3D Point Triangulation and Bundle Adjustment
COMP 776, Fall 2017
Due: November 1, 2017
Assignment Data: https://drive.google.com/open?id=0BzP2GHNy5xrfRlNOb1IteUkwSnc

Summary

The goal of this assignment is to give you experience with the final step of Structure from Motion (SfM): taking the 2D points detected, matched, and verified in the previous two assignments and turning them into points in 3D space. For this assignment, you will be implementing linear 3D point triangulation; you will not need to implement bundle adjustment. Python code (described below) has been provided for this assignment, along with data for you to experiment on.

Assignment

- Coding: Complete the *triangulation.py* file included in the code. Submit your code and a visualization of the results.
- Read and understand: The sections below (in addition to the provided code) outline the general steps to go from 2D correspondences to 3D points. We've also pointed you to optional related reading. Use these resources to help you understand what the code from the assignment is doing.
- Project proposals (single-page write-up due October 30): Start thinking about what your project proposal for the class will be. Some ideas are at the end of this document.

The rest of this document

Code

There are five files provided in this assignment. You only need to edit *triangulation.py*.

- *main.py*: For command-line usage, execute "*python main.py --help*". This file loads the camera calibration K, the corresponding image keypoints, and the fundamental matrix F for the two images. It then computes E and decomposes the matrix into four possible pose solutions. Points will be triangulated for each pose using your 3D triangulation implementation, and the best relative pose will be selected. Bundle adjustment is then applied to the solution, and the results are saved.

- *triangulation.py*: This file contains a single function that you will need to implement.
  - *triangulate_points()* : Given two arrays of corresponding keypoints (in normalized camera coordinates) and a projection matrix P for the second image w.r.t. the first, iterate through each correspondence and triangulate it using the linear triangulation method introduced below. Return the final set of 3D points.

- *bundle_adjustment.py*, *quaternion.py*: These files contain functions relevant to bundle adjustment. See the code comments for implementation details.

- *util.py*: Contains functions for saving PLY point clouds and camera meshes.

Data and Visualization

This assignment uses the same images as the F-matrix RANSAC assignment. The camera calibration and fundamental matrix are provided, along with inlier corresponding keypoints.

The code is currently set up to output the triangulated points to PLY files, which are easy-to-use files for 3D meshes and point clouds. In addition to the point clouds, the code saves PLY meshes for the cameras. You can visualize these files in programs such as MeshLab or Blender. There is commented code at the end of *main.py* that visualizes the point cloud in matplotlib, but we have found that 3D viewer to be near-unusable.

Submission

Submit a single PDF containing views of your 3D reconstruction before and after bundle adjustment (include both point clouds in each image). Show the results from left, right, and top-down views with the differences clearly visible,[1] and include the output camera meshes in at least some of the views. Also in your PDF, include the definition of your *triangulate_points()* function implementation (if possible: monospace text is preferred, instead of a picture).

In your PDF, please include your name and a link to your code in your department Google Drive account. Be sure to share the folder with Jan-Michael Frahm, Marc Eder, and True Price.

[1] In MeshLab, you may need to disable shading on the 3D point clouds, and you might need to increase the point size. Also, consider rendering the cameras with edges highlighted, so that you can clearly see the geometry.

<u>Related Reading (optional; the sections below try to give you the important points)</u>

- Hartley and Zisserman 9.6, 12.1, 12.2, A5.3 (available via library.unc.edu)
- Szeliski 7.1, 7.2
- Triggs *et al.*, "Bundle Adjustment – A Modern Synthesis." *International Workshop on Vision Algorithms*, pp. 298-372. Springer, Berlin, Heidelberg, 1999. [link]
- ceres-solver documentation (an excellent non-linear least-squares C++ library)

<u>Overview</u>

Because several steps in this assignment are non-trivial to implement, your task is only to write a function performing 3D point triangulation given: 1) a set of corresponding keypoints in two images and 2) a fundamental matrix relating the two images. For completeness, this assignment will cover all remaining parts of the two-view reconstruction pipeline, and you should familiarize yourself with the series of steps involved.

Recall that incremental Structure from Motion consists of the following steps:

1) Extract features in a set of images
2) Match features between pairs of images
3) Geometrically verify the matches (*e.g.*, through F-matrix RANSAC)
4) Triangulate 3D points from the inlier matches in an initial image pair
5) Refine the initial 3D points and camera parameters using bundle adjustment
6) Repeatedly register new images to the reconstruction (*i.e.*, solving Perspective-n-Point)
7) Triangulate additional 3D points from the newly placed cameras
8) Periodically perform bundle adjustment to refine the newly placed cameras and 3D points

The previous two assignments covered steps 1-3. This assignment covers steps 4 and 5. In particular, our task consists of the following sub-steps:

1) Recover an essential matrix from the initial image pair
2) Extract 4 possible P matrices for the image pair (recall $P = [R \mid \boldsymbol{t}]$)
3) For each P hypothesis, triangulate 3D points for all correspondences
4) Select the P hypothesis with the largest number of points in front of the cameras
5) Jointly optimize P; the set of 3D points; and (optionally) the camera intrinsics

The Essential Matrix

Up to this point in the class, we have focused mainly on the fundamental matrix F, which maps points in one image into *epipolar lines* in a second image. The essential matrix E is the *calibrated* version of F:

$$F = K_2^{-T} E K_1^{-1},$$

where K_1 and K_2 are the intrinsic camera matrices for the first and second images, respectively, and $K_2^{-T} = (K_2^{-1})^T$. We say E is calibrated because it operates on normalized camera coordinates (*i.e.*, $\tilde{\boldsymbol{x}} = K^{-1}\boldsymbol{x}$ for pixel coordinate $\boldsymbol{x}$ and known camera intrinsics). The epipolar constraint for E is directly analogous to F:

$$\boldsymbol{x}_2^T F \boldsymbol{x}_1 \;=\; \boldsymbol{x}_2^T (K_2^{-T} E K_1^{-1}) \boldsymbol{x}_1 \;=\; (\boldsymbol{x}_2^T K_2^{-T}) E (K_1^{-1} \boldsymbol{x}_1) \;=\; \tilde{\boldsymbol{x}}_2^T E \tilde{\boldsymbol{x}}_1 \;=\; 0.$$

Constructing *E* from Relative Pose

The pose (relative rotation R and translation $\boldsymbol{t}$) of the second image relative to the first can be directly used to construct E (see Szeliski 7.2). To motivate this, consider the 3D point X_1 projecting to the point $\boldsymbol{x}_1$. From projective geometry, we know that $X_1 = z_1 \tilde{\boldsymbol{x}}_1$ for some depth value z_1. (Note that because the pose is relative, the first camera has no translation or rotation.)

Similarly, the same 3D point can be expressed relative to the second camera, instead of the first:

$$X_2 \;=\; z_2 \tilde{\boldsymbol{x}}_2 \;=\; R X_1 + \boldsymbol{t} \;=\; R(z_1 \tilde{\boldsymbol{x}}_1) + \boldsymbol{t}.$$

Now, consider taking the cross product of the above equations with $\boldsymbol{t}$. First, let's simplify some notation by rewriting the cross product operator as a matrix multiplication. For some vector $\boldsymbol{v}$,

$$\boldsymbol{t} \times \boldsymbol{v} \;=\; \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \boldsymbol{v} \;=\; [\boldsymbol{t}]_\times \boldsymbol{v}.$$

Then, the cross product with $\boldsymbol{t}$ is

$$[\boldsymbol{t}]_\times (z_2 \tilde{\boldsymbol{x}}_2) \;=\; [\boldsymbol{t}]_\times (R(z_1 \tilde{\boldsymbol{x}}_1) + \boldsymbol{t}) \;=\; z_1 [\boldsymbol{t}]_\times R \tilde{\boldsymbol{x}}_1.$$

Further taking the dot product with $\tilde{\boldsymbol{x}}_2$:

$$z_2 \tilde{\boldsymbol{x}}_2^T [\boldsymbol{t}]_\times \tilde{\boldsymbol{x}}_2 \;=\; z_1 \tilde{\boldsymbol{x}}_2^T [\boldsymbol{t}]_\times R \tilde{\boldsymbol{x}}_1.$$

But, on the left side, note that $[\boldsymbol{t}]_\times \tilde{\boldsymbol{x}}_2$ is orthogonal to $\tilde{\boldsymbol{x}}_2$ – the left side equals zero! Dividing out the z_1 on the right side, we conclude that

$$\tilde{\boldsymbol{x}}_2^T [\boldsymbol{t}]_\times R \tilde{\boldsymbol{x}}_1 \;=\; 0,$$

which leads us to the pose construction of the essential matrix:

$$E = [\boldsymbol{t}]_\times R.$$

Recovering Pose from the Essential Matrix

(Szeliski 7.2) and (Hartley and Zisserman 9.6) address the decomposition of E into R and $\boldsymbol{t}$. It is not necessary for you to understand the exact math behind the decomposition, but we will provide a general overview, here.

The essential matrix has two singular values that are equal and one singular value that is zero. (The F matrix is also rank 2, but its singular values are not necessarily equal.) As such, the SVD of E with singular value σ has the form

$$E = U \begin{bmatrix} \sigma & 0 & 0 \\ 0 & \sigma & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T.$$

Define the matrix W as

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

It turns out (H & Z 9.6.2) that there are 4 possible values of the pose matrix $P = [R \mid \boldsymbol{t}]$:

$$P = [UWV^T \mid \boldsymbol{u_3}] \quad \text{or} \quad [UW^TV^T \mid \boldsymbol{u_3}] \quad \text{or} \quad [UWV^T \mid -\boldsymbol{u_3}] \quad \text{or} \quad [UW^TV^T \mid -\boldsymbol{u_3}],$$

where $\boldsymbol{u_3}$ is the third column of U. Note that the relative pose is only determined up to scale, so E only has 5 degrees of freedom: 3 for rotation and 2 for translation. The decomposition above naturally results in $\|\boldsymbol{t}\| = 1$.

The sign ambiguity of $\boldsymbol{t}$ (*i.e.*, $\pm\boldsymbol{u_3}$) is somewhat intuitive: E and $-E$ both satisfy the epipolar constraint. Concerning $R = UWV^T$ versus $R = UW^TV^T$, the two possible values are related by a $180°$ rotation around $\boldsymbol{t}$. The figure below (reproduced from Hartley and Zisserman) demonstrates the four possible configurations.

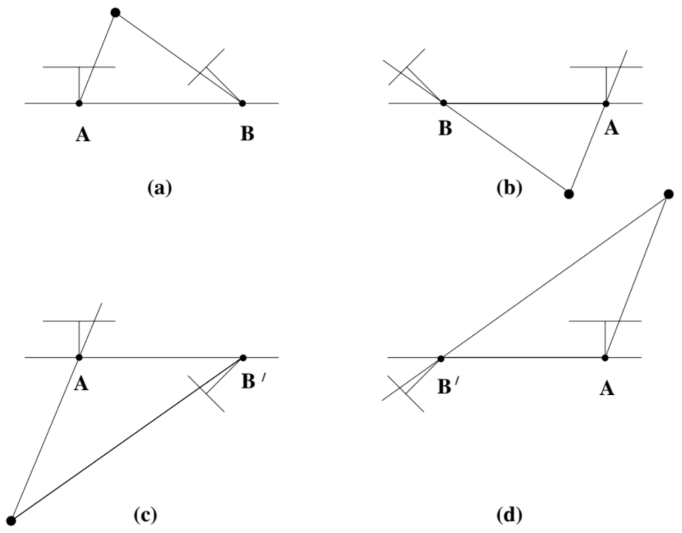

Figure 1. (Fig. 9.12 from Hartley and Zisserman) Illustration of the 4 possible pose configurations for a given essential matrix. The two cameras are denoted as **A** and **B**. Recall that $P_A = [I \mid \boldsymbol{0}]$ and $P_B = [R \mid \boldsymbol{t}]$. The rows correspond to the two possible values for R, and the columns correspond to the two possible values for $\boldsymbol{t}$.

Recovering Pose from the Fundamental Matrix with Known Camera Calibration

If we have an estimated fundamental matrix for two images and known camera intrinsic matrices, it is straightforward to compute E:[2]

$$E = K_2^T F K_1.$$

Then, we can simply recover pose from E as outlined above.

3D Point Triangulation

In this assignment, we will cover a linear method for 3D point triangulation (Hartley and Zisserman 12.2). This approach is generally applicable to points observed in arbitrarily many views (not just two), with each camera having known pose and calibration.

Consider a camera with pose P_c operating on a homogeneous 3D point X in world coordinates:

$$P_c X = z_c \tilde{\boldsymbol{x}}_c,$$

where $\tilde{\boldsymbol{x}}_c$ is the normalized camera coordinate of X when it is projected into the camera, and z_c is the associated depth of the point in the camera coordinate frame. If we then take the cross product with $\tilde{\boldsymbol{x}}_c$, we arrive at

$$\tilde{\boldsymbol{x}}_c \times P_c X = \begin{bmatrix} 0 & -1 & \tilde{y}_c \\ 1 & 0 & -\tilde{x}_c \\ -\tilde{y}_c & \tilde{x}_c & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_{c,1}^{T} \\ \boldsymbol{p}_{c,2}^{T} \\ \boldsymbol{p}_{c,3}^{T} \end{bmatrix} X = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

where the equality with $\mathbf{0}$ comes from $\tilde{\boldsymbol{x}}_c \times \tilde{\boldsymbol{x}}_c = \mathbf{0}$, and we have split P_c into a separate vector per row.

Suppose we have a set of 2D (normalized) observations of X in C calibrated cameras, *i.e.*, $\{\tilde{\boldsymbol{x}}_c\}_{1 \le c \le C}$. Then, we can build up a linear system of homogeneous equations just as we did with the direct linear transformation (DLT) for F-matrix estimation:

$$AX = \begin{bmatrix} -\boldsymbol{p}_{1,2}^{T} + \tilde{y}_1 \boldsymbol{p}_{1,3}^{T} \\ \boldsymbol{p}_{1,1}^{T} - \tilde{x}_1 \boldsymbol{p}_{1,3}^{T} \\ -\tilde{y}_1 \boldsymbol{p}_{1,1}^{T} + \tilde{x}_1 \boldsymbol{p}_{1,2}^{T} \\ \vdots \end{bmatrix} X = \mathbf{0}.$$

Every three rows in A contain equations for a different camera c. Just as in F-matrix estimation, we can simply compute the optimal value X by taking the last column of V in the SVD of A.

Remember: X is a homogeneous 3D coordinate, meaning its final value should have the form $(X_x, X_y, X_z, 1)$.

[2] There are methods for obtaining reasonable initial values of E when calibration is unknown, but we will not address them here.

Also note: We need at least two cameras in order to obtain the minimum number of linear equations (*i.e.*, we need 4 rows in A). In fact, we only need two equations per image; in practice, it is not necessary to include the third equation ($-\tilde{y}_c \boldsymbol{p}_{c,1}^{T} + \tilde{x}_c \boldsymbol{p}_{c,2}^{T}$). You may exclude such rows in your implementation.

Bundle Adjustment

The driving idea behind bundle adjustment is to minimize *reprojection error*. Say we have triangulated a 3D point X_i that is observed in a set of multiple cameras C_i, and that we have a pose P_c for each camera. Moreover, let $\pi_c(P_c X_i)$ denote the function for projecting X_i into pixel coordinates for camera c. $\pi_c(\cdot)$ may include parameters for radial distortion, as well as focal length and principal point. The reprojection error for X_i in camera c is the squared 2D distance between the projection of X into c and the observed 2D keypoint $\hat{\boldsymbol{x}}_{\boldsymbol{c}}^{(i)}$:

$$r(X_i, c) = \left\| \pi_c(P_c X_i) - \hat{\boldsymbol{x}}_{\boldsymbol{c}}^{(i)} \right\|^2.$$

Bundle adjustment seeks to minimize the total reprojection error over all 3D points $\{X_i\}$ and all cameras, potentially with a robust weighting function $\phi(\cdot)$ to avoid outliers:

$$\mathcal{R}(\{X_i\}, \{\boldsymbol{\theta_c}\}) = \sum_i \sum_{c \in C_i} \phi(r(X_i, c)),$$

where $\boldsymbol{\theta_c} = [R_c, \boldsymbol{t_c}, f, c_x, c_y, k_1, \ldots]$ denotes the set of parameters for camera c. In the most general case, all parameters $\boldsymbol{\theta_c}$ are non-constant, except for $(R_0, \boldsymbol{t_0}) = (I, \boldsymbol{0})$, which is fixed because the global world frame is arbitrary. Since the reconstruction is also scale-independent, it is also common to fix $\|\boldsymbol{t_1}\| = 1$.

$\mathcal{R}(\{X_i\}, \{\boldsymbol{\theta_c}\})$ is minimized w.r.t. all 3D point locations and all camera parameters using gradient descent-type approaches. If some camera parameters are known (*e.g.*, the intrinsics), some values of $\boldsymbol{\theta_c}$ might be fixed. Also, if multiple images were taken with the same physical camera, it is possible to reuse $\boldsymbol{\theta_c}$ for each image.

Bundle Adjustment: Implementation Details for this Assignment

Building a bundle adjuster is not necessarily a simple task for someone new to computer vision. As such, we have provided a working NumPy/SciPy implementation of a two-view bundle adjuster; our hope is that this code reinforces the general introduction to bundle adjustment given in class and above. You are encouraged to explore the topic in more detail. A few notes on the provided bundle adjuster:

- R_1 is parameterized as a 4-value unit quaternion, which is simpler to optimize.
- Camera intrinsics are shared between the two images, and only the focal length f is optimized (c_x and c_y are fixed). The optimizer also fits a k_1 radial distortion parameter.
- *Sparsity*: The reprojection error for each 3D point only depends on the associated camera parameters and the values for X_i. To speed up the optimization, the code explicitly computes the sparsity of the problem and passes it to SciPy's *least_squares()* function.

Possible Related Class Projects

Since class project proposals are due soon (Monday, October 30), here are a few related topics to the current assignment that you might consider exploring:

- Marker-less motion capture (several datasets available): Recent advances in neural networks have allowed us to automatically detect the joints of a human (feet, knees, elbows, etc.) in an image. If we have a set of synchronized, calibrated cameras set up around the edges of a room, we can perform a 3D triangulation of the joints detected in the individual images at any given time. Areas to explore include performing this triangulation robustly and incorporating temporal constraints on the 3D motion.

- Stereo vision (many datasets available, we'll cover this topic soon in class): Given a calibrated pair of synchronized cameras that are side-by-side, the goal of stereo is to triangulate each pixel in the two images. The challenges mainly involve efficiently identifying corresponding points in the two images on a sub-pixel level, especially for texture-less image regions and obliquely viewed surfaces.

- Multi-image bundle adjustment: While the assignment presented here only involves two-view BA, the approach could be extended two handle arbitrarily many views. One possible assignment related to this is to build an SfM pipeline using the code you've developed over the course of the semester. Given multiple input images of the same scene, your code would extract features for those images, match and geometrically verify the features between the image pairs, extract the relative pose for each pair, place the images and points into the 3D reconstruction space, and bundle adjust the entire result. Your main challenge will involve deciding how to best introduce new images and points into the reconstruction space, starting from some initial pair of images.

- Other possible topics: Of course, you are not limited to the topics presented here – any reasonable project related to the course material (including 2D features and convolutional neural networks) is fair game.