M4 - Week 5: Structure from Motion

Aditya Sangram Singh Rana, Juan Chaves, Carmen Garcia and German Barquero

Universitat Autònoma de Barcelona, Spain
{adityasangramsingh.rana, juanvictor.chaves, carmen.garciano,
german.barquero}@e-campus.uab.cat

This week's work focuses on 1) understanding how the pipeline of the structure-from-motion algorithm works and 2) applying it to a real set of images of a building facade. The structure-from-motion algorithms allows us to recover the camera matrices of each view and to generate a sparse set of 3D points. With enough overlapping views of the same scene, we should be able to reconstruct the 3D spatial structure of the scene. In our case, the structure is the facade of a building.

1 3D Reconstruction

We will be taking the "stratified" approach to reconstruction, where we begin with a projective reconstruction and then we refine it progressively to an affine and finally a metric reconstruction. The projective reconstruction between two uncalibrated images is computed by

(a) **Computing the fundamental matrix** from point correspondences $x_i \leftrightarrow x_i'$ between the images
(b) **Camera Retrieval**: computing the camera matrices P, P' from the fundamental matrix
(c) **Triangulation**: for each point correspondence $x_i \leftrightarrow x_i'$, compute a point X_i in space that projects to these imaging points

The result of the projective reconstruction can be seen in Fig. 1.

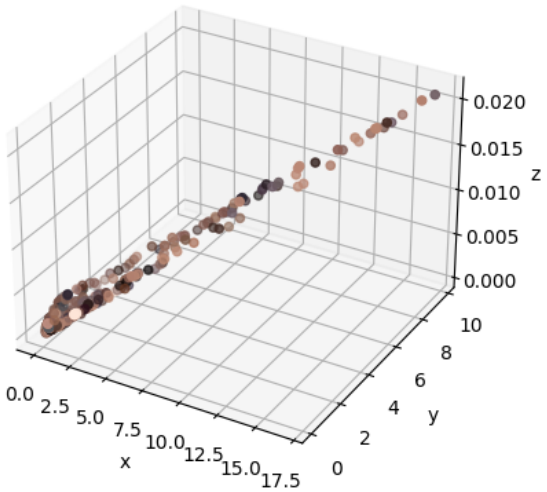

Fig. 1: Projective reconstruction

1.1 Affine reconstruction

The essence of an affine reconstruction is to locate the plane at infinity by some means, since this knowledge is equivalent to an affine reconstruction. Assuming we already know the projected plane at infinity, π_∞, we want to find the a transformation which maps it to the canonical plane at

infinity. This is, we want to find H such that $H^{-T}\pi = (0,0,0,1)^T$. Such a transformation is given by

$$H_{a\leftarrow p} = \begin{bmatrix} I_{3x3} \mid 0 \\ \pi^T \end{bmatrix} \tag{1}$$

The transformation H is now applied to all points and the two cameras as shown in Equation 2. The result of the affine reconstruction can be seen in Fig. 2.

$$P_i' = P_i H_p^{-1}, \quad H_p \mathbf{X} \tag{2}$$

For actually computing the plane at infinity, we will first compute points at infinity or vanishing points. Three points are enough for defining a plane, and so once we have computed three vanishing points the plane at infinity is obtained as the nullspace of a matrix whose rows are these three points (in homogeneous coordinates).

In order to obtain these 3D vanihsing points, we will need to not only find vanishing points, but also match them. For this, we can make use of the knowledge that 3D lines are in reality parallel. The intersection of the two parallel lines in space gives a point on the plane at infinity. The image of this point is the vanishing point of the line, and is the point of intersection of the two imaged lines. We compute a set of 3 vanishing points using Xiaohu Lu's [2] method, and use this to get the equation of the plane, as explained above.

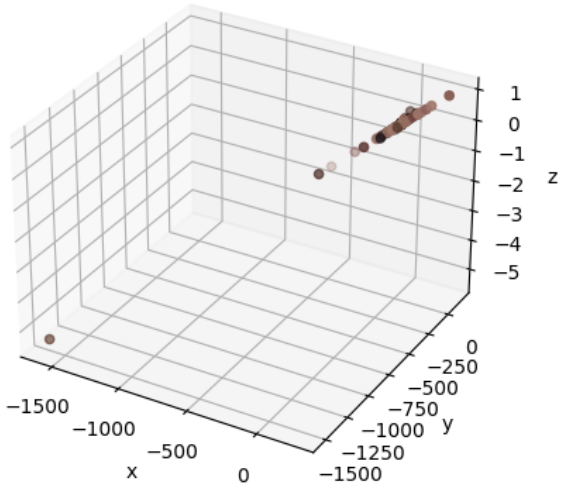

Fig. 2: Affine reconstruction

1.2 Metric reconstruction

The key to metric reconstruction is the identification of the absolute conic Ω_∞. The absolute conic, Ω_∞ , is a planar conic lying in the plane at infinity π_∞. The next step is to apply an affine transformation to the affine reconstruction such that the identified absolute conic is mapped to the absolute conic in the standard Euclidean frame (it will then have the equation $x_1^2 + x_2^2 + x_3^2 = 0$,on π_∞). We proceed by finding the image of the absolute conic in one of the images. In an affine reconstruction with a camera matrix given by $P = [M \mid m]$, the metric reconstruction can be achieved by applying a 3D transformation of the form

$$H_{e\leftarrow a} = \begin{bmatrix} A^{-1} & 0 \\ 0^T & 1 \end{bmatrix} \tag{3}$$

where ω the image of the absolute conic and A is obtained by Cholesky factorization $AA^T = (M^T \omega M)^{-1}$. This approach relies on computing ω which can be done by applying different kinds of constraints

1. **Constraints coming from scene orthogonality:** If v_1 and v_2 is a pair of vanishing points arising from orthogonal scene lines, then we have a linear constraint on ω that $v_1^T \omega v_2 = 0$ (This gives us 3 equations from each of the vanishing points u,v,z)
2. **Constraints coming from known internal parameters:** Since ω can be written as $K^{-T}K^{-1} = \begin{pmatrix} \omega_{11} & \omega_{12} & \omega_{13} \\ \omega_{12} & \omega_{22} & \omega_{23} \\ \omega_{13} & \omega_{23} & \omega_{33} \end{pmatrix}$, the knowledge about some restrictions on the internal parameters contained in K may be used to constraint or determine the elements of ω. For example, zero skew implies $\omega_{12} = 0$, and if skew is zero and $\alpha_x = \alpha_y$, then $\omega_{11} = \omega_{22}$

The five constraints can be stacked up to form a matrix s.t. $A\omega_v = 0$ where $\omega_v = (\omega_{11}, \omega_{12}, \omega_{12}, \omega_{22}, \omega_{23}, \omega_{33})$. The solution ω_v is the null vector of A. After applying $H_{e \leftarrow a}$,the result of the metric reconstruction can be seen in Fig. 3.

$$A = \begin{pmatrix} u_1v_1 & u_1v_2 + u_2v_1 & u_1v_3 + u_3v_1 & u_2v_2 & u_2v_3 + u_3v_2 & u_3v_3 \\ u_1z_1 & u_1z_2 + u_2z_1 & u_1z_3 + u_3z_1 & u_2z_2 & u_2z_3 + u_3z_2 & u_3z_3 \\ z_1v_1 & z_1v_2 + z_2v_1 & z_1v_3 + z_3v_1 & z_2v_2 & z_2v_3 + z_3v_2 & z_3v_3 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

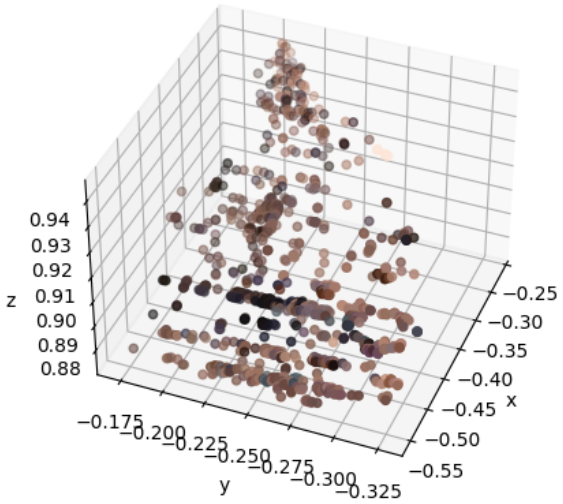

Fig. 3: Metric reconstruction

2 Estimations

2.1 First projective camera

The general formula for a pair of canonic camera matrices corresponding to a fundamental matrix F are given by

$$P = [I \mid 0] \text{ and } P' = [SF \mid e'] \tag{4}$$

where S is any skew-symmetric matrix (such that P' has rank 3). A good choice is $S = [e']_x$, where the epipole can be computed from $e'^T F = 0$. So the first camera matrix is implicitly assumed to be $P = [I \mid 0]$ and the other is calculated using the above formulation.

2.2 Reprojection error

The re-projection error is computed as:

$$error = \underbrace{||PX - \mathbf{x}||}_{\text{Error in Image 1}} + \underbrace{||P'X - \mathbf{x}'||}_{\text{Error in Image 2}} \tag{5}$$

With this in hand, we can try to measure the performance of the different steps of our algorithm.

The original re-projection error is of around 5e-3 already very small, even if visually we cannot tell whether the reconstruction is a good one or not (Figure 2). After applying metric rectification, points seen to be more aligned, as if in a wall (Figure 3). However, re-projection error increases drastically, reaching around 1e4. This could point to some kind of error of implementation on that section of the code.

3 Bundle adjustement

In order to refine our results, Bundle Adjustment will be applied. This algorithm aims at minimizing the re-projection error (Equation 6) by refining both the camera calibration and 3D point estimation. For this, we employ the Python library pySBA [6].

3.1 Preparation

Throughout the whole program, a close tracking of detected points is carried out through means of *tracks*. Each *track* holds information about a 3D point and all of its 2D observations from different cameras. Initially, this is designed to work for 2 images, but can be adapted to an arbitrary number.

$$\min_{\mathbf{R}^i, \mathbf{X}_p} \sum_{i=1}^{i} \sum_{p=1}^{P} d(\mathbf{R}^i \mathbf{X}_p, \hat{\mathbf{x}}_p^i)^2 \tag{6}$$

We need to transform our data (both from tracks and current camera calibration) to the library's format. The PySBA class requires the following input:

1. *Camera arrays*. An array for each camera containing info about translation, rotation, focal length and distorsion. As we did not have full access to calibration data (i.e., distorsion metrics) we assumed we had no distorsion.
2. *3D points*. The 3D point corresponding to the observation, as obtained from our tracks.
3. *2D points*. The 2D point (on the image specified in the parameter *Camera Index*) corresponding to the observation, as obtained from our tracks.
4. *Camera Index*. Index representing which camera observed each point.
5. *Point 2D Index*. Index representing the correspondence between 2D and 3D points.

All of this information was either stored in our tracks, known from a previous calibration of the cameras or obtained from the computed fundamental matrix.

3.2 Results

Once all data is in the correct format, pySBA is able to perform bundle adjustment. The algorithm runs for around a hundred iterations and greatly reduces the initial cost. Figure 4 displays points after this algorithm has been applied. Due to the sparsity of the data and means of visualization, it is hard to assess the quality of the reconstruction. We do seem to have a clear corner, which should coincide with where the two facades meet. Should this be the case, this would clearly show the power of bundle adjustment, as on the previous reconstruction it was hard to discern any structure.

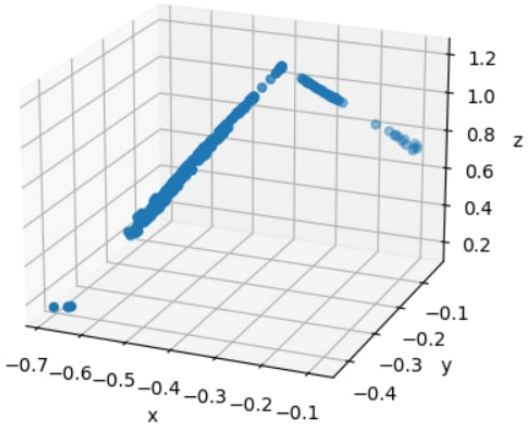

Fig. 4: Result after bundle adjustment

4 Affine homography estimation

An alternative method for affinely rectifying an image (as was done on lab1) is presented. Now, instead of starting from two sets of hand picked parallel lines, we will use three point correspondences and the fundamental matrix.

The homography is given by the following formula:

$$H = A - \mathbf{e}'(M^{-1}\mathbf{b})^T \tag{7}$$

Where $A = [\mathbf{e}']_x F$, $\mathbf{b}$ is a 3-vector with components:

$$b_i = \frac{(\mathbf{x}_i' \times (A\mathbf{x}_i))^T(\mathbf{x}_i' \times \mathbf{e})}{||\mathbf{x}_i' \times \mathbf{e}||^2} \tag{8}$$

and M is a 3 by 3 matrix with rows $\mathbf{x}_i^T$
Results can be seen in Figure 5.

Fig. 5: Affinely rectified images. Parallel lines are not perfectly so, but result still is reasonably good in comparison to lab 1's if we take into account this method is automatic. This comes at the price of dependence on the 3D reconstruction.

The clear advantage of this method over the one presented in lab 1 is that it is performed automatically, without need of human interaction. Results do not seem to be as accurate, but they were obtained with the first estimation of 3D points (which can be quite noisy), so we estimate this method can perform equally well with a refined 3D set.

5 Resection method

To extract the camera projection matrix from the two first views, we compute the fundamental matrix using sufficient keypoints correspondences. This lets us reconstruct the first set of tracked points in the 3D world. Once we already have an initial 3D reconstruction of our set of tracked points, we can use another method to extract the camera matrix of the new view: resectioning. The idea is similar to the one applied in *Week 2* where we had to compute a 2D projective transformation between images (DLT algorithm). In this case, we need sufficient (6) 2D to 3D correspondences, $X_i \leftrightarrow x_i$, so that the camera matrix P can be determined.

The equations of the homogeneous system that need to be solved are very similar to the ones for the DLT algorithm but with a matrix P of dimensions 3x4 instead of 3x3.

$$\begin{bmatrix} 0^T & -w_i X_i^T & y_i X_i^T \\ w_i X_i^T & 0 & -x_i X_i^T \end{bmatrix} \begin{pmatrix} P^1 \\ P^2 \\ P^3 \end{pmatrix} = 0 \tag{9}$$

From a set of n correspondences, we obtain a $2n$x12 matrix by stacking up the equations. As in previous weeks, the set of correspondences that we obtain is noisy, so we may have outliers that may cause an instability. This may lead to unrecoverable states of the structure from motion algorithm. To increase the robustness of the method we use the RANSAC version of the DLT algorithm instead. As an extra refinement step, a minimization of the geometric error is applied with an iterative technique such as Levenberg-Marquardt. In this method, the DLT solution is used as a starting point. It is important to note that a normalization is applied to both the 2D and 3D points in order to ensure the stability of the solutions found. This normalization scales them so that their root-mean-squared distance from the origin is $\sqrt{2}$. Finally, the projection matrix is denormalized.

Although we would have liked to show some results for this method, we did not have enough time this week to scale the code so it works with more than 2 images. Therefore, resection could not be properly tested (although partial results were and the code is available in the *resection.py* file).

6 Improvements

6.1 Tracks structure improvement

Sadly, we did not have time to implement this optional task.

6.2 Strategies for pipeline improvement

In this section, we present a discussion on how we would theoretically improve the pipeline of the presented SfM algorithm so it becomes more robust and efficient in a hypothetical deployment scenario. The ideas come from the work of Schnberger et al. [3]. We will first analyze the methodological improvements that could be included, then we will briefly explain some techniques to refine the results once the algorithm finishes and, finally, we give some hints on how to increase the computational performance of the algorithm.

Initialization. We know that an initial bad 3D world reconstruction may be fatal for the SfM algorithm due to its incremental nature. Therefore, choosing a suitable pair of images for the first initialization is crucial to ensure the good performance of the algorithm [4][5]. Thus, if we worked in an offline way, we would ensure that the first processed images correspond to the most dense location in the image, with the highest amount of overlapping cameras possible, so that the reconstruction in the first stages is accurate. This way, we would minimize the error in the incremental process.

Next best view. Also, when dealing with SfM, we may have no information of the images used for the reconstruction, as they may come from the Internet or unknown sources. Given the importance of the triangulation accuracy, for which errors could accumulate and lead to a cascade of mis-registrations and therefore to faulty triangulations, the decision of choosing the next best view is important. The method proposed in [3] to solve this problem is based on quantizing the image into cells which may be flagged for each pair of images as *full*, if there is a match inside, or as *empty*, if there is not. These flags are considered to extract a score and choose then the best next view. This could even be adapted to work in a multi-scale fashion, see Fig. 6 for a visual example.

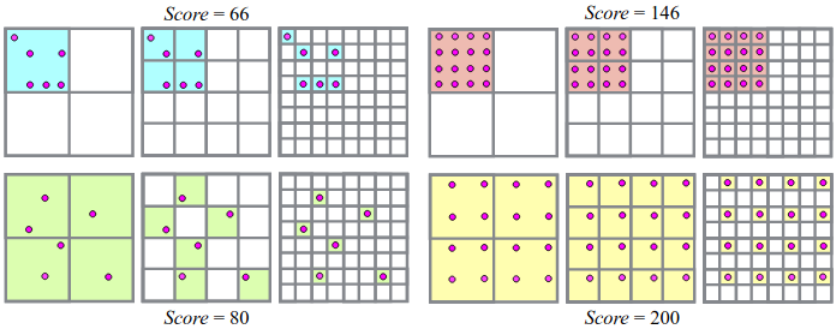

Fig. 6: Example of multi-scale grid-based scores for four candidates in the next best view selection method proposed.

Distinct tracks fusion. The problem derived from fusing distinct tracks is very critical too. This is due to ambiguous matches along the epipolar line in the two-view verification process. In [3], they propose a very efficient and sampling-based triangulation method to robustly estimate all points within a feature track contaminated with outliers. It considers the two constraints of a well-conditioned model: a sufficient triangulation angle and positive depths with respect to the views. Then, it applies iteratively the RANSAC method with the reprojection error to get the most robust estimations for the feature tracks in the new view.

Post-processing refinement. Once we have the result, we can apply other methods to refine or complete the results obtained in a post-processing stage. Such methods include, for example, the re-triangulation of tracks that previously failed to triangulate due to inaccurate poses. Re-triangulation, bundle adjustement and filtering can be applied in an iterative optimization until the number of filtered observations diminishes (we will have removed all outliers).

Computational performance boost. In terms of computational performance, the bundle adjustement step is highly demanding when applying SfM to big sets of images. To decrease the computational load, we can partition the scene into many small and highly overlapping camera groups. For each group, a single camera is considered so that the cost of solving the camera system is reduced. Also, the graph-based algorithms used in most of the state-of-the-art approaches are simplified with such clustering. The key requirement for this to work is that images within a group are as redundant as possible, which can be measured as the number of co-visible points between images. Thus, this metric can be used to generate the groups. In [3], they presume that the computational benefit is notable even for groups of size 2.

7 Discussion

This week we tried to solve a very simple structure-from-motion scenario, focusing in the case of just two images. However, the method could be adapted to an arbitrarily big case with ease, as all computations are performed with only two images. Modifications could be made in order to use, for example, an sliding window of pictures, depending on desired accuracy and computation power. Some of the steps and algorithms in this lab, like the estimation of the affine homography or the resection method, greatly resemble some the work done in previous labs (namely labs 1 and 2), being almost *mere* extensions of the same ideas to a 3D world.

A clear challenge in this sort of problems, is the qualitative assessment of results. Specially with as little input as two images, it is hard to visualize the computed point cloud and yield conclusions. Judging mainly by quantitative measures, like the re-projection error, our current implementation shows some issues in the metric rectification part, as this error increases dramatically. Bundle adjustment seems to refine the point cloud correctly (once again, it is difficult to interpret), but does take a big enough computation time as for realtime operation to be unfeasible. This, of course, depends immensely on the implementation and is probably not a problem in more *serious* implementations.

An interesting application of the tools developed for the last two labs is the automatic estimation of the affinity relating two images. As it relies on the accuracy of the pointcloud, this method is not as robust as the one presented in the first lab.

Although the given implementation of the structure from motion algorithm works well within the already mentioned limitations, we found state-of-the-art methods which could improve it. These pipeline enhancements consist in finding the most adequate initial pair of views and the best next view at each step, thus ensuring the proper stability of the algorithm. Also, there are novel pipeline improvements focused on trying to make SfM more efficient. This is critical when the deployment scenario leverages big sequences of images or the structure needs to be reconstructed in real time.

References

1. Hartley, Richard and Zisserman, Andrew. Multiple view geometry in computer vision. Cambridge university press, 2003.
2. Lu, Xiaohu and Yao, Jian and Li, Haoang and Liu, Yahui and Zhang, Xiaofeng, 2-Line Exhaustive Searching for Real-Time Vanishing Point Estimation in Manhattan World, IEEE Winter Conference on Applications of Computer Vision (WACV), March, 2017
3. J. L. Schnberger and J. Frahm, "Structure-from-Motion Revisited," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 4104-4113.
4. C. Beder and R. Steffen. Determining an initial image pairfor fixing the scale of a 3d reconstruction from an image sequence. Pattern Recognition, 2006.
5. N. Snavely. Scene reconstruction and visualization from internet photo collections. PhD thesis, 2008.
6. @jahdiel. pySBA - Python Bundle Adjustment, Github repository - https://github.com/jahdiel/pySBA Accesed on 01 February 2018.