decompose the essential matrix in to R and t and create the projection matrix for the second view assuming that the first one is $[I \mid 0]$.
- Triangulate the matched inlier features with the existing code and store their 3D positions. Now you have 3D-2D point correspondences for the inlier features.

7.2 Triangulation and adding new views (40%)

To add additional views of the scene you can now match 2D features in the new view and one of the existing views. These matches represent now also 3D-2D correspondences, the 2D point from the new image and the 3D from the 3D-2D correspondence of the existing view.

To find the pose of the new camera relative to the scene (the P matrix) you can use the 6-point DLT algorithm from the calibration exercise. To filter out wrong matches, use a 6-point RANSAC, that uses the reprojection error as error measure.

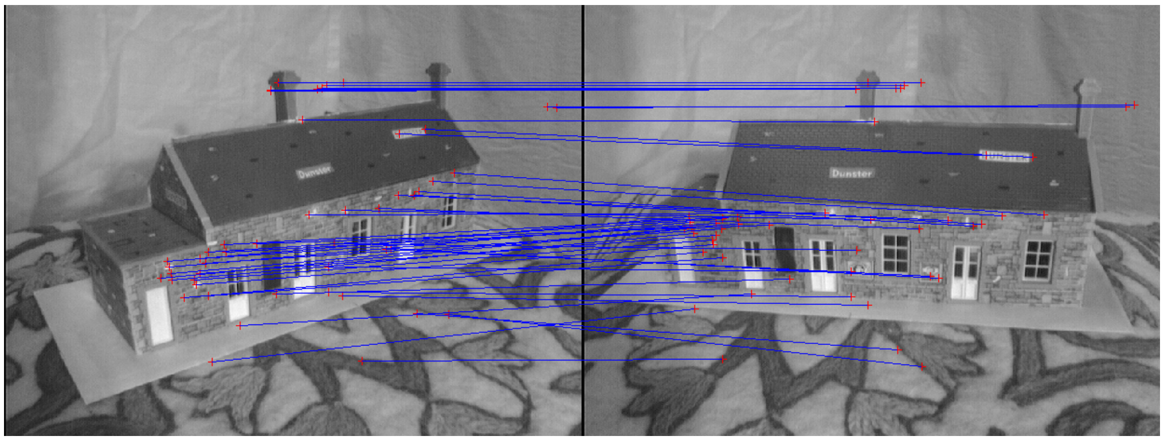

Figure 1: Task 1: Inlier matches after 8-point RANSAC (images 0 and 4).

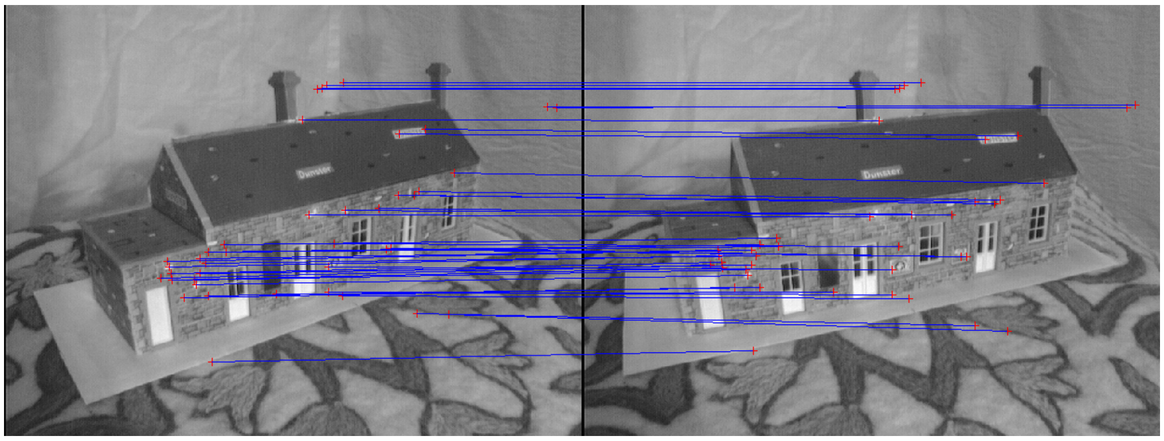

Figure 2: Task 2: Inlier matches after 6-point RANSAC (images 0 and 1).

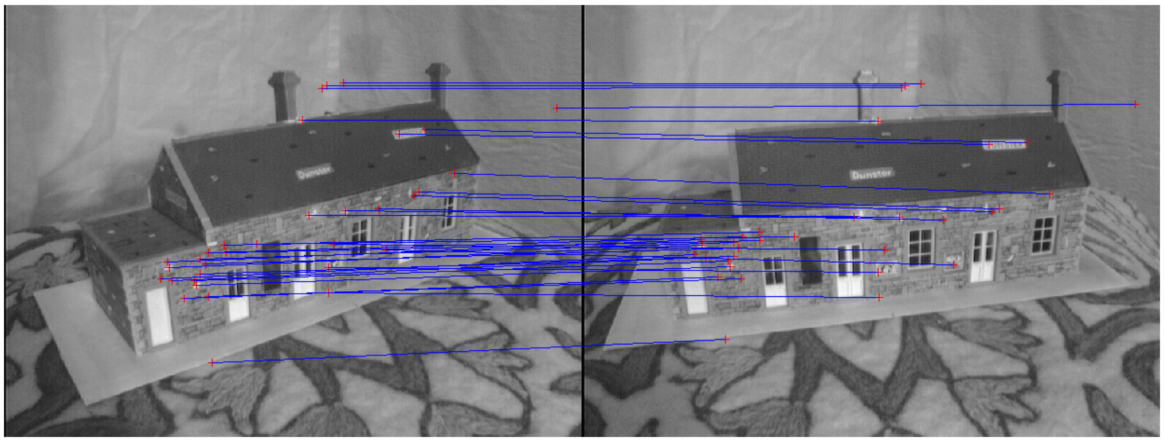

Figure 3: Task 2: Inlier matches after 6-point RANSAC (images 0 and 2).

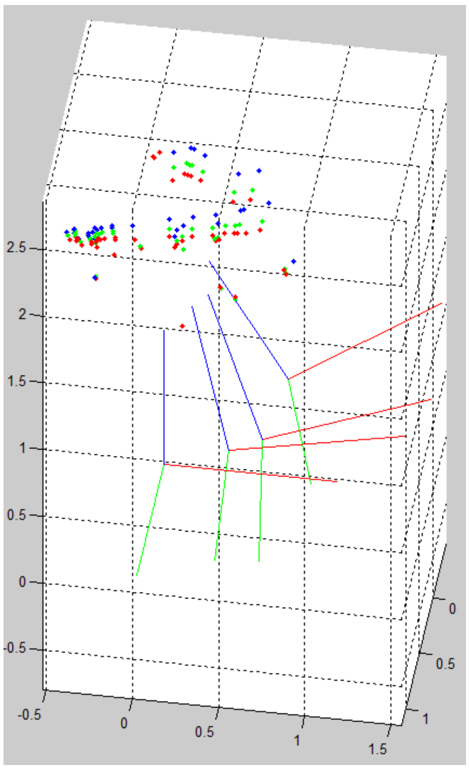

Figure 4: Task 3: Triangulated points and camera poses plotted in 3D. Red dots are the triangulated points from the initialization, green dots are the triangulated points from the first additional image and blue dots are from the second addtional image.

7.3 Plotting (20%)

To analyze your results, triangulate all inlier matches from the previous task and plot them in 3D together with the inliers from the first task. Also visualize the camera poses of every view (a function for that is provided).

You will see that the result is not perfect; in a typical SfM pipeline, one would now run a bundle adjustment that would globally optimize the feature and camera positions.

7.4 Hand in:

Write a short report explaining the main steps of your implementation. The report should contain figures showing:

a) Epipolar geometry of the image pair used for initialization

b) Inlier and outlier matches for every image used in each step

c) 3D image of the scene showing the triangulated feature point and cameras

Upload the report together with your source code (but not the data) to **Moodle**.

7.5 Dense reconstruction (bonus: up to 20%)

Until now you only have a sparse reconstruction of the scene based on the feature points. To get a dense reconstruction you can compute the depth map with stereo matching and triangulate every pixel that has a depth and project it into the scene. For this exercise it is enough to compute the depth map between two views only.

Instead of only showing a 3D point cloud, take the RGB value of the pixel where the feature is located in one of the images and color the 3D-point with that value or create a mesh of the scene (this will not be perfect near depth discontinuities of course) - both options are implemented in the `create3DModel` function. Look at the code and chose your favorite option. Of course, you can create an VRML scene or something similar - be creative.