The Perspective-n-Point (PnP) problem

Contents

1 Preliminaries **1**
- 1.1 Outline of the exercise . . . 1
- 1.2 Description of the input data . . . 1
- 1.3 Notations and coordinate systems . . . 2

2 Implementing DLT **2**
- 2.1 Reminder: The DLT algorithm. . . . 2
  - 2.1.1 Derivation of the linear system of equations to solve . . . 2
  - 2.1.2 Solving the over-determined system of equations . . . 3
  - 2.1.3 Extracting $\mathbf{R}_{CW}$, ${}_C\mathbf{t}_{CW}$ from $\tilde{M}$ with the correct scale . . . 4
- 2.2 Implementation of DLT . . . 4

3 Numerical Exercise **6**

The goal of this laboratory session is to implement the Direct Linear Transform (DLT) algorithm to estimate the pose of a camera, given a set of 2D-3D correspondences.

1 Preliminaries

1.1 Outline of the exercise

You will be given a dataset of images, with a set of 2D-3D correspondences for each image, as well as the camera matrix K. Your goal will be to implement the DLT algorithm described during the course to estimate the world-to-camera extrinsics $[\mathbf{R}_{CW}|{}_C\mathbf{t}_{CW}]$ (the 3×4 block of $\mathbf{T}_{CW}$) for each image.

1.2 Description of the input data

The ***data/*** folder contains the inputs that you will need to complete these exercises.

- ***images_undistorted/*** contains images that have been processed to compensate for lens distortion (using the function `undistortImage` from the previous exercise). This allows you to completely ignore the effect of lens distortion when solving the PnP problem.
- ***K.txt*** contains the camera matrix
- ***p_W_corners.txt*** contains the positions of the n reference 3D points (illustrated in Figure 1), given in the world coordinate system (defined below), in centimeters. $n = 12$ in this exercise.
- ***detected_corners.txt*** contains m lines (where m is the number of images). Each line i gives the pixel coordinates $(u_1, v_1, \ldots, u_n, v_n)$ of the projections of the n reference 3D points in undistorted image i.

1.3 Notations and coordinate systems

In this exercise, we use the following conventions. The left subscript is the frame in which a quantity is expressed. We write $\mathbf{R}$ for rotations (not $\mathbf{C}$).

- ${}_A\mathbf{P}$ denotes that the Euclidean point $\mathbf{P}$ is expressed in frame A. The homogeneous counterpart is ${}_A\tilde{\mathbf{P}}$.
- $\mathbf{R}_{BA}$ maps vectors from A to B (orientation of A with respect to B):

$$ {}_B\mathbf{v} = \mathbf{R}_{BA}\, {}_A\mathbf{v}. $$

- $\mathbf{T}_{BA}$ maps points from A to B (pose of A with respect to B):

$$ {}_B\tilde{\mathbf{P}} = \mathbf{T}_{BA}\, {}_A\tilde{\mathbf{P}}, \qquad \mathbf{T}_{BA} = \begin{bmatrix} \mathbf{R}_{BA} & {}_B\mathbf{t}_{BA} \\ \mathbf{0} & 1 \end{bmatrix}, $$

  where ${}_B\mathbf{t}_{BA}$ is the origin of A expressed in B.

The reference (or world) frame, denoted W, is right-handed and illustrated in Figure 1. The camera frame is denoted C. 3D points in ***p_ W_ corners.txt*** are given in W (centimetres).

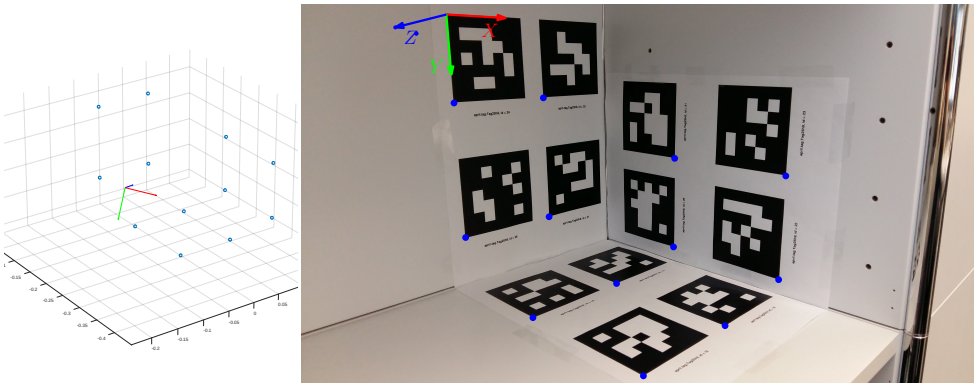


Figure 1: Left: 3D world points ${}_W\mathbf{P}^{(i)}$ and the camera pose in space. Right: Image from a camera with the corresponding pixel projections (u_i, v_i) marked in blue.

2 Implementing DLT

In this section, you will implement the DLT algorithm to determine the camera pose for each image in the dataset, using the 2D-3D correspondences provided for each image.

The DLT algorithm is described in the lecture slides. We first briefly remind here how it works (and define the notations that we use).

2.1 Reminder: The DLT algorithm.

2.1.1 Derivation of the linear system of equations to solve

As opposed to the lecture slides where the camera calibration matrix K was not given (and therefore had to be estimated jointly with the camera pose), in this exercise K is given. We briefly derive here a slightly modified version of the DLT algorithm presented in the course that takes into account the fact that K is known.

Our goal is to compute $\mathbf{R}_{CW}$ and ${}_C\mathbf{t}_{CW}$ that satisfy the perspective projection equation:

$$ \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[\mathbf{R}_{CW} | {}_C\mathbf{t}_{CW}]\, {}_W\tilde{\mathbf{P}} $$

Multiplying each side by K^{-1} on the left, we get:

$$\lambda K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = [\mathbf{R}_{CW}|_C\mathbf{t}_{CW}]\, {}_W\tilde{\mathbf{P}}$$

where we identify $[x, y, 1]^\top = K^{-1}[u, v, 1]^\top$ as the *normalized (calibrated) image coordinates* corresponding to pixel (u, v). Denoting $\tilde{M} = [\mathbf{R}_{CW}|_C\mathbf{t}_{CW}]$ the *projection matrix* in calibrated coordinates, the problem amounts to finding $\tilde{M}$ and the scale factors λ_i that satisfy:

$$\lambda_i\, {}_C\tilde{\mathbf{p}}^{(i)} = \tilde{M}\, {}_W\tilde{\mathbf{P}}^{(i)}$$

for every 2D-3D correspondence $i = 1, \ldots, n$, where ${}_C\tilde{\mathbf{p}}^{(i)} = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$ and ${}_W\tilde{\mathbf{P}}^{(i)} = \begin{bmatrix} X_W^{(i)} \\ Y_W^{(i)} \\ Z_W^{(i)} \\ 1 \end{bmatrix}$ are the i^{th} calibrated image point and world point.

However, as it was shown in the lecture, by considering ratios of pairs of equations (such as $\frac{\tilde{u}_i}{\tilde{w}_i} = \frac{\tilde{m}_1^T \cdot P_i}{\tilde{m}_2 \cdot P_i}$), the scale factors λ_i are cancelled, i.e. removed from the equations and the problem reduces to finding $\tilde{M}$ alone. The scale factors can be recovered, if desired, once $\tilde{M}$ is known. It is shown in the lecture that $\tilde{M}$ can be recovered by solving the following homogeneous system of linear equations for the unknown $[12 \times 1]$ matrix $\tilde{M}$:

$$Q \cdot \tilde{M} = 0 \tag{1}$$

where

$$Q = \begin{bmatrix} X_W^{(1)} & Y_W^{(1)} & Z_W^{(1)} & 1 & 0 & 0 & 0 & 0 & -x_1X_W^{(1)} & -x_1Y_W^{(1)} & -x_1Z_W^{(1)} & -x_1 \\ 0 & 0 & 0 & 0 & X_W^{(1)} & Y_W^{(1)} & Z_W^{(1)} & 1 & -y_1X_W^{(1)} & -y_1Y_W^{(1)} & -y_1Z_W^{(1)} & -y_1 \\ & & & & & & \cdots & \cdots & \cdots & & & \\ X_W^{(n)} & Y_W^{(n)} & Z_W^{(n)} & 1 & 0 & 0 & 0 & 0 & -x_nX_W^{(n)} & -x_nY_W^{(n)} & -x_nZ_W^{(n)} & -x_n \\ 0 & 0 & 0 & 0 & X_W^{(n)} & Y_W^{(n)} & Z_W^{(n)} & 1 & -y_nX_W^{(n)} & -y_nY_W^{(n)} & -y_nZ_W^{(n)} & -y_n \end{bmatrix}$$

can be built from the known 2D-3D point correspondences and

$$\tilde{M} = \begin{bmatrix} \tilde{m}_{11} & \tilde{m}_{12} & \tilde{m}_{13} & \tilde{m}_{14} & \tilde{m}_{21} & \tilde{m}_{22} & \tilde{m}_{23} & \tilde{m}_{24} & \tilde{m}_{31} & \tilde{m}_{32} & \tilde{m}_{33} & \tilde{m}_{34} \end{bmatrix}^T$$

is an unknown projection matrix which we wish to recover. *Hint:* You may use Matlab's function `kron` to build Q (although that is not strictly necessary). Note that $\tilde{M}$ here is a $[12 \times 1]$ vector obtained by unrolling the matrix:

$$\begin{bmatrix} \tilde{m}_{11} & \tilde{m}_{12} & \tilde{m}_{13} & \tilde{m}_{14} \\ \tilde{m}_{21} & \tilde{m}_{22} & \tilde{m}_{23} & \tilde{m}_{24} \\ \tilde{m}_{31} & \tilde{m}_{32} & \tilde{m}_{33} & \tilde{m}_{34} \end{bmatrix}$$

in a *row-wise* fashion, i.e. by unrolling it row by row. Keep that in mind when you later convert $\tilde{M}$ back to a $[3 \times 4]$ matrix, since Matlab usually reshapes matrices in a *column-wise* fashion.

2.1.2 Solving the over-determined system of equations

Since Q should have rank 11, and each 2D-3D point correspondence provides 2 independent equations, at least 6 point correspondences (in general position, thus avoiding degenerate configurations, such as all 3D points lying on a plane) are needed. In this exercise, $n = 12$ point correspondences are provided, thus the system of equations is over-determined. The trivial solution $\tilde{M} = 0$ is obviously of no interest for us. We can further observe that if $\tilde{M}$ is a solution of (1), then $\alpha \cdot \tilde{M}$ is also a solution (for any scalar α; thus we will recover the projection matrix $\tilde{M}$ up to an unknown scale factor). So, we look for a solution that minimizes $||Q \cdot \tilde{M}||$ subject to the constraint $||\tilde{M}|| = 1$. This can be done

using the Singular Value Decomposition (SVD) of Q: $Q = USV^T$ where U,V are unitary matrices and S is diagonal.

It can be shown that the solution of this problem is the eigenvector corresponding to the smallest eigenvalue of Q^TQ, which simply corresponds to that the last column of V if S has its diagonal entries sorted in descending order. The `svd` function from Matlab provides such a guarantee.

2.1.3 Extracting $\mathbf{R}_{CW}$, ${}_C\mathbf{t}_{CW}$ from $\tilde{M}$ with the correct scale

Enforcing $\det \mathbf{R}_{CW} = +1$ After solving the linear system (1), you will need to convert back the $[12 \times 1]$ vector to its corresponding $[3 \times 4]$ projection matrix $\tilde{M} = [\mathbf{R}_{CW}|{}_C\mathbf{t}_{CW}]$. Make sure that the Z component of the recovered translation, ${}_Ct_z = \tilde{m}_{34}$, is positive. If that is not the case, multiply $\tilde{M}$ by -1. This ensures that the rotation matrix extracted from $\tilde{M}$ is a *proper* rotation in $SO(3)$ with determinant $+1$.

Extracting a rotation matrix from $\mathbf{R}_{CW}$ When we solved the system of linear equations (1), we did not impose any constraint on $\mathbf{R}_{CW}$ to ensure it is actually a rotation matrix (i.e. we have no guarantee that $\mathbf{R}_{CW} \in SO(3)$). This means what we actually estimated is an approximation $\bar{\mathbf{R}}_{CW}$. In this step, we extract a true rotation matrix $\tilde{\mathbf{R}}_{CW} \in SO(3)$ from $\bar{\mathbf{R}}_{CW}$ by computing the matrix closest in the Frobenius norm. This is the Orthogonal Procrustes Problem. $\tilde{\mathbf{R}}_{CW}$ can be obtained by decomposing $\bar{\mathbf{R}}_{CW}$ with SVD: $\bar{\mathbf{R}}_{CW} = U\Sigma V^\top$, then setting $\tilde{\mathbf{R}}_{CW} = UV^\top$.

Recovering the scale of the projection matrix $\tilde{M}$ As explained above, DLT recovers $\tilde{M}$ only up to scale. Writing $\tilde{M} = [\tilde{\mathbf{R}}_{CW}|{}_C\tilde{\mathbf{t}}] = [\alpha\bar{\mathbf{R}}_{CW}|\alpha\,{}_C\mathbf{t}_{CW}]$ for an unknown scale α, the orthogonal Procrustes step yields $\tilde{\mathbf{R}}_{CW} \approx \alpha\bar{\mathbf{R}}_{CW}$. We estimate α from $\alpha = \|\tilde{\mathbf{R}}_{CW}\|/\|\bar{\mathbf{R}}_{CW}\|$, where $\|\cdot\|$ is any matrix norm (e.g. `norm(A)` in MATLAB).

Wrapping up Finally, the projection matrix is $\tilde{M} = [\tilde{\mathbf{R}}_{CW}|\alpha\,{}_C\mathbf{t}_{CW}]$.

2.2 Implementation of DLT

- (`estimate_pose_dlt.py`) Write a function `M = estimatePoseDLT(p, P, K)` that implements the steps of the DLT algorithm as described above, to solve for the projection matrix $\tilde{M} = [\mathbf{R}_{CW}|{}_C\mathbf{t}_{CW}]$, given the n 2D-3D point correspondences (u_i, v_i) and ${}_W\mathbf{P}^{(i)}$.
  - **Pay particular attention to the fact that the matrix Q is built using the *normalized coordinates* ${}_C\tilde{\mathbf{p}}^{(i)}$ and not the pixel coordinates (u_i, v_i).**
  - After you have computed the $[12 \times 1]$ vector, reshape it into a $[3 \times 4]$ matrix. You can use the `reshape` function from Matlab/NumPy to achieve this. Pay attention, however, that *in Matlab* this function works *column-wise* and not *row-wise*, so you will need to first reshape to a $[4 \times 3]$ matrix and then take its transpose.
  - Apply the methods explained in 2.1.3 to ensure that the resulting $\mathbf{R}_{CW}$ is a valid rotation matrix (i.e. $\det \mathbf{R}_{CW} = 1$ and $\mathbf{R}_{CW}^\top\mathbf{R}_{CW} = I$).
- (`reproject_points.py`) Write a function `[p_reprojected] = reprojectPoints(P, M, K)` that reprojects the 3D points ${}_W\mathbf{P}^{(i)}$ in the current image using the estimated projection matrix $\tilde{M}$ and camera matrix K.
  - (`main.py`) Check that the reprojected pixels fall close to the detected (u_i, v_i). The output should look like Figure 2.
- (`main.py`) Write a function that estimates the camera pose (using your implementation of DLT) for each image in the dataset, and create an animation which shows the motion of the camera.
  - You may use the provided function `plotTrajectory3D` and the utility function `rotMatrix2Quat`.

– You can use a framerate of 30 frames per second. *Warning:* `plotTrajectory3D` expects the pose of the camera with respect to the world, i.e. $\mathbf{T}_{WC}$ (which maps points from C to W). DLT returns $\tilde{M} = [\mathbf{R}_{CW}|_C\mathbf{t}_{CW}]$, the world-to-camera extrinsics. Invert $\tilde{M}$ to obtain $\mathbf{T}_{WC} = \mathbf{T}_{CW}^{-1}$ before calling `plotTrajectory3D`.
– Finally, **the units (for the translational part and the 3D points) should be meters (whereas you worked with centimetres throughout this exercise)**. Do not forget to convert to metres before calling the function.

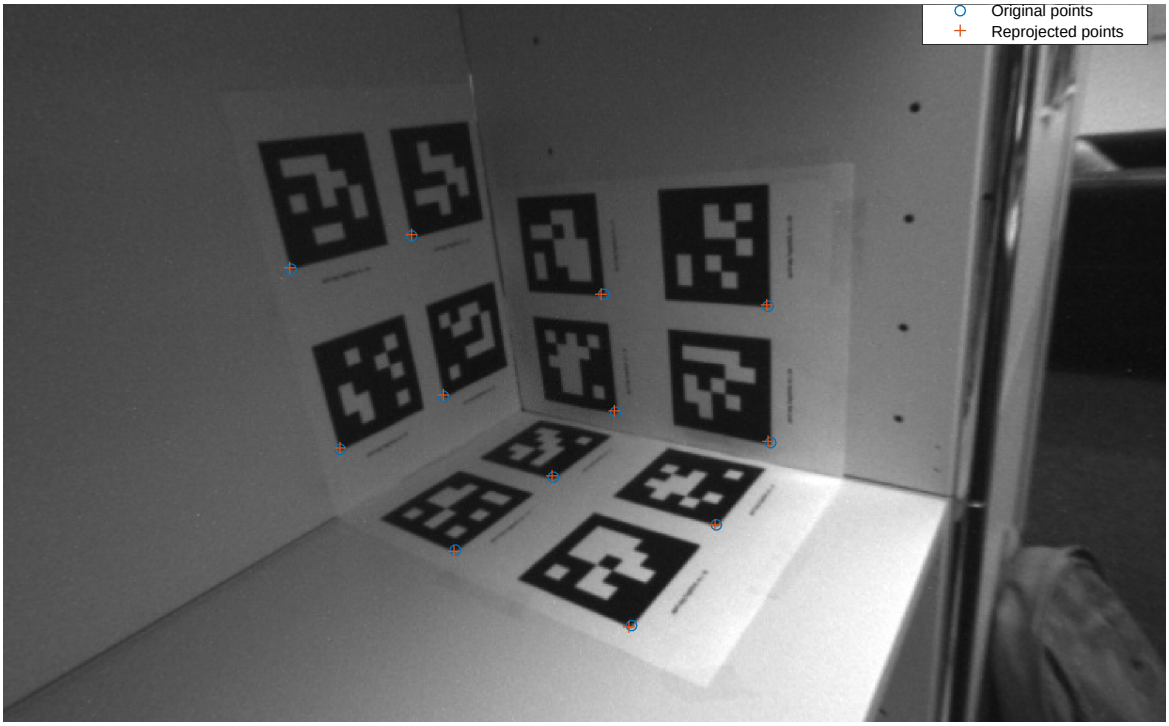


Figure 2: Original pixel detections (u_i, v_i) and reprojected points using the estimated $\mathbf{R}_{CW}$ and ${}_C\mathbf{t}_{CW}$

3 Numerical Exercise

1. Explain the Dolly-Zoom Effect.

2. What is the condition on the elements of a projective transformation H such that parallel lines remain parallel? Recall, a projective transformation relates the projected coordinates of the same 3D point between two different image planes/viewpoints.

3. Write the projective transformation H that corresponds to doubling the focal length.

4. Consider a single camera C with the following intrinsic parameters:

   - f the focal length $[m]$
   - k_x and k_y the resolution in $[\frac{pixels}{m}]$ of the camera in x and y direction
   - c_x and c_y in $[pixels]$ as the x and y coordinates of the camera central point

   and the following extrinsic parameters:

   - $\mathbf{R}_{CW}$ and ${}_C\mathbf{t}_{CW}$, forming the world-to-camera extrinsics $[\mathbf{R}_{CW}|{}_C\mathbf{t}_{CW}]$ (equivalently $\mathbf{T}_{CW}$).

   In homogeneous coordinates the projection $\tilde{\mathbf{q}}$ of a world point ${}_W\tilde{\mathbf{P}}$ onto the image plane is computed as

   $$\lambda\tilde{\mathbf{q}} = K[\mathbf{R}_{CW}|{}_C\mathbf{t}_{CW}]\,{}_W\tilde{\mathbf{P}}$$

   with the camera calibration matrix K

   $$K = \begin{pmatrix} fk_x & 0 & c_x \\ 0 & fk_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

   - Which parameters can be calibrated given a planar calibration pattern?
   - How can they be estimated?
   - Which parameters cannot be calibrated? How can this problem be tackled?

5. Discuss the use of regular squares and regular circles as calibration patterns.

   - Which points would you use as calibration points?
   - How would you detect them?
   - Which approach is more accurate?

6. You are tasked with finding the camera matrix P using the Direct Linear Transform (DLT) method. Given a set of 3D world points $X_i = (X_i, Y_i, Z_i, 1)$ and their corresponding 2D image points $x_i = (u_i, v_i, 1)$, use the following data to estimate the camera matrix P. Given world coordinates (in homogeneous coordinates):

   $$\begin{pmatrix} X_1 & Y_1 & Z_1 & 1 \\ X_2 & Y_2 & Z_2 & 1 \\ X_3 & Y_3 & Z_3 & 1 \\ X_4 & Y_4 & Z_4 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

   Given corresponding 2D image coordinates (in homogeneous coordinates):

   $$\begin{pmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ u_3 & v_3 & 1 \\ u_4 & v_4 & 1 \end{pmatrix} = \begin{pmatrix} 150 & 120 & 1 \\ 330 & 120 & 1 \\ 330 & 300 & 1 \\ 150 & 300 & 1 \end{pmatrix}$$

   Set up the linear system $A \cdot p = 0$, where A is constructed using the world coordinates and image coordinates, and p is the vector of unknown camera parameters. (Don't need to solve the system)

7. A video camera looks onto a table of variable height, illuminated by a projector which projects a pattern, comprising a single cross, onto the scene. The system is calibrated by adjusting the table in height to 50, 110, and 200 mm. For these table positions, the center of the cross is observed at the following image positions (in pixels):

$$(u, v) = \{(100, 250), (140, 330), (200, 450)\}$$

Try to determine the height of the table when the cross is observed at:

(a) (u, v) = (130, 310)

(b) (u, v) = (170, 390)

8. A drone is equipped with a calibrated camera and uses PnP (Perspective-n-Point) for localization and navigation in an urban environment. Consider the following scenario: The drone needs to navigate through a city to deliver a package. It detects 4 corners of a distinctive building, whose 3D coordinates are stored in its map database. The drone's camera captures these corners in its 2D image.

a) Explain how PnP could be used in this scenario to aid the drone's navigation. What specific information would PnP provide?

b) What are two potential challenges or sources of error when using PnP in this drone navigation scenario? How might these affect the drone's decision-making?

c) Apart from building corners, name two other types of urban features that could be useful for PnP-based localization in drone navigation. Why are they suitable choices?