Lab04 - Image Relative Orientation (Two Views)

ENV-408 - Jan Skaloud

13/03/2025

Objectives

Understand how to orient a pair of stereo images relative to one another by finding the pose (position t and attitude R) of the right image with respect to the left image. This is the same as considering the pose of the left image as origin ($t_1 = 0$, $R_1 = I$) and finding $R = R_2$ and $t = t_2$. Hence, triangulated points will be expressed in camera 1 frame. Inputs are the (automated) image observations of corresponding points obtained from the provided image pair for this lab.

Overview

Input data

- `IMG A: raw_data/img_1092311568.jpg` and `IMG B: raw_data/img_1092314704.jpg`: down scaled image pair for which feature correspondences should be computed and saved in a csv. `IMG A` was used as the reference image in `Lab03`. Be sure to load images in the correct order in your *Exercise 2* SIFT function: units: pixels, origin: top-left.
- `raw_data/cam_param.txt`: info on image size (`height width`) in pixels, the camera constant (`c`) in pixels, coordinates of perspective point (`cx cy`): units pixels, origin top-left and the distortion parameters (`k1 k2 k3`).

Functions to implement

- `p = undistort_solver(cam_param, uv)` and `undistort(var, x_d, y_d, k1, k2, p1, p2)`. These functions include your implementation of *Exercise 1* that transform raw perspective centered coordinates (x', y') to undistorted coordinates (x, y).
- `xy = topleft2perspective(uv, cam_p)` Your implementation of *Exercise 1* that expresses top left coordinates in perspective centered normalized coordinates (was used also in *Exercise 3*).
- `P = triangulation(p1, p2, Rt_1, Rt_2)`: Intersects correspondences p_1, p_2 using the relative orientation [R|t] as obtained via the E matrix determined with the `eightpoint(p1, p2)` function below.
- `E = eightpoint(p1, p2)`: Estimates the essential matrix E using the 8-points algorithm.
- `[R1, R2, t1, t2] = decomposeEssential(E)`: Decomposes the essential matrix E into four numerically possible solutions of relative orientation.
- `[R,t] = checkRelativePose(Rs, ts, p1, p2)`: Selects the valid solution out of four previously obtained in `decomposeEssential(E)` so the triangulated points lie in front of both images (have positive depth).

- Notation: $[x]_\times$ denotes a skew-symmetric matrix of vector $x = (x_1, x_2, x_3)^T$ as $[x]_\times = \begin{pmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{pmatrix}$

Task 1: Triangulation

The final goal of this exercise is to determine the spatial structure of points P_i through *stereo vision*, through their observations in left and right cameras. This situation is depicted in Figure 1: once the relative orientation (relative pose) between both cameras is known, the spatial intersection (also called triangulation) of rays defines their position in space.

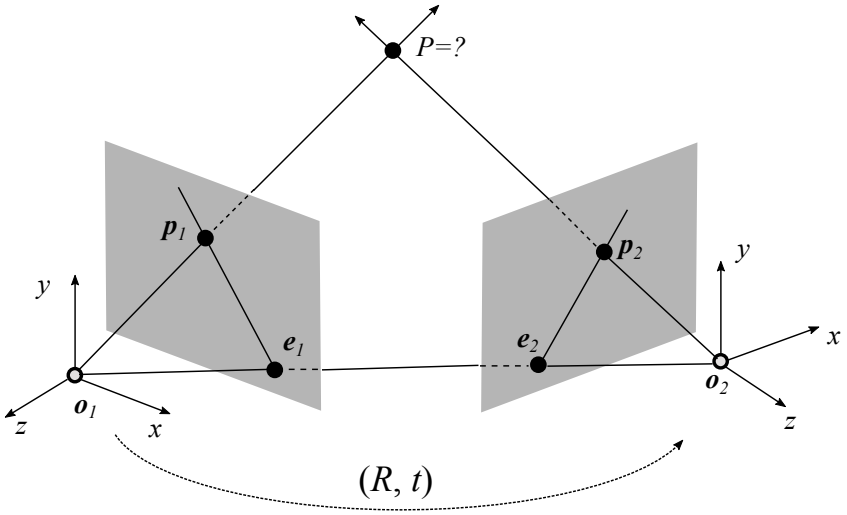

Figure 1: Goal: estimate P from p_1, p_2 and projection matrices, via triangulation (spatial intersection).

Although $[R|t]$ will be determined later, let's assume for the moment that they are known and defined as we implement the triangulation. As will be shown later, the triangulation function is useful in the process of recovering R and t.

Formulation

Lets consider the acquisition of stereo-pair through motion in which the left and right images are captured by the same camera but at different instances of time, in other words, $K_1 = K_2 = K$. In our implementation, we have already expressed measurements in normalized coordinates, which implies that $K = I_3$. Further, the relative mapping-frame is defined by the origin and axes of the *left* camera. With that, the projection of p_1 to object coordinates (X, Y, Z) of P for the *left* camera takes the following form:

$$\mu_1 \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = \underbrace{[I \,|\, 0]}_{\Pi_1'} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \implies \mu_1 p_1 = \Pi_1' \cdot P \implies p_1 \times \mu_1 p_1 = p_1 \times \Pi_1' \cdot P \implies 0 = [p_1]_\times \cdot \Pi_1' \cdot P$$

Similarly, we obtain for the *right* camera:

$$\mu_2 \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \underbrace{[R\,|\,t]}_{\Pi_2'} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \implies \mu_2 p_2 = \Pi_2' \cdot P \implies p_2 \times \mu_2 p_2 = p_2 \times \Pi_2' \cdot P \implies 0 = [p_2]_\times \cdot \Pi_2' \cdot P$$

Putting the last equations for both cameras together we obtain a homogeneous system of equations.

$$\begin{pmatrix} [p_1]_\times \cdot \Pi_1' \\ [p_2]_\times \cdot \Pi_2' \end{pmatrix} \cdot P = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \cdot P = A \cdot P = 0$$

.

We choose a non-zero P that minimizes the quadratic norm $||A \cdot P||_2$. This can be solved by the singular value decomposition (SVD) of A: $A = U\Sigma V^T$. Then the solution to $A \cdot P = 0$ is the last column of V that corresponds to the smallest eigenvalue in Σ.

Implementation

1. For each point p_1^i, p_2^i:

   1.1 Estimate $A_1 = [p_1]_\times \cdot \Pi_1'$, $A_2 = [p_2]_\times \cdot \Pi_2'$.

   Hint: `skewMatrix(pi)` will help you build $[p_i]_\times$.

   1.2 Perform SVD

   Hint: Numpy SVD outputs V^T, not V, `Matrix[:,-1]` accesses the last column of a matrix.

   1.3 Extract P from V.

2. Scale P back to homogeneous coordinates by dividing each P by its 4th coordinate that maybe no longer unity).

Control

Section 'Test task 1' is provided, which will call your `triangulation()` function on some test $[R|t]$ matrices test your implementation is correct.

Task 2: Essential matrix

The 8-point algorithm is presented in the lecture and in the supplementary reading (book chapter). It aims to find the *essential* (if K is known, otherwise *fundamental*) projection matrix that satisfies the coplanarity constraint between the normalized image correspondences p_1^i, p_2^i and the relative translation vector t as shown in Figure 1.

Formulation

The system to solve is the following, starting from coplanarity constraint and using a similar trick as in Lab03:

$$p_2^T\,[t]_\times\,R\,p_1 = p_2^T\,E\,p_1 = 0 \iff Q \cdot vec(E) = 0$$

With $vec(E) = (e_{12}, e_{21}, e_{31}, e_{21}, \ldots, e_{33})$ and

$$Q = \begin{pmatrix} (p_2^1 \otimes p_1^1)^T \\ \vdots \\ (p_2^n \otimes p_1^n)^T \end{pmatrix}$$

- The problem $Q \cdot vec(E) = 0$ is again in the form of a set homogeneous linear equations as in DLT or in triangulation. The solution follows from SVD: $Q = U\Sigma V^T$ where the vector $vec(E)$ is the last column of V corresponding to the smallest eigenvalue.
- The vector $vec(E)$ must be reshaped back to E.
- E is supposed to be of rank 2, which guarantees that all epipolar lines in an image intersect at a single point, the epipole. Numerically this corresponds to $det(E) = 0$. However, due to the presence of noise in the data, this condition must be enforced. One possibility to do so is the following:
  a. Compute SVD of $E = U\Sigma V^T$, where $\Sigma = diag(\sigma_1, \sigma_2, \sigma_3)$.
  b. Set the smallest eigenvalue to zero ($\sigma_3 = 0$) and the other two eigenvalues to their mean: $\sigma = 0.5 \cdot (\sigma_1 + \sigma_2)$.
  c. Recalculate $E = U\,\texttt{diag}(\sigma, \sigma, 0)V^T$.

Implement the function `E = eightpoint(cpts1, cpts2)` following steps below.

Steps

1. Compute Q matrix (using the Kronecker product $\otimes$ of p_1, p_2, see lecture codes hint[1])
2. Compute SVD of Q.
3. Extract $vec(E)$ and reshape it back to E matrix.
4. Folowing the steps a. b. c. above, compute SVD of E, set the first two eigen-values to their mean and the third to zero to enforce rank 2. This assures that the obtained E belongs to the space of essential matrices.
5. Return the re-estimated E from the modified $U\Sigma V^T$.

   Hint: The function `np.diag()` is useful in forming Σ.

Control

The second unitary test will estimate the correctness of your implementation by estimating the reprojection error (RMSE) of all tie points $\left(\frac{1}{N}\sum_{i=1}^{N}(p_2^i E p_1^i)^2\right)^{-1/2}$.

Task 3: Decomposition of E

3.1 Possible solutions

$(R|t)$ can be extracted from the decomposition of the essential matrix E. In the course slides, the method for extracting t is presented (slide "Extracting R,t from E, part II. finding t"). Here we propose a simplified method to extract R and t by exploiting the SVD of the previously determined essential matrix and testing the different combination of $R|t$ afterward (leading to the same solution as the one you would obtain from the method of the course).

1. Perform SVD of the $E = U\Sigma V^T$ returned by `eightpoint(cpts1, cpts2)`
2. The two possible translations are $t_1 = u_3$ and $t_2 = -u_3$ where u_3 is the **last column** of U
3. The two possible rotation matrices are $R_1 = UWV^T$ and $R_2 = UW^TV^T$, with:

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

   and U, V being the previously defined elements of the SVD.

[1] Refer to the lecture 4 on Stereo Vision, slide "The 8-point algorithm – SVD of Q in Python"

4. Finally, the 4 possible solutions to test for R and t are :

$$\begin{aligned}(R_1|t_1) &= [UWV^T|u_3] \\ (R_2|t_1) &= [UW^TV^T|u_3] \\ (R_1|t_2) &= [UWV^T|-u_3] \\ (R_2|t_2) &= [UW^TV^T|-u_3]\end{aligned}$$

Steps

Goal: Implement `[R1, R2, t1, t2] = decomposeEssential(E)`.

1. Perform SVD of E as estimated in the previous task.
2. Extract translation vector from `U` (third column) normalized it (divide all components by vector norm) and express the `t1` and `t2=-t1`
3. Express `W` matrix
4. Estimate `R1`, `R2` from `U`, `W` and `V` as defined in Step 4 above. *Note*: Make sure to normalize `t1`, `t2` and ensure that the determinant of `R1`, `R2` is positive (if not, multiply the rotation matrix with negative determinant by `-1`)

3.2 Finding the right solution

The correct solution is that which represents reality. For that, both cameras must look (within $pi/2$) toward the same direction and all triangulated points should lie in front of them, or in other words have a positive depth.

Implementation

Complete `[R,t] = checkRelativePose(Rs, ts, p1, p2)`. Here `triangulation()` function is useful. To select the correct rotation and translation, you can triangulate the points for each $R|t$ combination and select the solution that leads to all triangulated points P having positive depth, i.e. lying in front of both cameras.

Test task 3: Model visualization with control

We can now verify that the estimated $(R|t)$ transformation can correctly triangulate new points visible by both cameras. A code skeleton is provided that evaluates the model and can be executed as is, provided that you define the variables with the expected names and respect the format. The unitary test is implemented which verifies that your model is correct. *If your code pass this last test, the implementation of the 8-point algorithm is correct, congratulations!*

Visualization. Plot the reconstructed scene either in 2D or 3D: both cameras (with their orientation) and the coordinates of points using `plot_pipeline(P,  t, R, units)` that is contained in `utils.py`

Questions

1. **RMSE.** Are you happy with the obtained of reprojection error (test 2)? If yes why? If not, how do you propose to improve it?
2. **Closest point 1.** What is the closest point that you detected with respect to the *left=first* camera for the unitary baseline? What are the units?
3. **Absolute pose (position and attitude) of the right camera.** Considering a real distance between left and right camera to be 211.35 m. What is the absolute pose (position & attitude) of the *right=second* image? *Hint: you may want to use/integrate the results of some previous lab.*
4. **Coordinates of the closest point 1.** Express the absolute coordinates of the closet point 1 (considering the absolute orientation), specify its units.

5. **Absolute orientation of model - propose an alternative.** Put yourself in the situation where in the point 3. the distance between the left and right cameras are not given. Suggest briefly in steps:
   a. How do you propose to resolve it? *If needed, use formulas rather than description.*
   b. How do you propose to verify that your solution is correct?

Deliverables & report submission

Submit your lab to Moodle before **28-03-2025 23:59 hrs** as

- One zip file with the following content:
  - your code with auxiliary functions (so it can be executed),
  - the input files you used (*without* images!) and,
  - the report (max 2 pages) with answers to questions 1-5.

  Feel free to support your answers with plots or tables.