COMP5421 - Computer Vision - Final project report – 3D Reconstruction

Thomas Lew
HKUST
Student ID: 20414520
Exchange student from EPFL
tjlew@connect.ust.hk

Abstract

In this project report, we describe the COMP5421 course's project on 3D Reconstruction. This paper also includes the difficulties encountered, assumptions and work to be done to improve the quality of the reconstruction. The most recent version of the code and instructions are released at: https://github.com/thomasjlew/Reconstruction3D.

1. Introduction

The 3D reconstruction project aims to recover a 3D scene from multiple images, as stated in [1]. All steps except features extraction were reproduced in C++ and their results were compared with OpenCV library. Matlab was used to analyse the results, prototype the functions, debug the algorithm using a simplified model and for data analysis in general. The operating system is Ubuntu 14.04.

The current dataset which is used for testing purposes is fountain-P11 [2]. The intrinsic parameters of the camera were extracted using RQ-decomposition of the camera matrix provided with this dataset.

2. Project organization

Each step of the project is executable independently in order to allow more flexibility and easier debugging and grading of the project. Most of the algorithms were first tested and developed using Matlab for its faster prototyping possibilities. Visualization scripts were written for faster analysis of the results obtained with the C++ implementation. More information can be found on the Github repository.

3. Features extraction

The first step of the algorithm consists of extracting points of interest from the images. For this purpose, different feature detectors and descriptors can be used and the OpenCV library offers a wide selection of these.

In the current version of this project, different feature detectors are implemented. SIFT, SURF, FAST and ORB detectors were tested successfully and can be selected from the *main* module of the program.

4. Features matching

In order to match corresponding extracted features, an algorithm was implemented to compute the Euclidean distance between each feature match and to compare these distances to obtain the best matches. To further improve the performance of the matcher, each best match for the first set of features is compared to the best match in the second one to ensure reliable matching. Since each distance needs to be computed in this algorithm, it is too computationally expensive and could not run in real time.

In the working implementation, the Fast Approximate Nearest Neighbor Search library, as described in [4], is used.

5. Epipolar geometry

The following step consists of computing the fundamental matrix from the matched features found in the previous task. A normalized 8-point algorithm was first developed using Matlab before being implemented in C++ in the final version. The absence of RANSAC framework in the implementation causes imprecisions which are mostly avoided by sorting the matches in order of precision before computing the fundamental matrix. The error over all extracted feature matches is computed:

$$\text{Error} = x_2^T * F * x_1$$

In order to make sure that the reconstruction is successful, a more advanced algorithm using RANSAC from OpenCV is also executed, before comparing the results and keeping the most precise one. In general, an error inferior to 1 leads to an acceptable reconstruction.

Once the fundamental matrix is computed, the Epipolar lines are drawn and can be visualized on each image pairs using the *geometry* command when executing the program.

6. Two-view triangulation

6.1. Camera and coordinate systems

Knowledge of the coordinate systems used in the implementation is crucial for successful triangulation. As stated before, RQ-decomposition is used to retrieve the intrinsic parameters from the camera matrix. After extracting the features with OpenCV, the used coordinate system for the image features was determined, as shown in Figure 1.

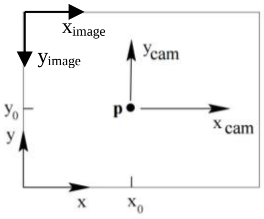


Fig. 1. Image (x_{image}, y_{image}) and camera (x_{cam}, y_{cam}) coordinate systems. Retrieved and modified from [3].

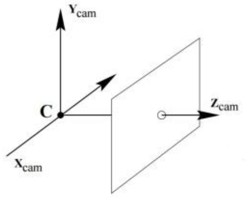


Fig. 2. Camera coordinate frame. Retrieved from [3].

Figure 2 above shows that compared to the image x and y axis, the camera coordinate frame's axes point in opposite direction than for the image. Hence, the intrinsic matrix is:

$$K = \begin{matrix} -2759.48 & 0 & 1520.69 \\ 0 & -2764.16 & 1006.81 \\ 0 & 0 & 1 \end{matrix}$$

The negative focal length in x and y directions express this difference in axis orientation.

6.2. Triangulation

The Direct Linear Transformation (DLT) [3] algorithm was implemented in Matlab[1] to recover the 3D features from two images. The first implementation does not include the normalization step, which is required to obtain better results. However, despite this, the results obtained in Matlab are visually as precise as the C++ version which uses OpenCV functions to perform triangulation (see 8. Results).

In order to perform triangulation, the relative translation and rotation between the pair of images needs to be computed from the essential matrix. Since there are 4 possible solutions to this problem, each one needs to be checked independently by reconstructing a point of interest and choosing the solution for which the reconstructed 3d point lies in front of both cameras. For this purpose, we chose the best possible match amongst all feature matches.

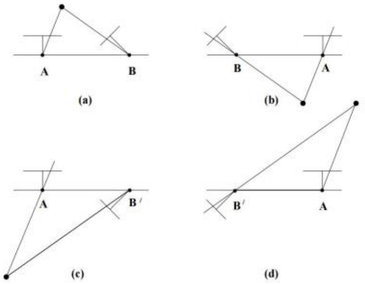


Fig. 3. The four possible solutions for calibrated reconstruction from E. [3]

These functions were implemented in both Matlab and C++ and work perfectly, up to a rotation of pi [rad]. To account for this imperfection which can cause problems in the final Multiview reconstruction, the sign of the elements of the rotation matrix are inversed in this case after the triangulation is performed, to ensure that the global rotation and translation between each frames is consistent. The assumption that no rotation of more than 180° occurs between two images is reasonable, since no matches between the two frames would be found if this wasn't the case. Also, the use of a scaler in section 7.2 allows more flexibility and corrects the sign of the translation vector.

7. Structure from motion

7.1. Projective transformations and scale

Since projective transformations are angle invariant but not distance invariant, recovering a 3D point from projective transformations of these features from an usual Euclidean space does not conserves the depth of these points. For this reason, it is impossible to reconstruct completely a 3D scene with real-world distances between points and camera centers. This is the main reason why most SLAM and odometry algorithms incorporate an Inertial Measurement Unit (IMU) and fuse its measurements with a camera system using a Kalman filter to recover the scale.

Hence, each two-view triangulation is defined up to a scale, recovering an object from multiple images requires to recover the scale of the translation vector between the two pairs of frames and to scale the depth of the 3D points. Since the absolute scale is unknown, we assume that the scale computed in the first image pair is the correct one and modify the following ones accordingly to precisely recover the 3D scene.

7.2. Recovering the scale

Since triangulated 3D features after two projection transformations conserve parallelism, we look for two scalars to recover the scale of the 3D features and of the translation vector[2].

We consider three cameras with their respective coordinate systems denoted with indices **0**, **1** and **2**. We denote $\mathbf{p_0}$ the triangulated 3D feature (or real world point) in the first frame, which coincides with the world frame. $\mathbf{R_{ij}}$ denotes a rotation matrix from frame **I** to frame **J**, $\mathbf{t_{ij,\ rec}}$ denotes the reconstructed translation vector between frames **I** and **J**, $\mathbf{p_{1,\ rec}}$ denotes the reconstructed 3D feature expressed in the first frame which corresponds to $\mathbf{p_0}$. The subscript ***homog*** denotes an homogeneous representation. Hence, we can express the reconstructed point in the world frame, or first frame n° 0, as:

$$p_{0,homog} = \bar{g}_{01}^{-1} \cdot p_{1,rec,homog}$$

With $\bar{g}_{01} = \begin{pmatrix} R_{01} & t_{01} \\ 0 & 1 \end{pmatrix} \in SE(3)$, $R_{01} \in S0(3)$ and t_{01} the real translation vector from frame 0 to 1.

Since $R_{01} \in S0(3)$, ${R_{01}}^{-1} = R_{10} = {R_{01}}^{T}$,

$$\bar{g}_{01}^{-1} = \begin{pmatrix} {R_{01}}^{T} & -{R_{01}}^{T} t_{01} \\ 0 & 1 \end{pmatrix}$$

Hence, we obtain the following equation

$$p_0 = {R_{01}}^{T} p_{1,rec} - {R_{01}}^{T} t_{01}$$

We introduce α and $\beta \in \mathbb{R}$ such that

$$p_1 = \alpha \cdot p_{1,rec}, t_{01} = \beta \cdot t_{01,rec}$$

We multiply both sides by R_{01} to obtain a linear equation expressed in the second frame n° 1,

$$R_{01} p_0 = \alpha \cdot p_{1,rec} - \beta \cdot t_{01,rec}$$

This forms three linear equations with two unknowns, α and β. We solve algebraically for these two values and obtain the two following equations, when using the x and y components:

$$\beta = \frac{[R_{01}p_0]_x [p_{1,rec}]_y - [R_{01}p_0]_y [p_{1,rec}]_x}{[p_{1,rec}]_x [t_{01,rec}]_y - [p_{1,rec}]_y \, [t_{01,rec}]_x}$$

$$\alpha = \frac{[R_{01}p_0]_x}{[p_{1,rec}]_x} + \beta \, \frac{[R_{01}p_0]_y}{[t_{01,rec}]_y}$$

[1] Because of a lack of time, the Matlab DLT triangulation algorithm was not ported to C++.

Please note that for this approach to be correct, the x and y component of these values must be different than zero. For these reason, they are tested before computing these scales. These equations are not unique and can be obtained from x and y projections as well.

The scalar β also accounts for the wrong sign in the translation part obtained when recovering the translation and rotation matrix. (See 6.2 Triangulation).

8. Matlab simple model to test algorithms

In order to test and debug the most important algorithms, a simple 3D model with features which positions are known was designed. For the exact 3D positions of these points, please refer to *init3dpts.m*.

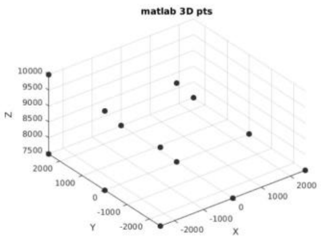


Fig 4. Simple model created in Matlab.

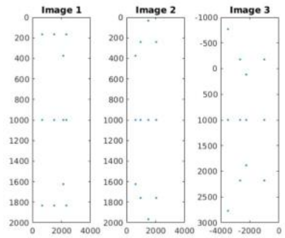


Fig. 5. Projections of these points for three camera poses.

The result of the complete 3D reconstruction, including α and β scaling is exactly the same as the initial model.

9. Results

9.1. Features extraction, matching and Epipolar geometry

Although various features detectors were implemented in this work, SIFT (implemented in OpenCV) is mostly used to obtain the results discussed in this paper.

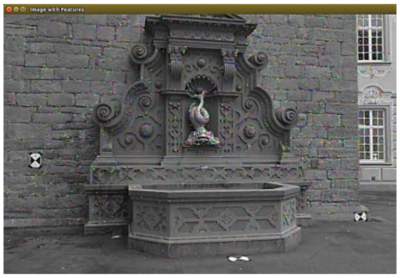


Fig. 6. Extracted features using SIFT.

After matching these features, computing the fundamental matrix and epipolar lines, the following 15 most precise matches were drawn, as can be seen in the following figure:

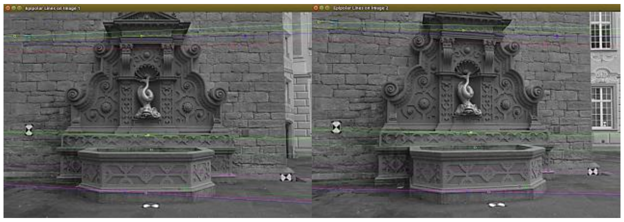


Fig. 7. Epipolar lines in images 5 and 6 of fountain_dense dataset [2].

The fundamental matrices in all images of the dataset were computed and different results were obtained. On images **5** and **6**, we obtained the following results:

$$F_{OpenCV} = \begin{matrix} -3.301\text{e}-09 & -4.835\text{e}-08 & 8.957\text{e}-06 \\ 4.540\text{e}-07 & 2.707\text{e}-08 & 5.667\text{e}-3 \\ -0.455\text{e}-3 & -6.438\text{e}-3 & 1 \end{matrix}$$

$$F_{Matlab} = \begin{matrix} -7.760\text{e}-09 & 1.411\text{e}-08 & -1.271\text{e}-04 \\ 3.860\text{e}-07 & 2.980\text{e}-08 & 0.0057 \\ -3.042\text{e}-04 & -0.0065 & 1 \end{matrix}$$

Hence, the implemented 8-point algorithm gives good results.

9.2. Two-view triangulation

After features extraction and matching, the features are saved into an external file and are used by the Matlab functions to reconstruct the image. Therefore, it is possible to compare the efficiency of the two implementations.

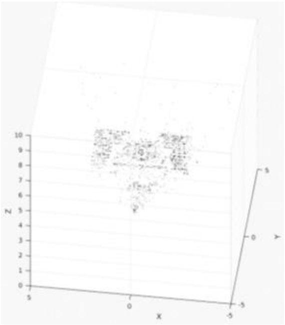


Fig. 8. Matlab triangulation

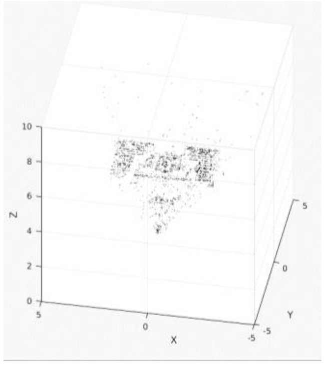
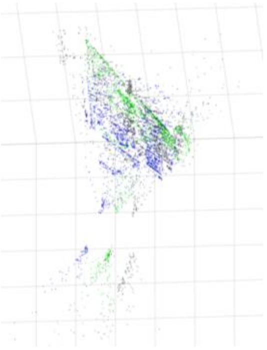


Fig. 9. C++ Triangulation

Visually, no notable difference can be observed on the two images. Please note that the features used in the reconstruction are exactly the same and come from the C++ code extracting and matching the features.

9.3. Structure from motion, Multiview 3D reconstruction

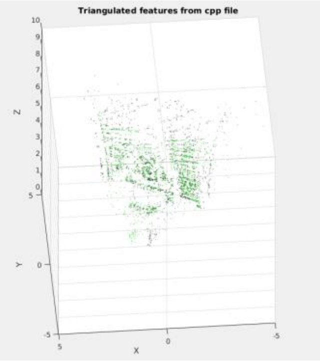
Because of the absence of a robust generalized RANSAC framework, the 3D reconstruction failed for images **8**, **9**, and **10,** since features far from the fountain on the building on the right side of the wall caused big errors in the computation of the fundamental matrix, even when using OpenCV functions with RANSAC.



This visualization shows that a three-view reconstruction without scale correction generates errors. The three cameras and views are coloured from right to left.

The camera centres and orientation can be seen because of outliers or mismatches.

Fig. 10. SFM without scale correction from images 1 to 4.



Here, we see that the scale is recoved with our new approach discussed in 7.2.

The black reconstructed features correspond to the first camera, or image pairs 2-3. The green coloured features correspond to the second camera.

Fig. 11. SFM with scale correction from images 2 to 4.

A full reconstruction using images **0** to **6** was also executed and the result can be seen in the following figure:

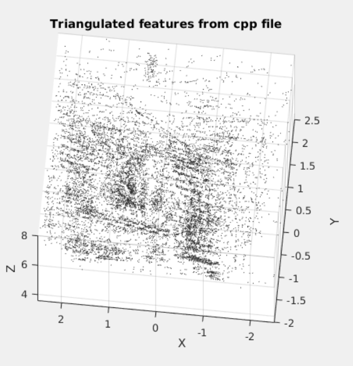


Fig. 12. Full reconstruction using images 0 to 6.

Although the precision is not very accurate yet, the fountain can be recognized with depth. The following graph shows that the depth of the point is not precise yet and that further investigations on the cause of this problem are still necessary to fix the full reconstruction, as discussed in *10. Optimization and further improvements*.

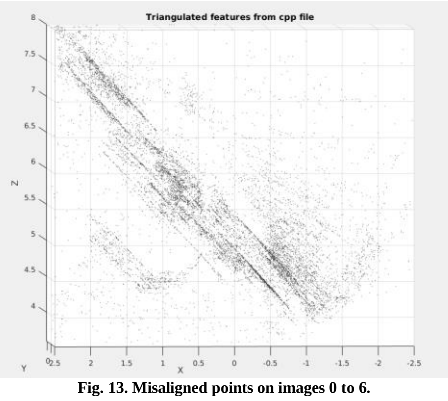


Fig. 13. Misaligned points on images 0 to 6.

Here are the computed α and β scalers computed for images **1** to **3** and **2** to **4**:

$$\alpha_{1,2,3} = 1.31713$$
$$\beta_{1,2,3} = -0.999402$$
$$\alpha_{2,3,4} = 0.988674$$
$$\beta_{2,3,4} = -0.998212$$

Finally, a comparison in the computation time between the Matlab and C++ implementation is shown below. This includes the full triangulation but without the features extraction and matching step, which is computationally expensive.

Time required for the two view reconstruction:
Matlab version: **1.73 sec.**
C++ version: **0.216 sec.**

This doesn't include the visualization which is usually done using Matlab.

10. Optimization and further improvements

First, it is necessary to improve the features matching algorithm, which is currently too computationally expensive. The Best Bin First algorithm should be able to cope with this problem very efficiently [5].

More importantly, a RANSAC framework should be implemented in both the Matlab and C++ version.

Also, the recovery of the rotation and translation between two frames should be improved, alongside with the triangulation, using Levenberg-Marquardt nonlinear least squares algorithms [6], as required in last year's version of the course. (http://www.cad.zju.edu.cn/home/gfzhang/training/SFM/SfM.html)

Finally, Bundle adjustment would also increase the accuracy of the overall reconstruction. Also, it would be useful to remove additional outliers which wouldn't have been removed using RANSAC.

Unfortunately, there was no time to change the code to a more usual C++ architecture with classes and to build a complete version running on ROS to test it in real time on a Raspberry Pi with an additional camera. The present code would require more advanced algorithms such as ORB for the features detection to speed up the reconstruction, since the triangulation of two images is currently too long (see Results which do not take into account features extraction).

11. Conclusion

In this project, a full reconstruction of a 3D scene given multiple images is made. This report showed that the resulting 3D points reconstruct the dataset although many imprecisions still occur. A Matlab version of this program is also released and complements the implementation, while completes the analysis of the project.

12. References

[1] Y. Yao, T. Shen, J. Wang. (2017, March). COMP5421 Computer Vision – Course Project. Retrieved from https://home.cse.ust.hk/~tshenaa/sub/COMP5421/comp5421_course_project.html

[2] C. Strecha, W. von Hansen, L. Van Gool, P. Fua, U. Thoennessen On Benchmarking Camera Calibration and Multi-View Stereo for High Resolution Imagery CVPR 2008.

[3] R. Hartley and A. Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press, ISBN: 0521623049, 2000.

[4] M. Muja, D. G. Lowe. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration, 2009.

[5] Jeffrey S. Beis and David G. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional

spaces," Conference on Computer Vision and Pattern Recognition, Puerto Rico (June 1997), pp. 1000-1006.

[6] M.I.A. Lourakis, levmar: Levenberg-Marquardt nonlinear least squares algorithms in C/C++, http://www.ics.forth.gr/~lourakis/levmar/+, Jul. 2004.