TTK4255: Robotic Vision

Homework 5: Two-view geometry

© Simen Haugo

This document is for the Spring 2021 class of TTK4255 only,
and may not be redistributed without permission.

Figure 1: Visualization of the epipolar constraint. For any point in one image, the corresponding point in the other image lies on the associated *epipolar line*. Here the right camera is in front of the left camera, which causes the epipolar lines to intersect at the *epipole*—the projection of the right camera's origin.

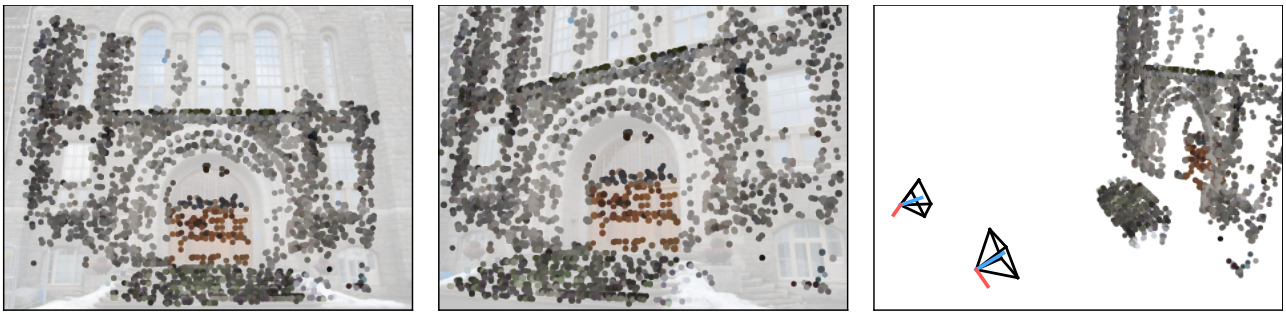

Figure 2: 3D scene reconstructed with the methods in this assignment, shown from the two original viewpoints and a novel viewpoint.

Instructions

For general information about the assignments, including grading critera and how to get help, consult the assignments.pdf document on BB. To get your assignment approved, you only need to complete 60% (weighting is next to each task). Upload the requested answers and figures as a single PDF. You don't need to submit your code. You may use any convenient tool to create your report.

About the assignment

Due to the loss of depth that occurs in projection, an image of a point only tells us that its 3D position is along a ray (the *viewing ray*). Given a second image of the point from a different view, we can easily triangulate its 3D position, if we know the transformation between the views (the *relative pose*).

What may be surprising is that we can recover (up to a scaling factor) both the 3D structure and the relative pose from a single image pair. The principle that enables this is the epipolar constraint, from which many reconstruction algorithms have been derived, such as Longuet-Higgins' 8-point algorithm and Nister's 5-point algorithm.

These algorithms require point correspondences between the images. The problem of finding these is not covered by this assignment, but is deferred to the final project. As you will come to learn, this is a step with a high likelihood of making mistakes. Most reconstruction algorithms are therefore used with a robust estimation strategy, such as RANSAC.

Relevant reading

The relevant chapters in Szeliski are 7.1 (Triangulation) and 7.2 (Two-frame structure from motion). Forty years since its invention, there have been several critiques of the original RANSAC algorithm; see e.g. the recent talk by Barath (link). Nevertheless, implementing the original algorithm is helpful for understanding the modern variants.

The assignments and projects in this course are copyrighted by Simen Haugo (haugo.simen@gmail.com) and may not be copied, redistributed, reused or repurposed without written permission.

Part 1 The epipolar constraint (20%)

The 3D information that can be extracted from a pair of images is encoded in the epipolar constraint,

$$\tilde{\boldsymbol{x}}_2^T \mathbf{E} \tilde{\boldsymbol{x}}_1 = 0. \tag{1}$$

Here, $\boldsymbol{x}_i$ are calibrated image coordinates in image i and $\mathbf{E}$ is called the essential matrix. To understand the origin and the interpretation of the epipolar constraint, you first need to understand homogeneous line representations. Recall that the line equation in normal form is $x\cos\theta + y\sin\theta = \rho$. This can equivalently be written as

$$\tilde{\boldsymbol{x}}^T \tilde{\boldsymbol{l}} = 0, \tag{2}$$

where $\tilde{\boldsymbol{x}} = (x, y, 1)$ and $\tilde{\boldsymbol{l}} = (\cos\theta, \sin\theta, -\rho)$.

Note: Since $\tilde{\boldsymbol{x}} = \mathbf{K}^{-1}\tilde{\mathbf{u}}$, we can alternatively express the epipolar constraint as $\tilde{\mathbf{u}}_2^T \mathbf{F} \tilde{\mathbf{u}}_1 = 0$, where $\mathbf{F} = \mathbf{K}^{-T}\mathbf{E}\mathbf{K}^{-1}$ is called the fundamental matrix. In robotics we normally have an accurate estimate of $\mathbf{K}$, so the fundamental matrix, if we need it, is simply computed from the essential matrix.

Task 1.1: (4%) Explain why any scalar multiple of $\tilde{\boldsymbol{l}}$ represents the same line, and explain how to normalize an arbitrary homogeneous line $\tilde{\boldsymbol{l}} = (a, b, c)$ into the normal form $(\cos\theta, \sin\theta, -\rho)$.

Task 1.2: (4%) Besides letting us write the line equation concisely with the dot product, the use of homogeneous coordinates also lets us to construct a line connecting two points with the cross product. Show that for any pair of 2D points $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$, the line passing through the points is $\tilde{\boldsymbol{l}} = \tilde{\boldsymbol{x}}_a \times \tilde{\boldsymbol{x}}_b$.

Task 1.3: (4%) Given a point $\boldsymbol{x}_1$ in image 1, we can infer that the original 3D point is somewhere along the associated viewing ray. This implies that there exists a scalar $\lambda \geq 0$ such that $\mathbf{X}^1 = \lambda\tilde{\boldsymbol{x}}_1$. The perspective projection of any point along this ray in image 2 is (in homogeneous coordinates)

$$\tilde{\boldsymbol{x}}_2 = \mathbf{R}\mathbf{X}^1 + \mathbf{t} = \lambda\mathbf{R}\tilde{\boldsymbol{x}}_1 + \mathbf{t}, \tag{3}$$

where $[\mathbf{R}\ \ \mathbf{t}]$ is the transformation from camera 1 to camera 2. Find the limit value of $\boldsymbol{x}_2$ when $\lambda \to 0$ and when $\lambda \to \infty$. Give an interpretation of what both limit values represent, possibly using the figure on the front page.

Task 1.4: (4%) Because a line in the scene remains a line in the image, the values for $\boldsymbol{x}_2$ obtained by sweeping λ from 0 to ∞ forms a line in image 2. Find an expression for this line's homogeneous representation and relate your answer to the epipolar constraint and the essential matrix $\mathbf{E}$.

Task 1.5: (4%) If the scene is static, the epipolar constraint is satisfied by all pairs of corresponding points $\boldsymbol{x}_1 \leftrightarrow \boldsymbol{x}_2$ with the same matrix $\mathbf{E}$. What can happen if the scene is not static?

Part 2 The 8-point algorithm (20%)

An algorithm to estimate $\mathbf{E}$ arises naturally by observing that the epipolar constraint is *linear* in the entries of $\mathbf{E}$, with the coefficients being products of the 2D coordinates. Therefore, we may let

$$\mathbf{E} = \begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{bmatrix} \tag{4}$$

and form a vector containing the entries of $\mathbf{E}$,

$$\mathbf{e} = \begin{bmatrix} E_{11} & E_{12} & E_{13} & E_{21} & E_{22} & E_{23} & E_{31} & E_{32} & E_{33} \end{bmatrix}^T. \tag{5}$$

Then it is a straightforward arithmetic exercise to show that the epipolar constraint for a single point correspondence can be written as

$$\tilde{\boldsymbol{x}}_2^T \mathbf{E} \tilde{\boldsymbol{x}}_1 = 0 \quad \Leftrightarrow \quad \begin{bmatrix} x_2x_1 & x_2y_1 & x_2 & y_2x_1 & y_2y_1 & y_2 & x_1 & y_1 & 1 \end{bmatrix} \mathbf{e} = 0, \tag{6}$$

assuming that the coordinates have been dehomogenized. Each correspondence gives a single linear equation. By stacking the equations from n correspondences, we obtain a homogeneous linear system $\mathbf{A}\mathbf{e} = \mathbf{0}$, with $\mathbf{A} \in \mathbb{R}^{n \times 9}$. As in Homework 4, the solution is unique only up to a scaling factor, which reflects our inability to recover the original scale of the scene and motion.

$\mathbf{A}$ must have a one-dimensional null-space for there to be a unique (up to scale) non-trivial solution $\mathbf{e}$. This can be achieved with at least eight correspondences. As usual, if the system is over-determined ($n > 8$), there may not be an exact solution, and a least-squares solution can instead be obtained from the singular value decomposition of $\mathbf{A}$.

The 8-point algorithm takes its name from the minimum number of correspondences required for $\mathbf{A}\mathbf{e} = \mathbf{0}$ to have a well-determined solution, but it can be used with more than eight correspondences to increase the accuracy, as you will do here. Note also that $\mathbf{E}$ can be estimated with fewer than eight correspondences (at least five), but that requires a more complicated algorithm.

Task 2.1: (20%) Implement the 8-point algorithm in the provided stub function `estimate_E`.

The provided `main` script loads the image pair shown previously and a pre-determined set of good point correspondences between them (in uncalibrated pixel coordinates). The images have been corrected for lens distortion and the camera intrinsic matrix is also provided. Modify the script to estimate the essential matrix for the image pair.

Convert the estimated essential matrix into a fundamental matrix and uncomment the line calling the `draw_correspondences` function. This should generate a figure of the resulting epipolar lines in both images, for a random subset of the points. Include this figure in your report and comment on your results; are they what you expect?

Part 3 Triangulation and 3D reconstruction (40%)

Szeliski 7.2 explains how $\mathbf{E}$ can be decomposed into a rotation and translation between the two views (the relative pose). Any point's 3D position can then be triangulated from its pair of corresponding image coordinates. In the absence of noise, this is simply the intersection of the two viewing rays. In practice, these rays will not intersect and we instead seek a 3D position that is, in some sense, optimal.

A simple solution (though not optimal in any useful sense!) can be obtained from the direct linear transform. Consider a point $\mathbf{X}$ in an arbitrary "world" frame. We observe its projection in the two images, which we assume satisfy a perspective projection, such that

$$\tilde{\boldsymbol{x}}_1 = \mathbf{P}^1\tilde{\mathbf{X}} \quad \text{and} \quad \tilde{\boldsymbol{x}}_2 = \mathbf{P}^2\tilde{\mathbf{X}}, \tag{7}$$

where $\mathbf{P}^i = [\mathbf{R}^i \;\; \mathbf{t}^i]$ are the 3×4 projection matrices for the two images (omitting $\mathbf{K}$ because we have calibrated cameras). The image coordinates that we measure are the dehomogenized coordinates,

$$x_i = \frac{\boldsymbol{p}_1^i\tilde{\mathbf{X}}}{\boldsymbol{p}_3^i\tilde{\mathbf{X}}} \quad \text{and} \quad y_i = \frac{\boldsymbol{p}_2^i\tilde{\mathbf{X}}}{\boldsymbol{p}_3^i\tilde{\mathbf{X}}}, \tag{8}$$

where $\boldsymbol{p}_j^i$ is the j'th row of $\mathbf{P}^i$. Including both images, this gives a total of four non-linear equations. Multiplying each equation by its denominator, we can write these as a linear system,

$$\mathbf{A}\tilde{\mathbf{X}} = \begin{bmatrix} x_1\boldsymbol{p}_3^1 - \boldsymbol{p}_1^1 \\ y_1\boldsymbol{p}_3^1 - \boldsymbol{p}_2^1 \\ x_2\boldsymbol{p}_3^2 - \boldsymbol{p}_1^2 \\ y_2\boldsymbol{p}_3^2 - \boldsymbol{p}_2^2 \end{bmatrix} \begin{bmatrix} \tilde{X} \\ \tilde{Y} \\ \tilde{Z} \\ \tilde{W} \end{bmatrix} = 0, \tag{9}$$

for which there is an exact, unique (up to scale), non-trivial solution $\tilde{\mathbf{X}}$, obtainable by the SVD of $\mathbf{A}$. The real 3D position can be found by dividing by $\tilde{W}$ if it's non-zero.

Task 3.1: (10%) Points far away from the camera are best represented in homogeneous coordinates with $\tilde{W} \approx 0$. This avoids issues that would otherwise occur when storing large numbers. However, is the $\tilde{W}$ component of the SVD solution necessarily close to zero if the point happens to be far away?

Task 3.2: (30%) Implement `triangulate_many`. The function should take all the correspondences, the two projection matrices, and return the dehomogenized 3D points in world coordinates. You can test your implementation by running the `test_triangulate` script.

An implementation of the decomposition in Szeliski is in the hand-out code (`decompose_E`). Use it to extract the relative pose from $\mathbf{E}$. Then, triangulate the 3D position of the correspondences. Uncomment the code that plots the 3D point cloud and include a figure from some viewpoint.

As explained in the book, there are always four valid decompositions of a given essential matrix. The correct one can be identified by requiring that the triangulated points are in front of **both** cameras.

You can arbitrarily attach the "world" frame to the camera frame of image 1. The projection matrices are then $\mathbf{P}^1 = [\mathbf{I}_{3\times3} \;\; \mathbf{0}]$ and $\mathbf{P}^2 = [\mathbf{R} \;\; \mathbf{t}]$, where $[\mathbf{R} \;\; \mathbf{t}]$ is the relative pose extracted from $\mathbf{E}$.

Part 4 Robust estimation with RANSAC (20%)

Outliers are misinformative measurements that should be excluded from the estimation process. Point correspondences often contain outliers caused by failure of the matching algorithm or independently moving objects. Ideally, an essential matrix is estimated using only its *inliers*—the correspondences that are consistent with the motion that it represents, within some error tolerance. The classical strategy for estimating an essential matrix and simultaneously identifying its inliers is RANSAC:

1. In each iteration of a loop, randomly select a sample of m correspondences.
   1.1. Estimate the essential matrix using only the m correspondences.
   1.2. Compute a vector of residuals for the full set of correspondences.
   1.3. Count the number of residuals within a specified threshold (the inlier count).
2. Return the essential matrix and the associated inlier set that had the highest inlier count.

The essential matrix is typically re-estimated using the full inlier set at the end. If the 8-point algorithm is used, $m = 8$ is a natural choice for the sample size in the loop.

For this part you should use the outlier-containing correspondences in `task4matches.txt`.

Task 4.1: (5%) The residuals should quantify how well the correspondences fit an essential matrix, with the hope that inliers are distinguished from outliers by having lower residuals. One definition is based on the distance between each point and its epipolar line, in the two images,

$$e_2 = \frac{\mathbf{u}_2^T \mathbf{F} \mathbf{u}_1}{\sqrt{(\mathbf{F}\mathbf{u}_1)_1^2 + (\mathbf{F}\mathbf{u}_1)_2^2}} \quad \text{and} \quad e_1 = \frac{\mathbf{u}_1^T \mathbf{F}^T \mathbf{u}_2}{\sqrt{(\mathbf{F}^T\mathbf{u}_2)_1^2 + (\mathbf{F}^T\mathbf{u}_2)_2^2}}. \tag{10}$$

The reason for the denominator should be clear from Task 1.1 and the use of the fundamental matrix is to get a distance in pixel units. These can be averaged to get one scalar residual per correspondence. (Note that the numerators are actually identical, so they don't cancel by having different signs).

Implement `epipolar_distances` based on this definition. Include a histogram of the residuals for the outlier-containing correspondences in `task4matches.txt`, with the essential matrix from Part 2.

Task 4.2: (5%) Implement RANSAC in `estimate_E_ransac`. The residuals are signed, so the absolute value should be taken before comparing against the threshold. Run for $20{,}000$ iterations with an inlier threshold of 4 pixels. Modify your script to triangulate only the inliers of `task4matches.txt`. Include the generated figures and the size of the inlier set.

Task 4.3: (5%) You may get a few floating points that clearly should not be part of the reconstruction. Are these outliers or inliers? Explain why these arise and how you could get rid of them.

Task 4.4: (5%) Szeliski 6.1.4 has a formula (Eq. (6.30)) for the required number of iterations, given a desired probability of success (the *confidence*) and the inlier fraction. The inlier fraction here is 0.50. Try running RANSAC for the number of iterations produced by the formula with a confidence of 0.99. Try this a few times. Do you expect it to find the largest inlier set 99% of the time? Why/why not?

The document continues on the next page →

Task 4.5: (Optional self-study task - 0%) Although the two-frame reconstruction algorithm here lacks some features you would want in a solid implementation, it can still produce a reasonable-looking 3D model and camera motion estimate. As preparation for the final project, or possibly just for fun, you may want to try your algorithm on an image pair acquired with your own camera.

To find correspondences between the image pair, we recommend OpenCV (link) for Python and the Computer Vision Toolbox (link) for Matlab. Note that the patent for SIFT expired last year, so you no longer have to compile OpenCV from scratch to use it in Python.

You may first try to find your own correspondences on the provided image pair and compare your results with those using the provided correspondences. To use your own images, you will need to calibrate your camera to get the intrinsic matrix and to be able to undistort the images. Again, OpenCV and the Computer Vision Toolbox have functionality to do this.