STRUCTURE FROM MOTION AND BUNDLE ADJUSTMENT

Student ID: A0283551A

March 29, 2024

Introduction

In this assignment a dataset of images was given, from which the structure of the imaged scene was to retrieved. This was done in a two step fashion, by first running an iterative structure from motion procedure followed by a non-linear least-square optimization phase(a.k.a bundle adjustment/sparse pose adjustment). The general approach can be summarized as follows:

1. **Preprocessing**
   (a) Find interest points and descriptors for each image (SIFT was used)
   (b) For each pair of images find matches between keypoints (rejecting using Lowe ratio see Lowe 2004
   (c) Perform geometric outlier rejection using RANSAC and compute essential matrix for each image pair
   (d) Create adjancy graph connecting all images with a sufficient number of matcher
2. **Iterative Structure from Motion**
   (a) Find the two images with the highest number of matches, and find their relative position from the essential matrix. Also triangulate to get the corresponding 3d points.
   (b) For each unprocessed image, pick the processed image with the most matches, and solve the PnP problem between these two images to get relative pose.
   (c) Add any new 3D points from the matches
3. **Bundle Adjustment**
   (a) Use a iterative solver to improve the estimates of the poses and the 3D points.

While the system is quite complex, and relies on a lot of different techniques and algorithms outlined elsewhere, the tasks to be completed in this project were relatively straight forward. This report will mention all the specific task that were implemented, and comment upon anything non-trivial.

Preprocessing

For the preprocessing four functions were to be implemented, namely `detect_keypoints`, `create_feature_matches`, `create_ransac_matches`, `create_scene_graph`. This section will go through any important implementation details for these functions.

As mentioned keypoints and corresponding descriptors were found using SIFT as outlined in Lowe 2004. The ratio test was implemented as per this paper to reduce the number of mismatches. This is implemented in the `create_feature_matches` funciton. As the CV2 documentation outlines, an alternative to this could be to cross check the matching. I.e instead of finding the two nearest neighbours and checking the Lowe ratio, one could instead, for each keypoint in image 1 find the best match in the second image, and discard it if it is not the best match of that keypoint of all keypoints in image 1OpenCV 2022. This was tested also and qualitatively seemed to work well.

The `create_scene_graph` function quite simply looped over all pairs of images and checked the number of ransac verified matches, and added a edge between the two images if the number was above a certain threshold. Since the graph is undirected, a slightly optimized approached was used, where each pair was checked only once, and an edge was added in each direction. This is why the inner loop only loops from the value of the outer loop iterator.

Iterative Structure from Motion

In this part mainly 6 functions were to be implemented. Namely `get_init_image_ids`, `get_init_extrinsics`, `get_next_pair`, `solve_pnp`, `add_points3d` and `get_reprojection_residuals`. This section covers the non-obvious implementation details.

Finding the image pair with the most matches in `get_init_image_ids` is a very similar problem to finding all image pairs with sufficient number of matches in `create_scene_graph` from the preprocessing step. Therefore, also here we only need to check each pair once. Since the graph structure doesn't lend itself to easily implement this in the for loop, this in enforced with an if statement, checking the relation between the image ids.

Following the method in [Hartley and Zisserman 2003, sec. 9.6, p.258-259] the `get_init_extrinsics` function finds the relative pose between the two camera poses from the SVD of the essential matrix. The issue is that there are 4 possible solutions to this pose. As outlined in [Hartley and Zisserman 2003, sec. 9.6, p.258-259] the right one can be found be checking that a 3D point is in front of both cameras. Since this is perhaps the most involved part of this project, this is briefly explained in the following.

A random 2d point correspondence is first chosen and normalized by the camera calibration matrix $\mathbf{K}$. Here the first one is chosen. Triangulation is performed for each of the four possible camera matrices $\mathbf{P}$ coordinate using a DLT style algorithm from the two image points as explained in [Hartley and Zisserman 2003, sec. 12.2 p.312-313]. While the depth of the point can be estimated from a camera matrix and the 3D point (see p.163 in Hartley and Zisserman 2003), here we only need to find its sign. For the first canonical camera it can be seen that this is the same as X_3 that is the third element of the scene point in homogeneous coordinates. For the second camera the sign is the same as the value of $\mathbf{P}^{3\top}X$. Note that in finding the camera matrices of the second camera the determinant of the rotation matrix $\mathbf{R}$ should be $+1$. Since the SVD might give $\mathbf{U}, \mathbf{V}$ matrices with either ± 1 as determinant this can be enforced by dividing the rotation matrix by its determinant. Therefore $\det \mathbf{M} = 1$ in $\mathbf{P} = \left(\mathbf{M}|\mathbf{p_4}\right)$. Consequently we select the camera matrix, where both these values are positive.

Bundle Adjustment

This part only consists of implementing one function - `compute_ba_residuals`. Implementing this function requires some thinking relating to the order of indices for the matrices, but with a little diligence can be implemented without using loops which are especially slow in python. Verifying the correctness and understanding the code should however be straight forward. The only thing worth noting is the residuals the optimizer is expecting. These should be the difference, in each element, of the image points and the re-projected image points, not the euclidean distance between these as implied in the code comments. Both implementations were tested and the using the difference rather than the distance resulted in more than 10 times lower re-projection error.

Bibliography

Hartley, Richard and Andrew Zisserman (2003). *Multiple View Geometry in Computer Vision*. PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE.

Lowe, David G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60, pp. 91–110. URL: `https://api.semanticscholar.org/CorpusID:174065`.

OpenCV (2022). *OpenCV: Feature Matching*. Accessed: March 29, 2024. URL: `https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html`.