TTK4255 Robotic Vision, Spring 2020

Exercise 3: Homography and Pose Estimation

Authored by Simen Haugo (simen.haugo@ntnu.no)

Due: See Blackboard

This exercise is a mix of programming problems and theory questions. You are free to use any programming language, but solutions will be provided in Python and Matlab, and we may not be able to help you effectively if you use a different language. We therefore recommend that you use either Python or Matlab. You should also do exercise 0 to get familiar with language features and libraries that are useful for the exercises.

Instructions

This exercise is **approved** / **not approved**. To get your exercise approved, submit your answers to the questions as a **PDF** to Blackboard. You **don't** have to submit your code. Feel free to use LaTeX, Word, handwritten scans, or any other tool to write your answers.

Getting help

If you have questions about the exercise, you can:

- Post a question on Piazza (you can post anonymously if you want)
- Ask for help during the tutorial sessions (see Piazza for time and place)

Relevant resources

Below are the relevant slides and chapters from the course syllabus for this exercise (see Piazza for how to access the course materials).

- Lecture 4
- *Multiple View Geometry in Computer Vision*
  - Ch. 2.3: Projective transformations
  - Ch. 4.1: The Direct Linear Transformation (DLT) algorithm

Provided code

For task 2 and task 3, the zip includes a template Python and Matlab script (`main.py/m`) which takes care of loading the images and marker data, and generates the required figures. Your job will be to implement the missing function stubs in that file.

1 Homography from observing a planar object

Consider a planar object located in the scene. Without loss of generality, let points in the object coordinate frame be of the form $(X, Y, 0)^T$. Under the pinhole camera model, the pixel coordinate (u, v) of a point on the object is given by

$$u = c_x + f_x X^c / Z^c \tag{1}$$
$$v = c_y + f_y Y^c / Z^c \tag{2}$$

where

$$\begin{bmatrix} X^c \\ Y^c \\ Z^c \end{bmatrix} = \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} X \\ Y \\ 0 \end{bmatrix} + \underbrace{\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}}_{\mathbf{t}} \tag{3}$$

is the point in camera coordinates, and $(\mathbf{R}, \mathbf{t})$ is the rotation and translation between the object and camera frames. From the above, we obtain the following relationship between normalized image coordinates (x, y) and points on the plane (X, Y):

$$x = X^c / Z^c = \frac{u - c_x}{f_x} = \frac{r_{11} X + r_{12} Y + t_x}{r_{31} X + r_{32} Y + t_z} \tag{4}$$
$$y = Y^c / Z^c = \frac{v - c_y}{f_y} = \frac{r_{21} X + r_{22} Y + t_y}{r_{31} X + r_{32} Y + t_z} \tag{5}$$

The matrix

$$\mathbf{H} = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \tag{6}$$

defines a *homography* or *projective transformation.* [1]

(a) Using the definition of homogeneous image coordinates, $x = \tilde{x}/\tilde{z}$ and $y = \tilde{y}/\tilde{z}$, show that $\mathbf{H}$ defines a linear transformation between homogeneous image coordinates and points on the plane, i.e. that the following is equivalent to (4)-(5):

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = \mathbf{H} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{7}$$

(b) Explain why we say that the homography is only defined "up-to-scale".

2 The Direct Linear Transformation

The Direct Linear Transformation (DLT) can be used to estimate a homography $\mathbf{H}$ from a set of correspondences $(x_i, y_i) \leftrightarrow (X_i, Y_i)$. It works by rearranging equations (4)-(5) into a linear system by multiplying by the denominator on both sides, giving:

$$(r_{31}X + r_{32}Y + t_z)x = r_{11}X + r_{12}Y + t_x \tag{8}$$
$$(r_{31}X + r_{32}Y + t_z)y = r_{21}X + r_{22}Y + t_y \tag{9}$$

Rearranging terms gives:

$$r_{11}X + r_{12}Y + t_x - (r_{31}X + r_{32}Y + t_z)x = 0 \tag{10}$$
$$r_{21}X + r_{22}Y + t_y - (r_{31}X + r_{32}Y + t_z)y = 0 \tag{11}$$

Now, if we let $\mathbf{h}$ be a vector containing the unknown entries of $\mathbf{H}$:

$$\mathbf{h} = \begin{bmatrix} r_{11} & r_{12} & t_x & r_{21} & r_{22} & t_y & r_{31} & r_{32} & t_z \end{bmatrix}^T \tag{12}$$

then the above can be rearranged into a system of linear equations of the form

$$\mathbf{A}\mathbf{h} = \mathbf{0} \tag{13}$$

where

$$\mathbf{A} = \begin{bmatrix} X & Y & 1 & 0 & 0 & 0 & -Xx & -Yx & -x \\ 0 & 0 & 0 & X & Y & 1 & -Xy & -Yy & -y \end{bmatrix} \tag{14}$$

Equation (13) is called a homogeneous system. Unlike non-homogeneous systems ($\mathbf{A}\mathbf{h} = \mathbf{b}, \mathbf{b} \neq \mathbf{0}$), homogeneous systems always have the trivial solution $\mathbf{h} = \mathbf{0}$, which is of no interest. It's shown in the curriculum that $\mathbf{A}$ should have rank 8 for there to be a unique (up to scale) non-zero solution $\mathbf{h}$, and that this can be achieved by stacking the equations from $n \geq 4$ non-collinear points:

$$\mathbf{A} = \begin{bmatrix} X_1 & Y_1 & 1 & 0 & 0 & 0 & -X_1x_1 & -Y_1x_1 & -x_1 \\ 0 & 0 & 0 & X_1 & Y_1 & 1 & -X_1y_1 & -Y_1y_1 & -y_1 \\ & & & & & \vdots & & & \\ X_n & Y_n & 1 & 0 & 0 & 0 & -X_nx_n & -Y_nx_n & -x_n \\ 0 & 0 & 0 & X_n & Y_n & 1 & -X_ny_n & -Y_ny_n & -y_n \end{bmatrix} \tag{15}$$

The null-space of $\mathbf{A}$ is then 1-dimensional, and a solution $\mathbf{h}$ can be obtained as any non-zero scalar multiple of the null-space vector. In practice, due to noisy correspondences, $\mathbf{A}$ will not be *exactly* rank 8, and one solves the following least squares problem instead:

$$\arg\min_{\mathbf{h}} ||\mathbf{A}\mathbf{h}|| \text{ subject to } ||\mathbf{h}|| = 1 \tag{16}$$

The solution $\mathbf{h}$ is then obtained from the singular value decomposition (SVD) of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ as the column of $\mathbf{V}$ corresponding to the smallest singular value. [2]

Note however that the solution to (13) is only defined up to scale. This means that the entries in the *solved-for* $\mathbf{h}$ cannot necessarily be interpreted as the original rotation matrix and translation vector components.

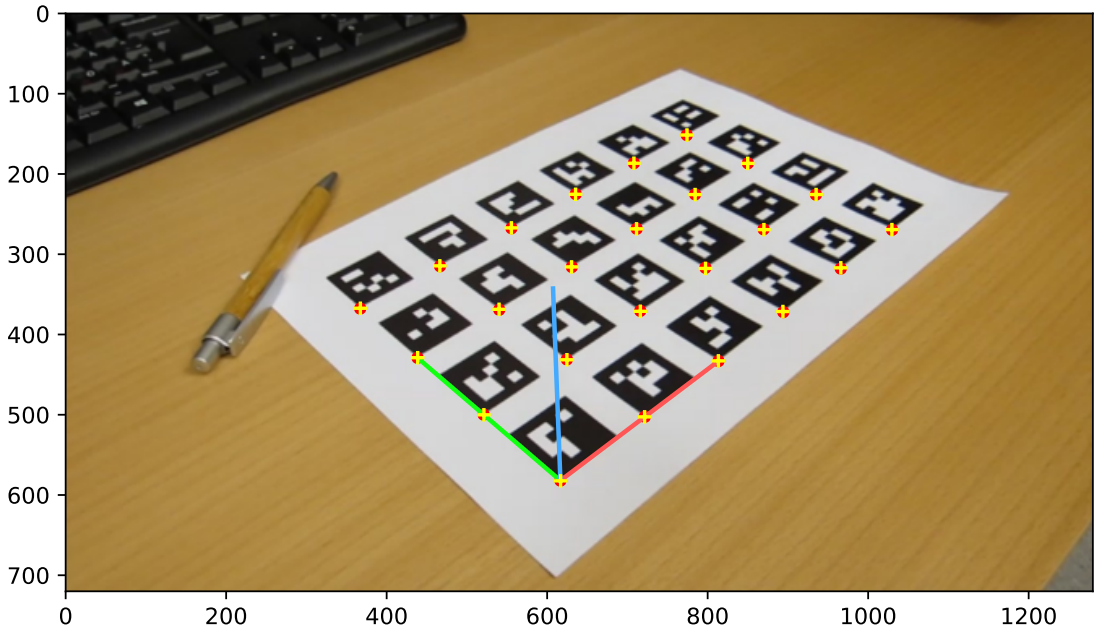

Figure 1: Frame 0 (`data/video0000.jpg`) from the dataset. Detected markers (+) and predicted markers (red circle) using homography.

The following dataset is included in the zip for this exercise:

- `data/cameraK`: Contains the camera intrinsic matrix $\mathbf{K}$.
- `data/video0000.jpg...video0022.jpg`: Image sequence.
- `data/model.txt`: Object coordinates $(X_i, Y_i, 1)$ in centimeters.
- `data/markers.txt`: Detected markers, one row per image. Each row contains 24 tuples of the form (m_i, u_i, v_i), where $m_i = 1$ if the marker associated with object coordinate i was detected, and (u_i, v_i) is the marker corner in pixel coordinates.

The provided code takes care of reading in images and the marker data, and generating the required figure.

(a) Write a function that estimates the homography $\mathbf{H}$ from a set of matched points $(x_i, y_i) \leftrightarrow (X_i, Y_i)$ using the Direct Linear Transformation.

Use `numpy.linalg.svd` (Python) or `svd` (Matlab) to compute the singular value decomposition. Test your function by computing the marker pixel coordinates predicted by the homography (this is done for you in the provided script). Check that they fall close to the detected markers. **Include a figure of your results on image 8 (`data/video0008.jpg`).**

3 Extracting $\mathbf{R}, \mathbf{t}$ from $\mathbf{H}$

Once we have estimated $\mathbf{H}$, we can decompose it into the original $(\mathbf{R}, \mathbf{t})$ by observing that equation (6) contains the two rotation matrix columns and the translation vector. The third column of the rotation matrix is not present, but if we know two columns, the third can always be obtained using the cross product:

$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$$

Recall however that we were only able to solve for $\mathbf{h}$ up to scale. The estimated $\mathbf{H}$ will therefore in general not satisfy equation (6), but instead

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \lambda \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \tag{17}$$

for some unknown scaling factor λ. [3] We can identify the scaling factor by utilizing the fact that the columns of a rotation matrix should be unit-length. Hence,

$$\sqrt{h_{11}^2 + h_{21}^2 + h_{31}^2} = \sqrt{h_{12}^2 + h_{22}^2 + h_{32}^2} = \pm\lambda \tag{18}$$

The rotation matrix entries and translation vector can then be extracted. Note that we can't recover the sign of λ, so we get two possible solutions for the pose.

(a) Write a function that returns the two possible pose decompositions from $\mathbf{H}$.

(b) Only one of the returned poses is physically possible in the sense that the plane is visible. For this particular dataset, the origin of the plane is always in front of the camera. Use what this implies about the z-component of the correct translation $\mathbf{t}$ and write a function that chooses between the two solutions.

(c) Test your function on the provided dataset by drawing the coordinate frame axes of the object plane as in Fig. 1. Check that the coordinate frame satisfies the right-hand rule in each image.

Include a figure of your results on image 5 (`data/video0005.jpg`).

Notes

[1] Homographies (projective transformations) are introduced in Ch. 2.3 in *Multiple View Geometry in Computer Vision*. In general, a homography is a mapping between two homogeneous 2D coordinates:

$$\tilde{\mathbf{x}}' = \mathbf{H}\tilde{\mathbf{x}}$$

where $\mathbf{H}$ is an arbitrary 3x3 matrix. Since $\mathbf{H}$ has nine entries, but any scalar multiple defines the same mapping, a homography has eight degrees of freedom. Note that equation (6) lets us construct a homography if we know the pose of the camera relative to a plane. However, equation (6) is **not** a general expression for a homography, since it has fewer than eight degrees of freedom. Later in the course we'll see that there is also a homography that maps between two camera images of a planar scene, which yields a different expression from (6).

[2]This is shown in *Multiple View Geometry in Computer Vision* Appendix A5.3: Least-squares solution of homogeneous equations. In Python and Matlab, the column of $\mathbf{V}$ corresponding to the smallest singular value is always the last column, as the singular values are automatically ordered by decreasing magnitude.

[3]Because of noise in data this relationship will in general not be satisfied either and the so-computed rotation matrix will not satisfy the properties of a rotation matrix. There are several ways to address this, for example: orthonormalizing the result, or computing the closest rotation matrix in the sense of the Frobenius norm (see Appendix C in *A Flexible New Technique for Camera Calibration* by Zhengyou Zhang).