Zhang's Camera Calibration Algorithm: In-Depth Tutorial and Implementation

Technical Report · May 2016

CITATIONS
12

READS
11,285

1 author:

Wilhelm Burger
Fachhochschule Oberösterreich
184 PUBLICATIONS **1,421** CITATIONS

SEE PROFILE

Zhang's Camera Calibration Algorithm: In-Depth Tutorial and Implementation

by

Wilhelm Burger

wilhelm.burger@fh-hagenberg.at

Technical Report HGB16-05

16th May, 2016 (revised April 2020)

Department of Digital Media

University of Applied Sciences Upper Austria, School of Informatics, Communications and Media, Softwarepark 11, 4232 Hagenberg, Austria

www.fh-hagenberg.at

Abstract

This report details the algorithmic steps involved in the well-known camera calibration method by Zhang and describes an associated open-source Java implementation that depends only upon the Apache Commons Math library.

Index Terms: Computer vision, camera calibration, Zhang's method, camera projection, imaging geometry, image rectification, Java implementation.

Software Repository: https://github.com/imagingbook/imagingbook-calibrate

Citation

Wilhelm Burger: *Zhang's Camera Calibration Algorithm: In-Depth Tutorial and Implementation*, Technical Report HGB16-05, University of Applied Sciences Upper Austria, School of Informatics, Communications and Media, Dept. of Digital Media, Hagenberg, Austria, May 2016. https://www.researchgate.net/publication/303233579_Zhang's_Camera_Calibration_Algorithm_In-Depth_Tutorial_and_Implementation.

```
@techreport{BurgerCalibration2016,
            author =  {Burger, Wilhelm},
             title =  {Zhang's Camera Calibration Algorithm:  In-Depth Tutorial
                       and Implementation},
          language =  {english},
       institution =  {University of Applied Sciences Upper Austria, School of
                       Informatics, Communications and Media, Dept. of Digital
                       Media},
           address =  {Hagenberg, Austria},
            number =  {HGB16-05},
              year =  {2016},
             month =  {05},
               url =  {https://www.researchgate.net/publication/303233579_Zhang'
                       s_Camera_Calibration_Algorithm_In-
                       Depth_Tutorial_and_Implementation}
}
```

1 Introduction

Accurate knowledge of the image projection parameters is an essential prerequisite for any kind of quantitative geometric measurement in computer vision. The real projection parameters depend on numerous technical elements and are usually not provided by manufacturers of imaging systems. Also, e. g., in the case of cameras equipped with zoom lenses, the projection parameters may be variable.

Many approaches to camera calibration exist (see, e. g., [9, p. 226]) using different strategies with regard to what about the 3D scene is known. Some approaches make use of a special, calibrated 3D setup (*calibration rig*), where the position of all 3D points and the camera center are known. Other approaches, such as the one by Zhang described here, use multiple views of a 3D pattern of known structure but unknown position and orientation in space. Finally, calibration methods exist that make no assumptions about the 3D structure of the scene, using multiple views of arbitrary, rigid structures. This is commonly called "self calibration". In this case the intrinsic camera parameters *and* the extrinsic viewing parameters (3D structure) are recovered together. Based on the imaging model described in Sec. 2, the following parameters are recovered:

- The intrinsic camera parameters, i.e., the inner transformations of the camera, including focal length, position of the principal point, sensor scale and skew.
- The parameters of the non-linear lens distortion.
- The external transformation parameters (3D rotation and translation) for each of the given views of the reference pattern.

2 The perspective projection model

This section describes the underlying projection process from 3D world points to 2D sensor coordinates and outlines the associated notation.

2.1 The pinhole camera model

The simple and well-known pinhole camera model (see, e. g., [2, Chap. 1]) is used to describe the projection of 3D world points onto the camera's sensor plane. We assume that the image plane is positioned in *front* of the optical center, thus the image is not upside-down. The image plane is positioned at the distance f from the optical center $\mathcal{C} = (0, 0, 0)^\mathsf{T}$, perpendicular to the optical axis. The optical center $\mathcal{C}$ is the origin of the 3D camera coordinate system. The optical axis aligns with the Z-axis of the coordinate system and intersects the image plane at $(0, 0, f)^\mathsf{T}$. Throughout this text, we use the definitions listed in Table 1.

We assume that initially the camera coordinate system is identical to the world coordinate system (we later remove this constraint and use two separate coordinate systems). From similar triangles, every 3D point $\boldsymbol{X} = (X, Y, Z)^\mathsf{T}$

Table 1: List of symbols and notation used in this document.

$\boldsymbol{X} = (X, Y, Z)^\mathsf{T}$	a 3D world point
$\mathcal{X} = (\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1})$	the 3D target points, with $\boldsymbol{X}_j = (X, Y, 0)^\mathsf{T}$
$\underline{\boldsymbol{X}} = \hom(\boldsymbol{X})$	*homogeneous* coordinate for a *Cartesian* coordinate $\boldsymbol{X}$
$\boldsymbol{X} = \hom^{-1}(\underline{\boldsymbol{X}})$	*Cartesian* coordinate for a *homogeneous* coordinate $\underline{\boldsymbol{X}}$
$\boldsymbol{x} = (x, y)^\mathsf{T}$	a projected 2D point in the normalized image plane
$\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y})^\mathsf{T}$	a 2D point after lens distortion (in the normalized image plane)
$\boldsymbol{u} = (u, v)^\mathsf{T}$	a projected 2D sensor point
$\dot{\boldsymbol{u}} = (\dot{u}, \dot{v})^\mathsf{T}$	an observed 2D sensor point
$\dot{\boldsymbol{U}}_i = (\dot{\boldsymbol{u}}_{i,0}, \ldots, \dot{\boldsymbol{u}}_{i,N-1})$	the observed sensor points for view i
$\dot{\mathcal{U}} = (\dot{\boldsymbol{U}}_0, \ldots, \dot{\boldsymbol{U}}_{M-1})$	the observed sensor points for all M views
$\mathbf{A}$	intrinsic camera matrix (Eqn. (15))
$\boldsymbol{a}$	vector of intrinsic camera parameters, including the distortion coefficients (Eqn. (120))
f	focal length
$\mathbf{R}$	a 3×3 rotation matrix
$\boldsymbol{\rho} = (\rho_x, \rho_y, \rho_z)$	a 3D rotation (Rodrigues) vector (Eqn. (126))
$\boldsymbol{t} = (t_x, t_y, t_z)^\mathsf{T}$	a 3D translation vector
$\mathbf{W} = (\mathbf{R} \mid \boldsymbol{t})$	an extrinsic camera (view) matrix (Eqn. (16))
$\boldsymbol{w}$	vector of 3D view parameters (Eqn. (121))
$\mathcal{W} = (\mathbf{W}_i)$	a sequence of camera views
$\check{P}(\mathbf{W}, \boldsymbol{X})$	3D $\mapsto$ 2D projection, which maps the 3D point $\boldsymbol{X}$ to normalized image coordinates with view parameters $\mathbf{W}$ (Eqn. (20))
$P(\mathbf{A}, \mathbf{W}, \boldsymbol{X})$	3D $\mapsto$ 2D projection, which maps the 3D point $\boldsymbol{X}$ to the associated 2D sensor point $\boldsymbol{u}$ with camera intrinsics $\mathbf{A}$ and view parameters $\mathbf{W}$ (Eqn. (24))
$P(\mathbf{A}, \boldsymbol{k}, \mathbf{W}, \boldsymbol{X})$	3D $\mapsto$ 2D projection, which includes camera lens distortion with parameters $\boldsymbol{k}$ (Eqn. (28)).

yields the relations

$$\frac{X}{Z} = \frac{x}{f}, \qquad \frac{Y}{Z} = \frac{y}{f} \tag{1}$$

and thus the projection point in the image plane is

$$x = f \cdot \frac{X}{Z}, \qquad y = f \cdot \frac{Y}{Z} \tag{2}$$

or, in vector notation,

$$\boldsymbol{x} = \begin{pmatrix} x \\ y \end{pmatrix} = \frac{f}{Z} \cdot \begin{pmatrix} X \\ Y \end{pmatrix}. \tag{3}$$

The world point $\boldsymbol{X}$ lies on a ray which passes through the optical center $\mathcal{C} = (0, 0, 0)^\mathsf{T}$ and the corresponding image point $\boldsymbol{x}_i$, i.e.,

$$\boldsymbol{X}_i = \lambda \cdot \big(\boldsymbol{x}_i - \mathcal{C}\big) = \lambda \cdot \boldsymbol{x}_i, \tag{4}$$

for some $\lambda > 1$.

2.2 The projection matrix

Equations (2) and (3) describe nonlinear transformations in the domain of Cartesian coordinates. Using *homogeneous coordinates*,[1] the perspective transformation can be written as a (linear) matrix equation

$$\begin{pmatrix} x \\ y \end{pmatrix} \equiv \frac{f}{Z} \cdot \begin{pmatrix} X \\ Y \end{pmatrix} \equiv \begin{pmatrix} fX/Z \\ fY/Z \\ 1 \end{pmatrix} \equiv \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \underbrace{\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\mathbf{M}_\mathrm{P}} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{5}$$

or, written more compactly,[2]

$$\boldsymbol{x} = \mathrm{hom}^{-1}\left(\mathbf{M}_\mathrm{P} \cdot \mathrm{hom}(\boldsymbol{X})\right). \tag{6}$$

The projection matrix $\mathbf{M}_\mathrm{P}$ can be decomposed into two matrices $\mathbf{M}_f$ and $\mathbf{M}_0$ in the form

$$\mathbf{M}_\mathrm{P} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \underbrace{\begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{M}_f} \cdot \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\mathbf{M}_0} = \mathbf{M}_f \cdot \mathbf{M}_0, \tag{7}$$

where $\mathbf{M}_f$ models the internals of the (ideal) pinhole camera with focal length f and $\mathbf{M}_0$ describes the transformation between the camera coordinates and the world coordinates. In particular, $\mathbf{M}_0$ is often referred to as the *standard* (or *canonical*) *projection matrix* [9], which corresponds to the simple viewing geometry (the optical axis being aligned with the Z-axis), which we have assumed so far.

2.3 Viewing under rigid motion

If the camera has its own (non-canonical) coordinate system, it observes 3D points that were subjected to rigid body motion, as described in Sec. A.2. Thus the projective transformation $\mathbf{M}_P$ (Eqn. (5)) is now applied to the modified (rotated and translated) points $\underline{\boldsymbol{X}}'$ instead of the original 3D points $\underline{\boldsymbol{X}} = \mathrm{hom}(\boldsymbol{X})$, that is,

$$\boldsymbol{x} = \mathrm{hom}^{-1}[\mathbf{M}_\mathrm{P} \cdot \underline{\boldsymbol{X}}'] = \mathrm{hom}^{-1}[\mathbf{M}_\mathrm{P} \cdot \mathbf{M}_\mathrm{rb} \cdot \mathrm{hom}(\boldsymbol{X})], \tag{8}$$

where the matrix $\mathbf{M}_\mathrm{rb}$ specifies some rigid body motion in 3D.[3] The complete perspective imaging transformation for the ideal pinhole camera with focal length f under rigid motion can thus be written as

$$\boldsymbol{x} = \mathrm{hom}^{-1}[\mathbf{M}_f \cdot \mathbf{M}_0 \cdot \mathbf{M}_\mathrm{rb} \cdot \mathrm{hom}(\boldsymbol{X})] \tag{9}$$

[1] See also Sec. A.1 in the Appendix.

[2] The operator $\underline{\boldsymbol{x}} = \mathrm{hom}(\boldsymbol{x})$ converts Cartesian coordinates to homogeneous coordinates. Inversely, $\boldsymbol{x} = \mathrm{hom}^{-1}(\underline{\boldsymbol{x}})$ denotes the conversion from homogeneous to Cartesian coordinates (see Sec. A.1 of the Appendix).

[3] See also Sec. A.2.3, Eqn. (170) in the Appendix.

or, by combining $\mathbf{M}_0$ and $\mathbf{M}_{\mathrm{rb}}$ into a single matrix,

$$\begin{aligned}\begin{pmatrix} x \\ y \end{pmatrix} &= \hom^{-1}\left[\begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}\right] \\ &= \hom^{-1}\left[\underbrace{\begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{M}_f} \cdot \underbrace{\left(\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{array}\right)}_{(\mathbf{R}\,\boldsymbol{t})} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}\right] \qquad (10) \\ &= \hom^{-1}\left[\mathbf{M}_f \cdot (\mathbf{R}\,\boldsymbol{t}) \cdot \hom(\boldsymbol{X})\right]. \qquad (11)\end{aligned}$$

In the special case of $f = 1$, $\mathbf{M}_f$ becomes the identity matrix and can thus be omitted, that is,

$$\boldsymbol{x} = \hom^{-1}\left[(\mathbf{R}\,\boldsymbol{t}) \cdot \hom(\boldsymbol{X})\right]. \qquad (12)$$

In the following, this is referred to as the "normalized projection".

2.4 Intrinsic camera parameters

A final small step is required to make the perspective imaging transformation in Eqn. (11) useful as a model for real cameras. In particular, we need to define how the continuous x/y-coordinates on the image plane map to actual pixel coordinates by taking into account

- the (possibly different) sensor scales s_x, s_y in x- and y-direction, respectively,
- the location of the image center $\boldsymbol{u}_c = (u_c, v_c)$ with respect to the image coordinate system (i. e., the optical axis), and
- the skewedness (diagonal distortion) s_θ of the image plane (which is usually negligible or zero).

The final **sensor coordinates** $\boldsymbol{u} = (u, v)^\mathsf{T}$ are obtained from the *normalized* image coordinates $\boldsymbol{x} = (x, y)^\mathsf{T}$ (see Eqn. (12)) as

$$\begin{aligned}\begin{pmatrix} u \\ v \end{pmatrix} &= \hom^{-1}\left[\underbrace{\begin{pmatrix} s_x & s_\theta & u_c \\ 0 & s_y & v_c \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{A}} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}\right] \qquad (13) \\ &= \hom^{-1}\left[\mathbf{A} \cdot \hom(\boldsymbol{x})\right], \qquad (14)\end{aligned}$$

where

$$\mathbf{A} = \begin{pmatrix} f s_x & f s_\theta & u_c \\ 0 & f s_y & v_c \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \alpha & \gamma & u_c \\ 0 & \beta & v_c \\ 0 & 0 & 1 \end{pmatrix} \qquad (15)$$

is the so-called **intrinsic** camera matrix. Taking into account these additional inner camera parameters, the complete perspective imaging transformation can

now be written as

$$\begin{pmatrix} u \\ v \end{pmatrix} = \hom^{-1}\left[\underbrace{\begin{pmatrix} \alpha & \gamma & u_c \\ 0 & \beta & v_c \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{A}} \cdot \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix}}_{\mathbf{W}=(\mathbf{R}|\boldsymbol{t})} \cdot \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{pmatrix}\right] \tag{16}$$

$$= \hom^{-1}\left[\mathbf{A} \cdot \mathbf{W} \cdot \hom(\boldsymbol{X})\right], \tag{17}$$

where $\mathbf{A}$ captures the **intrinsic** properties of the camera ("intrinsics"), and $\mathbf{W} = (\mathbf{R} \mid \boldsymbol{t})$ are the **extrinsic** parameters of the projection transformation ("extrinsics"). Note that we can calculate the projection in Eqn. (17) in two steps:

Step 1: Calculate the normalized projection $\boldsymbol{x} = (x, y)^\mathsf{T}$ (see Eqn. (12)):

$$\begin{pmatrix} x \\ y \end{pmatrix} = \hom^{-1}\left[\mathbf{W} \cdot \hom(\boldsymbol{X})\right] \tag{18}$$

$$= \hom^{-1}\left[\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \cdot \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{pmatrix}\right] \tag{19}$$

$$= \check{P}(\mathbf{W}, \boldsymbol{X}). \tag{20}$$

Step 2: Map from normalized coordinates $\boldsymbol{x}$ to sensor coordinates $\boldsymbol{u} = (u, v)^\mathsf{T}$ by the affine transformation $\mathbf{A}$ (see Eqn. (14)):

$$\begin{pmatrix} u \\ v \end{pmatrix} = \hom^{-1}\left[\mathbf{A} \cdot \hom(\boldsymbol{x})\right] = \mathbf{A}' \cdot \hom(\boldsymbol{x}) \tag{21}$$

$$= \hom^{-1}\left[\begin{pmatrix} \alpha & \gamma & u_c \\ 0 & \beta & v_c \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}\right] = \underbrace{\begin{pmatrix} \alpha & \gamma & u_c \\ 0 & \beta & v_c \end{pmatrix}}_{\mathbf{A}'} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{22}$$

where $\mathbf{A}'$ is the upper 2×3 submatrix of $\mathbf{A}$. Note that, by using $\mathbf{A}'$, no explicit conversion to Cartesian coordinates ($\hom^{-1}$) is required in Eqn. (22). $\mathbf{A}$ and $\mathbf{A}'$ are affine mappings in 2D. Combining the two steps above we can summarize the whole 3D to 2D projection process, (from world coordinates $\boldsymbol{X}$ to sensor coordinates $\boldsymbol{u}$) in a single expression,

$$\boldsymbol{u} = \mathbf{A}' \cdot \hom[\boldsymbol{x}] = \mathbf{A}' \cdot \hom\big[\hom^{-1}\left[\mathbf{W} \cdot \hom(\boldsymbol{X})\right]\big] = P(\mathbf{A}, \mathbf{W}, \boldsymbol{X}). \tag{23}$$

We will refer to $P(\mathbf{A}, \mathbf{W}, \boldsymbol{X})$, as defined in Eqn. (23), as the *projection function*, which maps the 3D point $\boldsymbol{X} = (X, Y, Z)^\mathsf{T}$ (defined in world coordinates) to the 2D sensor point $\boldsymbol{u} = (u, v)^\mathsf{T}$, using the intrinsic parameters $\mathbf{A}$ and the extrinsic (view) parameters $\mathbf{W}$. This function can be separated into two component functions in the form

$$P(\mathbf{A}, \mathbf{W}, \boldsymbol{X}) = \begin{pmatrix} P_\mathrm{x}(\mathbf{A}, \mathbf{W}, \boldsymbol{X}) \\ P_\mathrm{y}(\mathbf{A}, \mathbf{W}, \boldsymbol{X}) \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} = \boldsymbol{u}, \tag{24}$$

We will build on this notation in the following steps.

The primary goal of camera calibration is to determine the unknown intrinsic properties of the camera i. e., the elements of the matrix $\mathbf{A}$ (as detailed in Sec. 3).

2.5 Lens distortion

So far we have relied on the naive pinhole camera model which exhibits no distortions beyond the projective transformation described above. Real cameras are built with lenses instead of pinholes and these introduce additional geometric distortions that include two main components [8, p 342]:

- **Decentering errors** caused by a displacement of the lens center from the optical axis (this is mostly taken care of by the variable offset (u_c, v_c) in Eqn. (13));
- **Radial distortion** caused by variations in light refractions, which is typically apparent in wide-angle lenses ("barrel distortion").

While lens distortion is a complex physical phenomenon in general, it is usually modeled with sufficient accuracy as a single-variable polynomial function $D(r)$ of the radial distance r from the lens center [8, p 343] (see Sec. 2.5.2, Eqn. (32)).

2.5.1 Where does the lens distortion come in?

Lens distortion affects the normalized projection coordinates $\boldsymbol{x}$, i. e., before the image-to-sensor transformation (defined by the intrinsic camera parameters) is applied. Before we investigate the actual distortion model, we define a general distortion function $\text{warp}\colon \mathbb{R}^2 \mapsto \mathbb{R}^2$, which maps an undistorted 2D coordinate $\boldsymbol{x}$ to a distorted 2D coordinate $\tilde{\boldsymbol{x}}$ (again in the normalized projection plane) by

$$\tilde{\boldsymbol{x}} = \text{warp}(\boldsymbol{x}, \boldsymbol{k}), \tag{25}$$

where $\boldsymbol{k}$ is a vector of distortion parameters. With this definition, we can reformulate the projection process in Eqn. (23) to include the lens distortion as

$$\begin{aligned} \boldsymbol{u} &= \mathbf{A}' \cdot \hom\big[\tilde{\boldsymbol{x}}\big] = \mathbf{A}' \cdot \hom\big[\text{warp}(\boldsymbol{x}, \boldsymbol{k})\big] && (26)\\ &= \mathbf{A}' \cdot \hom\big[\text{warp}\big(\underbrace{\hom^{-1}\left[\mathbf{W} \cdot \hom(\boldsymbol{X})\right]}_{\boldsymbol{x}}, \boldsymbol{k}\big)\big] && (27)\\ &= P(\mathbf{A}, \boldsymbol{k}, \mathbf{W}, \boldsymbol{X}). && (28) \end{aligned}$$

In the following, we describe how the warp() function (referenced in Eqns. (26)–(27)) is specified and calculated.

2.5.2 Radial distortion model

A *radial* model is most commonly used for correcting geometric lens distortions. By radial distortion we understand that the displacement is restricted to radial lines emanating from the image center; the amount of *radial displacement* (inwards or outwards) is a function of the radius only. In the normalized projection plane, the optical axis intersects the image plane at $\boldsymbol{x}_c = (0, 0)$, which

is assumed to be the center of the lens distortion. The radial distance r_i of a projected point $\boldsymbol{x} = (x, y)^\mathsf{T}$ from the center can thus be simply calculated as

$$r_i = \|\boldsymbol{x}_i - \boldsymbol{x}_c\| = \|\boldsymbol{x}_i\| = \sqrt{x_i^2 + y_i^2}. \tag{29}$$

The distortion is assumed to be *radially symmetric*, i. e., it only depends on the original radius r_i of a given point $\boldsymbol{x}_i$, as defined in Eqn. (29). The distortion model can thus be specified by a single-variable function $D(r, \boldsymbol{k})$, such that the distorted radius is

$$\tilde{r} = f_{\text{rad}}(r) = r \cdot [1 + D(r, \boldsymbol{k})]. \tag{30}$$

Consequently (see Eqn. (25)), the warped projection point is $\tilde{\boldsymbol{x}}_i = \text{warp}(\boldsymbol{x}_i, \boldsymbol{k})$, with

$$\text{warp}(\boldsymbol{x}_i, \boldsymbol{k}) = \boldsymbol{x}_i \cdot [1 + D(\|\boldsymbol{x}_i\|, \boldsymbol{k})]. \tag{31}$$

The function $D(r, \boldsymbol{k})$ specifies the (positive or negative) radial deviation for a given radius r. A simple but effective radial distortion model is based on the polynomial function

$$D(r, \boldsymbol{k}) = k_0 \cdot r^2 + k_1 \cdot r^4 = \boldsymbol{k} \cdot \begin{pmatrix} r^2 \\ r^4 \end{pmatrix}, \tag{32}$$

with the (unknown) coefficients $\boldsymbol{k} = (k_0, k_1)$, as illustrated in Fig. 1.[4] Note that, if $k_0 = k_1 = 0$, then also $D(r, \boldsymbol{k}) = 0$ and there is no distortion.

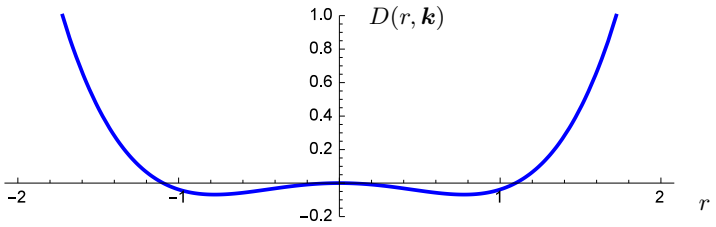


Figure 1: Plot of the radial deviation function $D(r, \boldsymbol{k})$ in Eqn. (32) for coefficients $\boldsymbol{k} = (k_0, k_1) = (-0.2286, 0.190335)$.

2.6 Summary of the projection process

In summary, the following steps model the complete projection process (see Fig. 3):

[4] Other formulations of the radial distortion function can be found in the literature. For example, $D(r) = k_0 \cdot r^2 + k_1 \cdot r^4 + k_2 \cdot r^6$ is used in [8, p. 343] or $D(r) = k_0 \cdot r + k_1 \cdot r^2 + k_2 \cdot r^3 + k_3 \cdot r^4$ by Devernay and Faugeras (mentioned in [9, p. 58]). For a detailed analysis of radial distortion in fisheye lenses see [3].

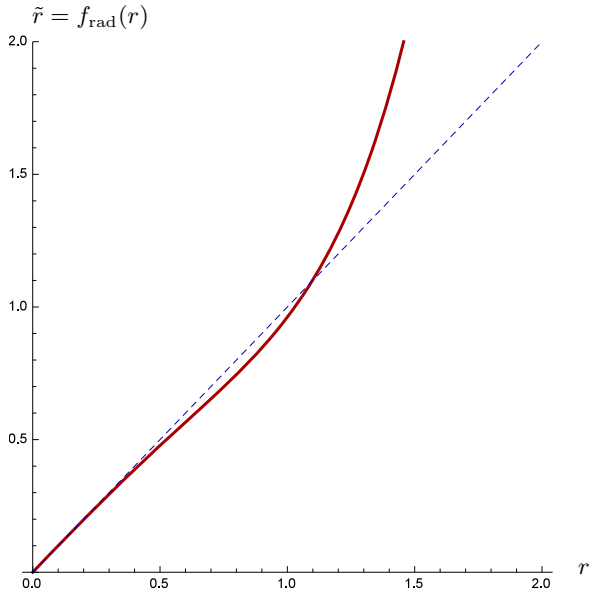


Figure 2: Plot of the radial distortion $\tilde{r} = f_{\text{rad}}(r) = r \cdot [1 + D(r, \boldsymbol{k})]$ in Eqn. (30) for the same parameters ($\boldsymbol{k}$) as in Fig. 1.

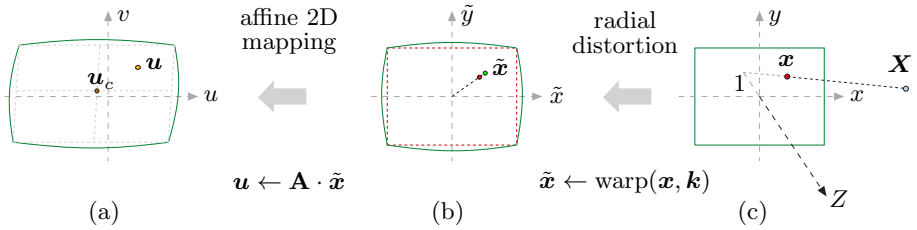


Figure 3: Summary of the projection chain (from right to left). In (c) the 3D point $\boldsymbol{X}$ (in camera coordinates) is projected (with $f = 1$) onto the "ideal" image plane to the normalized coordinates $\boldsymbol{x} = (x, y)^\mathsf{T}$. Radial lens distortion in (b) maps point $\boldsymbol{x}$ to $\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y})^\mathsf{T}$. The affine mapping specified by the intrinsic camera transformation (matrix $\mathbf{A}$) finally yields the observed sensor image coordinates $\boldsymbol{u} = (u, v)^\mathsf{T}$ in (a).

1. **World-to-camera transformation:** Given a point $(X, Y, Z)^\mathsf{T}$, expressed in 3D world coordinates, its position in the 3D camera coordinate system is specified by the *viewing transformation* $\mathbf{W}$ (see Eqn. (168)),

$$\begin{pmatrix} X' \\ Y' \\ Z' \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{W}} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \tag{33}$$

in homogeneous coordinates, or simply

$$\boldsymbol{X}' = \mathbf{W} \cdot \boldsymbol{X}. \tag{34}$$

2. **Projection onto the "normalized" (ideal) image plane:** The perspective projection from the 3D-point $\boldsymbol{X}' = (X', Y', Z')^\mathsf{T}$ (in 3D camera co-ordinates) onto continuous, normalized 2D coordinates $\boldsymbol{x} = (x, y)^\mathsf{T}$ on the image plane is defined (see Eqn. (5)) as

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{Z'} \cdot \begin{pmatrix} X' \\ Y' \end{pmatrix} = \hom^{-1}\Big[\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} X' \\ Y' \\ Z' \\ 1 \end{pmatrix} \Big] = \check{P}(\mathbf{W}, \boldsymbol{X}) \,, \quad (35)$$

which is equivalent to an ideal pinhole projection with focal length $f = 1$.

3. **Radial lens distortion:** The normalized 2D projection coordinates $\boldsymbol{x} = (x, y)^\mathsf{T}$ are subjected to a non-linear radial distortion with respect to the optical center $\boldsymbol{x}_c = (0, 0)^\mathsf{T}$, expressed by the mapping

$$\tilde{\boldsymbol{x}} = \mathrm{warp}(\boldsymbol{x}, \boldsymbol{k}) \qquad \text{or} \qquad \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \cdot [1 + D(r, \boldsymbol{k})], \quad (36)$$

with $r = \sqrt{x^2 + x^2}$ and $D(r, \boldsymbol{k})$ as defined in Eqn. (32). $\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y})^\mathsf{T}$ are the lens-distorted 2D coordinates—still in the normalized image plane.

4. **Affine 2D transformation to sensor coordinates:** The normalized projection points are finally mapped to the scaled and skewed sensor coordinates (see Eqn. (13)) by the affine transformation

$$\boldsymbol{u} = \mathbf{A}' \cdot \hom[\tilde{\boldsymbol{x}}] \qquad \text{or} \qquad \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \alpha & \gamma & u_c \\ 0 & \beta & v_c \end{pmatrix} \cdot \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ 1 \end{pmatrix}, \quad (37)$$

where α, β, γ, u_c, v_c are the intrinsic camera parameters (see Eqns. (15) and (22)).

3 Plane-based self calibration

The popular camera calibration method by Zhang [14, 15] uses a few (at least two) views of a planar calibration pattern, called "model" or "target", whose layout and metric dimensions are precisely known.[5] The calibration procedure works roughly as follows:

1. Images $I_0, \ldots, I_{M-1}$ of the model are taken under different views by either moving the model or the camera (or both).
2. From each image I_i $(i = 0, \ldots, M-1)$, N sensor points $\dot{\boldsymbol{u}}_{i,0}, \ldots, \dot{\boldsymbol{u}}_{i,j}, \ldots, \dot{\boldsymbol{u}}_{i,N-1}$ are extracted (observed), assumed to be in 1:1 correspondence with the points on the model plane.
3. From the observed points, the associated *homographies* $\mathbf{H}_0, \ldots, \mathbf{H}_{M-1}$ (linear mappings from the model points and the observed 2D image points) are estimated for each view i. (see Sec. 3.2).

[5] A variant of this calibration technique is also implemented in OpenCV [7].

4. From the view homographies $\mathbf{H}_i$, the five intrinsic parameters (α, β, γ, u_c, v_c) of the camera are estimated using a closed-form (linear) solution, ignoring any lens distortion at this point. $M \geq 3$ views give a unique solution (up to an undetermined scale factor). If the sensor plane is assumed to be without skew (i. e., $\gamma = 0$, which is a reasonable assumption) then $N = 2$ images are sufficient. More views generally lead to more accurate results (see Sec. 3.3).
5. Once the camera intrinsics are known, the extrinsic 3D parameters $\mathbf{R}_i$, $\boldsymbol{t}_i$ are calculated for each camera view i (see Sec. 3.4).
6. The radial distortion parameters k_0, k_1 are estimated by linear least-squares minimization (see Sec. 3.5).
7. Finally, using the estimated parameter values as an initial guess, all parameters are refined by non-linear optimization over all M views (see Sec. 3.6).

These steps are explained in greater detail below (see [14] for a complete description and the list of symbols in Table 1).

3.1 Calibration model and observed views

The calibration model contains N reference points $\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1}$ whose 3D coordinates are known. The points are assumed to lie in the XY-plane, i. e., their Z-component is zero.

We assume that M different views (i. e., pictures) are taken of the model plane and we use $i = 0, \ldots, M-1$ to denote the i^{th} view of the model. From each camera picture I_i, N feature points are extracted, so we get the *observed sensor points*

$$\dot{\boldsymbol{u}}_{i,j} \in \mathbb{R}^2, \tag{38}$$

with view numbers $i = 0, \ldots, M-1$ and point numbers $j = 0, \ldots, N-1$. Note that every observed point $\dot{\boldsymbol{u}}_{i,j}$ must correspond to the associated model point $\boldsymbol{X}_j$. Thus the model points $\boldsymbol{X}_j$ and the image points $\dot{\boldsymbol{u}}_{i,j}$ must be supplied in the same order. It is essential that this condition is met, since otherwise the calibration will deliver invalid results.

3.2 Step 1: Estimating the homography for each view

Using Eqn. (17), the mapping (homography) between the observed image coordinates $\dot{\boldsymbol{u}}_{i,j} = (\dot{u}_{i,j}, \dot{v}_{i,j})^\mathsf{T}$ and the corresponding 3D point coordinates $\boldsymbol{X}_j$ can be expressed as

$$s \cdot \begin{pmatrix} \dot{u}_{i,j} \\ \dot{v}_{i,j} \\ 1 \end{pmatrix} = \mathbf{A} \cdot \left(\mathbf{R}_i \;\middle|\; \boldsymbol{t}_i \right) \cdot \begin{pmatrix} X_j \\ Y_j \\ Z_j \\ 1 \end{pmatrix} \tag{39}$$

or

$$s \cdot \underline{\dot{\boldsymbol{u}}}_{i,j} = \mathbf{A} \cdot \left(\mathbf{R}_i \;\middle|\; \boldsymbol{t}_i \right) \cdot \underline{\boldsymbol{X}}_j \tag{40}$$

(with homogeneous coordinates $\underline{\dot{\boldsymbol{u}}}, \underline{\boldsymbol{X}}$), where

$$\mathbf{A} = \begin{pmatrix} \alpha & \gamma & u_c \\ 0 & \beta & v_c \\ 0 & 0 & 1 \end{pmatrix} \tag{41}$$

represents the intrinsic camera parameters (common to every view) and and s is an arbitrary, non-zero scale factor. $\mathbf{R}_i$, $\boldsymbol{t}_i$ are the (extrinsic) 3D rotation matrix and the translation vector for the specific view i.

Since the model points $\boldsymbol{X}_j$ are assumed to lie in the XY-plane of the world coordinate system (i. e., $Z_j = 0$ for all j),[6] we can rewrite Eqn. (39) as

$$s \cdot \begin{pmatrix} \dot{u}_{i,j} \\ \dot{v}_{i,j} \\ 1 \end{pmatrix} = \mathbf{A} \cdot \begin{pmatrix} | & | & | & | \\ \boldsymbol{r}_{i,0} & \boldsymbol{r}_{i,1} & \boldsymbol{r}_{i,2} & \boldsymbol{t}_i \\ | & | & | & | \end{pmatrix} \cdot \begin{pmatrix} X_j \\ Y_j \\ 0 \\ 1 \end{pmatrix} = \mathbf{A} \cdot \begin{pmatrix} | & | & | \\ \boldsymbol{r}_{i,0} & \boldsymbol{r}_{i,1} & \boldsymbol{t}_i \\ | & | & | \end{pmatrix} \cdot \begin{pmatrix} X_j \\ Y_j \\ 1 \end{pmatrix}, \quad (42)$$

where $\boldsymbol{r}_{i,0}, \boldsymbol{r}_{i,1}, \boldsymbol{r}_{i,2}$ denote the three column vectors of the matrix $\mathbf{R}_i$. Note that $Z_j = 0$ makes the third vector ($\boldsymbol{r}_{i,2}$) redundant and it is therefore omitted in the right part of Eqn. (42). This is equivalent to a 2D *homography* mapping

$$s \cdot \begin{pmatrix} \dot{u}_{i,j} \\ \dot{v}_{i,j} \\ 1 \end{pmatrix} = \mathbf{H}_i \cdot \begin{pmatrix} X_j \\ Y_j \\ 1 \end{pmatrix}, \quad (43)$$

where s is a (undetermined) non-zero scale factor and $\mathbf{H}_i = \lambda \cdot \mathbf{A} \cdot \left(\begin{array}{c|c} \mathbf{R}_i & \boldsymbol{t}_i \end{array} \right)$ is a 3×3 homography matrix (λ being an arbitrary scalar value which can be ignored since we work with homogeneous coordinates). The matrix $\mathbf{H}_i$ is composed of the 3 column vectors $\boldsymbol{h}_{i,0}$, $\boldsymbol{h}_{i,1}$, $\boldsymbol{h}_{i,2}$, that is,

$$\mathbf{H}_i = \begin{pmatrix} | & | & | \\ \boldsymbol{h}_{i,0} & \boldsymbol{h}_{i,1} & \boldsymbol{h}_{i,2} \\ | & | & | \end{pmatrix} = \lambda \cdot \mathbf{A} \cdot \begin{pmatrix} | & | & | \\ \boldsymbol{r}_{i,0} & \boldsymbol{r}_{i,1} & \boldsymbol{t}_i \\ | & | & | \end{pmatrix}. \quad (44)$$

Thus the task is to determine the homography matrix $\mathbf{H}_i$ for a set of corresponding 2D points $(\dot{u}_{i,j}, \dot{v}_{i,j})^\mathsf{T}$ and $(X_j, Y_j)^\mathsf{T}$.

3.2.1 Homography estimation with the Direct Linear Transformation (DLT)

Among the several methods for estimating homography mappings, the DLT method is the simplest (see [5, Ch. 4]). It is also used in Zhang's original implementation. We assume that the two corresponding 2D points sequences, the *model points* $\mathcal{X} = (\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1})$, with $\boldsymbol{X}_j = (X_j, Y_j)$ and the associated (observed) *sensor points* $\dot{\boldsymbol{U}} = (\dot{\boldsymbol{u}}_0, \ldots, \dot{\boldsymbol{u}}_{N-1})$, with $\dot{\boldsymbol{u}}_j = (\dot{u}_j, \dot{v}_j)^\mathsf{T}$, are related by a homography transformation, that is (written in homogeneous coordinates),

$$\underline{\dot{\boldsymbol{u}}}_j = \mathbf{H} \cdot \underline{\boldsymbol{X}}_j \quad (45)$$

or

$$\begin{pmatrix} \underline{\dot{u}}_j \\ \underline{\dot{v}}_j \\ \underline{\dot{w}}_j \end{pmatrix} = \begin{pmatrix} H_{0,0} & H_{0,1} & H_{0,2} \\ H_{1,0} & H_{1,1} & H_{1,2} \\ H_{2,0} & H_{2,1} & H_{2,2} \end{pmatrix} \cdot \begin{pmatrix} \underline{X}_j \\ \underline{Y}_j \\ \underline{Z}_j \end{pmatrix}, \quad (46)$$

[6] This only means that we assume a moving camera instead of a moving object plane and makes no difference, because only the relative coordinates with respect to the camera is required for calibration.

for $j = 0, \ldots, N-1$. Without loss of generality (to show!), we can set $\underline{Z}_j = 1$ (such that $\underline{X}_j = X_j$, $\underline{Y}_j = Y_j$) and can thus rewrite Eqn. (46) as

$$\begin{pmatrix} \underline{\dot{u}}_j \\ \underline{\dot{v}}_j \\ \underline{\dot{w}}_j \end{pmatrix} = \begin{pmatrix} H_{0,0} & H_{0,1} & H_{0,2} \\ H_{1,0} & H_{1,1} & H_{1,2} \\ H_{2,0} & H_{2,1} & H_{2,2} \end{pmatrix} \cdot \begin{pmatrix} X_j \\ Y_j \\ 1 \end{pmatrix}. \tag{47}$$

In Cartesian coordinates, this corresponds to a pair of *non-linear* equations,

$$u_j = \frac{\underline{\dot{u}}_j}{\underline{\dot{w}}_j} = \frac{H_{0,0} \cdot X_j + H_{0,1} \cdot Y_j + H_{0,2}}{H_{2,0} \cdot X_j + H_{2,1} \cdot Y_j + H_{2,2}}, \tag{48}$$

$$v_j = \frac{\underline{\dot{v}}_j}{\underline{\dot{w}}_j} = \frac{H_{1,0} \cdot X_j + H_{1,1} \cdot Y_j + H_{1,2}}{H_{2,0} \cdot X_j + H_{2,1} \cdot Y_j + H_{2,2}}, \tag{49}$$

which can be rearranged to

$$\dot{u}_j \cdot (H_{2,0} \cdot X_j + H_{2,1} \cdot Y_j + H_{2,2}) = H_{0,0} \cdot X_j + H_{0,1} \cdot Y_j + H_{0,2}\,, \tag{50}$$

$$\dot{v}_j \cdot (H_{2,0} \cdot X_j + H_{2,1} \cdot Y_j + H_{2,2}) = H_{1,0} \cdot X_j + H_{1,1} \cdot Y_j + H_{1,2}\,, \tag{51}$$

and finally

$$\dot{u}_j \cdot X_j \cdot H_{2,0} + \dot{u}_j \cdot Y_j \cdot H_{2,1} + \dot{u}_j \cdot H_{2,2} - H_{0,0} \cdot X_j - H_{0,1} \cdot Y_j - H_{0,2} = 0, \tag{52}$$

$$\dot{v}_j \cdot X_j \cdot H_{2,0} + \dot{v}_j \cdot Y_j \cdot H_{2,1} + \dot{v}_j \cdot H_{2,2} - H_{1,0} \cdot X_j - H_{1,1} \cdot Y_j - H_{1,2} = 0. \tag{53}$$

This is a pair of *homogeneous* equations (since the right hand side is zero) that are *linear* in the unknown coefficients $H_{r,c}$ (although, due to the mixed terms, still non-linear w.r.t. the coordinates $\dot{u}_j, \dot{v}_j, X_j, Y_j$). By collecting the nine elements of the unknown homography matrix $\mathbf{H}$ into the vector

$$\boldsymbol{h} = (H_{0,0}, H_{0,1}, H_{0,2}, H_{1,0}, H_{1,1}, H_{1,2}, H_{2,0}, H_{2,1}, H_{2,2})^\mathsf{T}, \tag{54}$$

Eqns. (52) and (53) can be written in the form

$$\begin{pmatrix} -X_j & -Y_j & -1 & 0 & 0 & 0 & \dot{u}_j X_j & \dot{u}_j Y_j & \dot{u}_j \\ 0 & 0 & 0 & -X_j & -Y_j & -1 & \dot{v}_j X_j & \dot{v}_j Y_j & \dot{v}_j \end{pmatrix} \cdot \boldsymbol{h} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \tag{55}$$

for every corresponding point pair $(\dot{u}_j, \dot{v}_j) \leftrightarrow (X_j, Y_j)$. Thus, N point pairs, assumed to be related by the same homography $\mathbf{H}$, yield a system of $2N$ homogeneous linear equations in the form

$$\left(\begin{array}{ccccccccc} -X_0 & -Y_0 & -1 & 0 & 0 & 0 & \dot{u}_0 X_0 & \dot{u}_0 Y_0 & \dot{u}_0 \\ 0 & 0 & 0 & -X_0 & -Y_0 & -1 & \dot{v}_0 X_0 & \dot{v}_0 Y_0 & \dot{v}_0 \\ \hdashline -X_1 & -Y_1 & -1 & 0 & 0 & 0 & \dot{u}_1 X_1 & \dot{u}_1 Y_1 & \dot{u}_1 \\ 0 & 0 & 0 & -X_1 & -Y_1 & -1 & \dot{v}_1 X_1 & \dot{v}_1 Y_1 & \dot{v}_1 \\ \hdashline -X_2 & -Y_2 & -1 & 0 & 0 & 0 & \dot{u}_2 X_2 & \dot{u}_2 Y_2 & \dot{u}_2 \\ 0 & 0 & 0 & -X_2 & -Y_2 & -1 & \dot{v}_2 X_2 & \dot{v}_2 Y_2 & \dot{v}_2 \\ \hdashline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hdashline -X_{N-1} & -Y_{N-1} & -1 & 0 & 0 & 0 & \dot{u}_{N-1} X_{N-1} & \dot{u}_{N-1} Y_{N-1} & \dot{u}_{N-1} \\ 0 & 0 & 0 & -X_{N-1} & -Y_{N-1} & -1 & \dot{v}_{N-1} X_{N-1} & \dot{v}_{N-1} Y_{N-1} & \dot{v}_{N-1} \end{array}\right) \cdot \begin{pmatrix} H_{0,0} \\ H_{0,1} \\ H_{0,2} \\ H_{1,0} \\ H_{1,1} \\ H_{1,2} \\ H_{2,0} \\ H_{2,1} \\ H_{2,2} \end{pmatrix} = \left(\begin{array}{c} 0 \\ 0 \\ \hdashline 0 \\ 0 \\ \hdashline 0 \\ 0 \\ \hdashline \vdots \\ \hdashline 0 \\ 0 \end{array}\right) \tag{56}$$

or, in the usual matrix-vector notation,

$$\mathbf{M} \cdot \boldsymbol{h} = \mathbf{0}, \tag{57}$$

where $\mathbf{M}$ is a $2N \times 9$ matrix (with all elements being known constants), $\boldsymbol{d}$ is the vector of the nine unknowns, and $\mathbf{0}$ is the zero vector of length $2N$.

Solving homogeneous systems of linear equations: While Eqn. (57) looks quite similar to an ordinary system of linear equations of the form $\mathbf{M}\cdot\boldsymbol{x} = \boldsymbol{b}$, it cannot be solved in the usual way (without additional constraints), since it always has $\boldsymbol{h} = \mathbf{0}$ as a trivial (and thus useless) solution. However, Eqn. (57) can be solved by *singular-value decomposition* (SVD, [4, Sec. 6.11][1, Sec. 4.5.3][10, Sec. 2.6]) of the matrix $\mathbf{M}$, which separates $\mathbf{M}$ (of size $2N \times 9$) into the product of three matrices $\mathbf{U}, \mathbf{S}, \mathbf{V}$ in the form

$$\mathbf{M} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^\mathsf{T}. \tag{58}$$

Here[7] $\mathbf{U}$ is a $2N \times 2N$ (in this particular case) *unitary*[8] matrix, $\mathbf{S}$ is a $2N \times 9$ rectangular *diagonal* matrix with non-negative real values, and $\mathbf{V}$ is a 9×9 unitary matrix. The concrete decomposition of $\mathbf{M}$ is thus

$$\mathbf{M} = \underbrace{\begin{pmatrix} U_{0,0} & \cdots & U_{0,2N-1} \\ U_{1,0} & \cdots & U_{1,2N-1} \\ \vdots & \ddots & \vdots \\ U_{2N-1,0} & \cdots & U_{2N-1,2N-1} \end{pmatrix}}_{\mathbf{U}} \cdot \underbrace{\begin{pmatrix} s_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & s_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & s_2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & s_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & s_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & s_5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & s_6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & s_7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & s_8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}}_{\mathbf{S}} \cdot \underbrace{\begin{pmatrix} V_{0,0} & \cdots & V_{8,0} \\ V_{0,1} & \cdots & V_{8,1} \\ \vdots & \ddots & \vdots \\ V_{0,8} & \cdots & V_{8,8} \end{pmatrix}}_{\mathbf{V}^\mathsf{T}}. \tag{59}$$

The diagonal elements of $\mathbf{S}$, $s_0, \ldots, s_8$ are called the *singular values* of the decomposed matrix $\mathbf{M}$. Each singular value s_i has an associated column vector $\boldsymbol{u}_i$ in $\mathbf{U}$ (called a *left singular vector* of $\mathbf{M}$) and a dedicated *row* vector $\boldsymbol{v}_i^\mathsf{T}$ in $\mathbf{V}^\mathsf{T}$ (i. e., a *column* vector $\boldsymbol{v}_i$ in $\mathbf{V}$), called a *right singular vector*. Thus Eqn. (59) can be equally written as

$$\mathbf{M} = \begin{pmatrix} | & & | & & | \\ \boldsymbol{u}_0 & \cdots & \boldsymbol{u}_i & \cdots & \boldsymbol{u}_{2N-1} \\ | & & | & & | \end{pmatrix} \cdot \mathbf{S} \cdot \begin{pmatrix} — & \boldsymbol{v}_0^\mathsf{T} & — \\ & \vdots & \\ — & \boldsymbol{v}_i^\mathsf{T} & — \\ & \vdots & \\ — & \boldsymbol{v}_8^\mathsf{T} & — \end{pmatrix}, \tag{60}$$

where $\mathbf{U}$ consists of the column vectors $(\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{2N-1})$[9] and $\mathbf{V}$ is composed of the row vectors $(\boldsymbol{v}_0^\mathsf{T}, \ldots, \boldsymbol{v}_8^\mathsf{T})$.

The seeked-for solution to Eqn. (57), i. e., the unknown parameter vector $\boldsymbol{h}$, is finally found as the *right singular vector* $\boldsymbol{v}_k$ associated with the *smallest* singular value $s_k = \min(s_0, \ldots, s_8)$, that is,

$$\boldsymbol{h} = \boldsymbol{v}_k, \qquad \text{with} \quad k = \operatorname*{argmin}_{0 \le i < 9} s_i\,. \tag{61}$$

[7] There are several different notations in use regarding the size of the matrices involved in the SVD. We use the version adopted by Matlab and Mathematica (http://mathworld.wolfram.com/SingularValueDecomposition.html), where $\mathbf{S}$ has the same size as the original matrix $\mathbf{M}$ and $\mathbf{U}, \mathbf{V}$ are square.

[8] A square matrix $\mathbf{U}$ is called *unitary* if its column vectors are orthogonal, i. e., if $\mathbf{U} \cdot \mathbf{U}^\mathsf{T} = \mathbf{U}^\mathsf{T} \cdot \mathbf{U} = \mathbf{I}$.

[9] Note that the vectors $\boldsymbol{u}_i$ are zero for $i \ge 9$.

If the resulting homography transformation for the corresponding point sets is *exact*, the value of s_k is zero. This is generally the case when the homography is calculated from 4 corresponding point pairs, which is the required minimum number to solve for the eight degrees of freedom.

If more than 4 point pairs are involved, the system in Eqn. (57) is *overdetermined* (which is the usual case). Here the value of s_k indicates the residual or "goodness of fit" of the resulting homography. Of course, if the fit is exact for all point pairs, s_k will again be zero. In the case of an overdetermined system, the obtained solution minimizes Eqn. (57) in the least-squares sense, that is,

$$\|\mathbf{M} \cdot \boldsymbol{h}\|^2 \rightarrow \min . \tag{62}$$

In many common SVD implementations,[10] the singular values in $\boldsymbol{S}$ are arranged in *non-increasing* order (i. e., $s_i \geq s_{i+1}$), such that (in our case) s_8 comes out as the smallest value and $\boldsymbol{v}_8$ (the last column vector of $\boldsymbol{V}$) is the corresponding solution. For example, the following Java code segment shows an implementation with the *Apache Commons Math* (ACM) library:[11]

```
RealMatrix M;
SingularValueDecomposition svd = new SingularValueDecomposition(M);
RealMatrix V = svd.getV();
RealVector h = V.getColumnVector(V.getColumnDimension() - 1);
```

Note that the formulation in Eqn. (62) minimizes an *algebraic* residual that does not directly relate to the geometric projection error. This does not cause a problem in general, since the remaining errors are eliminated in the final, overall optimization step (see Sec. 3.6). However, minimizing the projection errors of the homographies at this stage (which is not done in Zhang's implementation) may help to improve the convergence of the final optimization. It requires nonlinear optimization, for which the above solution can serve as a good initial guess (see Sec. 3.2.3).

3.2.2 Normalization of input data

To improve numerical stability of the calculations, it is recommended [6] to normalize both 2D point sets $\mathcal{X}$ and $\dot{\boldsymbol{U}}$, before performing the homography estimation described in the previous section.

Normalization is accomplished by transforming each point set by a dedicated 3×3 normalization matrix $\mathbf{N}$ (in homogeneous coordinates), such that the transformed point set becomes centered at the origin and scaled to a standard diameter, i. e.,

$$\mathcal{X}' = \text{normalize}(\mathcal{X}) = (\mathbf{N}_\text{X} \cdot \boldsymbol{X}_0, \ldots, \mathbf{N}_\text{X} \cdot \boldsymbol{X}_{N-1}), \tag{63}$$

$$\dot{\boldsymbol{U}}' = \text{normalize}(\dot{\boldsymbol{U}}) = (\mathbf{N}_\text{U} \cdot \dot{\boldsymbol{u}}_0, \ldots, \mathbf{N}_\text{U} \cdot \dot{\boldsymbol{u}}_{N-1}). \tag{64}$$

See Sec. B of the Appendix for methods to calculate the normalization matrices $\mathbf{N}_\text{X}$ and $\mathbf{N}_\text{U}$. Homography estimation is then (analogous to Eqn. (45)) performed on the *normalized* point sets $\mathcal{X}', \dot{\boldsymbol{U}}'$, by calculating a matrix $\mathbf{H}'$ satisfying (in the least squares sense)

$$\mathbf{N}_\text{U} \cdot \underline{\dot{\boldsymbol{u}}}_j = \mathbf{H}' \cdot \mathbf{N}_\text{X} \cdot \underline{\boldsymbol{X}}_j, \tag{65}$$

[10] E. g., *Matlab*, *Mathematica*, *Apache Commons Math*.
[11] http://commons.apache.org/math/ (version 3.6)

for $j = 0, \ldots, N-1$. By substituting $\dot{\underline{\boldsymbol{u}}}_j$ (from Eqn. (45)) we get

$$\mathbf{N}_{\mathrm{U}} \cdot \mathbf{H} \cdot \underline{\boldsymbol{X}}_j = \mathbf{H}' \cdot \mathbf{N}_{\mathrm{X}} \cdot \underline{\boldsymbol{X}}_j \tag{66}$$

and then, by dropping $\underline{\boldsymbol{x}}_j$ on both sides,

$$\mathbf{N}_{\mathrm{U}} \cdot \mathbf{H} = \mathbf{H}' \cdot \mathbf{N}_{\mathrm{X}}. \tag{67}$$

Thus the de-normalized homography can be calculated (by multiplying both sides with $\mathbf{N}_{\mathrm{U}}^{-1}$) as

$$\mathbf{H} = \mathbf{N}_{\mathrm{U}}^{-1} \cdot \mathbf{H}' \cdot \mathbf{N}_{\mathrm{X}}. \tag{68}$$

3.2.3 Non-linear refinement of estimated homographies

As mentioned above, the homography estimates obtained with the DLT method do not generally minimize the projection errors in the sensor image. In this step, the estimated homography $\mathbf{H}$ for a *single* view is numerically refined by minimizing the projection error. Minimizing the projection error requires non-linear optimization, which is usually implemented with iterative schemes, such as the Levenberg-Marquart (LM) method, described in Sec. E of the Appendix.

Given is the ordered sequence of model (target) points $\mathcal{X} = (\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1})$, the set of corresponding (observed) sensor points $\dot{\boldsymbol{U}} = (\dot{\boldsymbol{u}}_0, \ldots, \dot{\boldsymbol{u}}_{N-1})$ for a particular view, and the estimated homography $\mathbf{H}$ (calculated as described in Sec. 3.2.1). Following Eqn. (45), the goal is to minimize the projection error

$$E_{\mathrm{proj}} = \sum_{j=0}^{N-1} \left\| \dot{\underline{\boldsymbol{u}}}_j - \mathbf{H} \cdot \underline{\boldsymbol{X}}_j \right\|^2 \tag{69}$$

in the sensor plane, assuming that the positions of the model points $\boldsymbol{X}_j$ are accurate. The Levenberg-Marquart optimization, performed by procedure Optimize() in Alg. 4.3, requires two functions for specifying the optimization problem:

The *value* function: The function $\textsc{Val}(\mathcal{X}, \mathtt{h})$ returns the vector (of length $2N$) of projected (u, v) coordinates, obtained by applying the homography $\mathbf{H}$ (represented by the vector $\mathtt{h} = (\mathtt{h}_0, \ldots, \mathtt{h}_8)$) to all model points $\boldsymbol{X}_j = (X_j, Y_j)^{\mathsf{T}}$ in $\mathcal{X}$, that is,

$$\boldsymbol{u}_j = \begin{pmatrix} u_j \\ v_j \end{pmatrix} = \mathrm{hom}^{-1}(\mathbf{H} \cdot \mathrm{hom}(\boldsymbol{X}_j)). \tag{70}$$

The returned vector stacks the u, v values for all N model points in the form

$$\textsc{Val}(\mathcal{X}, \mathtt{h}) = \mathtt{Y} = \begin{pmatrix} u_0 \\ v_0 \\ u_1 \\ v_1 \\ \vdots \\ u_{N-1} \\ v_{N-1} \end{pmatrix}. \tag{71}$$

The *Jacobian* function: The function $\textsc{Jac}(\mathcal{X}, \mathtt{h})$ returns a stacked Jacobian matrix (of size $2N \times 9$), consisting of the partial derivatives w.r.t the nine homography parameters. The structure of the returned matrix is

$$\textsc{Jac}(\mathcal{X}, \mathtt{h}) = \mathtt{J} = \begin{pmatrix} \frac{\partial u_0}{\partial \mathtt{h}_0} & \frac{\partial u_0}{\partial \mathtt{h}_1} & \frac{\partial u_0}{\partial \mathtt{h}_2} & \frac{\partial u_0}{\partial \mathtt{h}_3} & \frac{\partial u_0}{\partial \mathtt{h}_4} & \frac{\partial u_0}{\partial \mathtt{h}_5} & \frac{\partial u_0}{\partial \mathtt{h}_6} & \frac{\partial u_0}{\partial \mathtt{h}_7} & \frac{\partial u_0}{\partial \mathtt{h}_8} \\ \frac{\partial v_0}{\partial \mathtt{h}_0} & \frac{\partial v_0}{\partial \mathtt{h}_1} & \frac{\partial v_0}{\partial \mathtt{h}_2} & \frac{\partial v_0}{\partial \mathtt{h}_3} & \frac{\partial v_0}{\partial \mathtt{h}_4} & \frac{\partial v_0}{\partial \mathtt{h}_5} & \frac{\partial v_0}{\partial \mathtt{h}_6} & \frac{\partial v_0}{\partial \mathtt{h}_7} & \frac{\partial v_0}{\partial \mathtt{h}_8} \\ \hdashline \frac{\partial u_1}{\partial \mathtt{h}_0} & \frac{\partial u_1}{\partial \mathtt{h}_1} & \frac{\partial u_1}{\partial \mathtt{h}_2} & \frac{\partial u_1}{\partial \mathtt{h}_3} & \frac{\partial u_1}{\partial \mathtt{h}_4} & \frac{\partial u_1}{\partial \mathtt{h}_5} & \frac{\partial u_1}{\partial \mathtt{h}_6} & \frac{\partial u_1}{\partial \mathtt{h}_7} & \frac{\partial u_1}{\partial \mathtt{h}_8} \\ \frac{\partial v_1}{\partial \mathtt{h}_0} & \frac{\partial v_1}{\partial \mathtt{h}_1} & \frac{\partial v_1}{\partial \mathtt{h}_2} & \frac{\partial v_1}{\partial \mathtt{h}_3} & \frac{\partial v_1}{\partial \mathtt{h}_4} & \frac{\partial v_1}{\partial \mathtt{h}_5} & \frac{\partial v_1}{\partial \mathtt{h}_6} & \frac{\partial v_1}{\partial \mathtt{h}_7} & \frac{\partial v_1}{\partial \mathtt{h}_8} \\ \hdashline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hdashline \frac{\partial u_{N-1}}{\partial \mathtt{h}_0} & \frac{\partial u_{N-1}}{\partial \mathtt{h}_1} & \frac{\partial u_{N-1}}{\partial \mathtt{h}_2} & \frac{\partial u_{N-1}}{\partial \mathtt{h}_3} & \frac{\partial u_{N-1}}{\partial \mathtt{h}_4} & \frac{\partial u_{N-1}}{\partial \mathtt{h}_5} & \frac{\partial u_{N-1}}{\partial \mathtt{h}_6} & \frac{\partial u_{N-1}}{\partial \mathtt{h}_7} & \frac{\partial u_{N-1}}{\partial \mathtt{h}_8} \\ \frac{\partial v_{N-1}}{\partial \mathtt{h}_0} & \frac{\partial v_{N-1}}{\partial \mathtt{h}_1} & \frac{\partial v_{N-1}}{\partial \mathtt{h}_2} & \frac{\partial v_{N-1}}{\partial \mathtt{h}_3} & \frac{\partial v_{N-1}}{\partial \mathtt{h}_4} & \frac{\partial v_{N-1}}{\partial \mathtt{h}_5} & \frac{\partial v_{N-1}}{\partial \mathtt{h}_6} & \frac{\partial v_{N-1}}{\partial \mathtt{h}_7} & \frac{\partial v_{N-1}}{\partial \mathtt{h}_8} \end{pmatrix}, \tag{72}$$

where each pair of rows is associated with a particular model point $\boldsymbol{X}_j = (X_j, Y_j)$ and contains the partial derivatives

$$\begin{pmatrix} \mathtt{J}_{2j,*} \\ \mathtt{J}_{2j+1,*} \end{pmatrix} = \begin{pmatrix} \frac{X_j}{w} & \frac{Y_j}{w} & \frac{1}{w} & 0 & 0 & 0 & \frac{-s_x \cdot X_j}{w^2} & \frac{-s_x \cdot Y_j}{w^2} & \frac{-s_x}{w^2} \\ 0 & 0 & 0 & \frac{X_j}{w} & \frac{Y_j}{w} & \frac{1}{w} & \frac{-s_y \cdot X_j}{w^2} & \frac{-s_y \cdot Y_j}{w^2} & \frac{-s_y}{w^2}) \end{pmatrix}, \tag{73}$$

with

$$s_x = \mathtt{h}_0 \cdot X_j + \mathtt{h}_1 \cdot Y_j + \mathtt{h}_2, \tag{74}$$
$$s_y = \mathtt{h}_3 \cdot X_j + \mathtt{h}_4 \cdot Y_j + \mathtt{h}_5, \tag{75}$$
$$w = \mathtt{h}_6 \cdot X_j + \mathtt{h}_7 \cdot Y_j + \mathtt{h}_8. \tag{76}$$

See Alg. 4.3 for a compact summary of this step.

3.3 Step 2: Determining the intrinsic camera parameters

In the previous step, the homographies $\mathbf{H}_0, \ldots, \mathbf{H}_{M-1}$ were calculated *independently* for each of the M views. The homographies encode both the common camera intrinsics as well as the extrinsic transformation parameters that are generally different for each view. This section describes the extraction of the intrinsic camera parameters from the given set of homographies.

As defined in Eqn. (42), a homography $\mathbf{H} = \mathbf{H}_i$ combines the inner camera transformation $\mathbf{A}$ and the view-specific external transformation $\mathbf{R}$, $\boldsymbol{t}$, such that

$$\mathbf{H} = \begin{pmatrix} | & | & | \\ \boldsymbol{h}_0 & \boldsymbol{h}_1 & \boldsymbol{h}_2 \\ | & | & | \end{pmatrix} = \lambda \cdot \mathbf{A} \cdot \begin{pmatrix} | & | & | \\ \boldsymbol{r}_0 & \boldsymbol{r}_1 & \boldsymbol{t} \\ | & | & | \end{pmatrix}, \tag{77}$$

where λ is an arbitrary nonzero scale factor. For $\mathbf{R}$ to be a valid rotation matrix, the column vectors $\boldsymbol{r}_0$, $\boldsymbol{r}_1$ must be *orthonormal*, i. e.,

$$\boldsymbol{r}_0^\mathsf{T} \cdot \boldsymbol{r}_1 = \boldsymbol{r}_1^\mathsf{T} \cdot \boldsymbol{r}_0 = 0 \qquad \text{and} \tag{78}$$
$$\boldsymbol{r}_0^\mathsf{T} \cdot \boldsymbol{r}_0 = \boldsymbol{r}_1^\mathsf{T} \cdot \boldsymbol{r}_1 = 1. \tag{79}$$

We can see from Eqn. (77) that

$$\boldsymbol{h}_0 = \lambda \cdot \mathbf{A} \cdot \boldsymbol{r}_0, \tag{80}$$
$$\boldsymbol{h}_1 = \lambda \cdot \mathbf{A} \cdot \boldsymbol{r}_1 \tag{81}$$

and thus

$$\mathbf{A}^{-1} \cdot \boldsymbol{h}_0 = \lambda \cdot \boldsymbol{r}_0, \tag{82}$$
$$\mathbf{A}^{-1} \cdot \boldsymbol{h}_1 = \lambda \cdot \boldsymbol{r}_1 \tag{83}$$

and furthermore[12]

$$\boldsymbol{h}_0^\mathsf{T} \cdot (\mathbf{A}^{-1})^\mathsf{T} = \lambda \cdot \boldsymbol{r}_0^\mathsf{T}, \tag{84}$$
$$\boldsymbol{h}_1^\mathsf{T} \cdot (\mathbf{A}^{-1})^\mathsf{T} = \lambda \cdot \boldsymbol{r}_1^\mathsf{T}. \tag{85}$$

Based on Eqns. (78)–(79) this yields **two fundamental contraints** on the intrinsic parameters for a given homography $\mathbf{H}$:

$$\boldsymbol{h}_0^\mathsf{T} \cdot (\mathbf{A}^{-1})^\mathsf{T} \cdot \mathbf{A}^{-1} \cdot \boldsymbol{h}_1 = 0, \tag{86}$$
$$\boldsymbol{h}_0^\mathsf{T} \cdot (\mathbf{A}^{-1})^\mathsf{T} \cdot \mathbf{A}^{-1} \cdot \boldsymbol{h}_0 = \boldsymbol{h}_1^\mathsf{T} \cdot (\mathbf{A}^{-1})^\mathsf{T} \cdot \mathbf{A}^{-1} \cdot \boldsymbol{h}_1 \tag{87}$$

(the factor λ is irrelevant here). For estimating the camera intrinsics, Zhang substitutes the above expression $(\mathbf{A}^{-1})^\mathsf{T} \cdot \mathbf{A}^{-1}$ by a new matrix[13]

$$\mathbf{B} = (\mathbf{A}^{-1})^\mathsf{T} \cdot \mathbf{A}^{-1} = \begin{pmatrix} B_0 & B_1 & B_3 \\ B_1 & B_2 & B_4 \\ B_3 & B_4 & B_5 \end{pmatrix}, \tag{88}$$

which is symmetric and composed of only 6 distinct quantities (by inserting from Eqn. (15)):

$$B_0 = \frac{1}{\alpha^2}, \qquad B_1 = -\frac{\gamma}{\alpha^2 \beta}, \tag{89}$$
$$B_2 = \frac{\gamma^2}{\alpha^2 \beta^2} + \frac{1}{\beta^2}, \qquad B_3 = \frac{v_c \gamma - u_c \beta}{\alpha^2 \beta}, \tag{90}$$
$$B_4 = -\frac{\gamma(v_c \gamma - u_c \beta)}{\alpha^2 \beta^2} - \frac{v_c}{\beta^2}, \qquad B_5 = \frac{(v_c \gamma - u_c \beta)^2}{\alpha^2 \beta^2} + \frac{v_c^2}{\beta^2} + 1. \tag{91}$$

We can now write the constraints in Eqns. (86)–(87) in the form

$$\boldsymbol{h}_0^\mathsf{T} \cdot \mathbf{B} \cdot \boldsymbol{h}_1 = 0, \tag{92}$$
$$\boldsymbol{h}_0^\mathsf{T} \cdot \mathbf{B} \cdot \boldsymbol{h}_0 - \boldsymbol{h}_1^\mathsf{T} \cdot \mathbf{B} \cdot \boldsymbol{h}_1 = 0, \tag{93}$$

with $\boldsymbol{h}_p = (H_{p,0}, H_{p,1}, H_{p,2})^\mathsf{T}$ being the p^{th} column vector (for $p \in \{0, 1, 2\}$), of homography $\mathbf{H}$ (see Eqn. (77)). Using the 6-dimensional vector

$$\boldsymbol{b} = (B_0, B_1, B_2, B_3, B_4, B_5)^\mathsf{T}. \tag{94}$$

[12] Since $(A \cdot B)^\mathsf{T} = B^\mathsf{T} \cdot A^\mathsf{T}$.

[13] We use the same (not necessarily intuitive) sequencing of the matrix elements as in [15] for compatibility.

to represent the matrix $\mathbf{B}$ (Eqn. (88)), we get the identity

$$\boldsymbol{h}_p^\mathsf{T} \cdot \mathbf{B} \cdot \boldsymbol{h}_q = \boldsymbol{v}_{p,q}(\mathbf{H}) \cdot \boldsymbol{b}, \tag{95}$$

where $\boldsymbol{v}_{p,q}(\mathbf{H})$ is a 6-dimensional *row* vector obtained from the (estimated) homography $\mathbf{H}$ as

$$\boldsymbol{v}_{p,q}(\mathbf{H}) = \begin{pmatrix} H_{0,p} \cdot H_{0,q} \\ H_{0,p} \cdot H_{1,q} + H_{1,p} \cdot H_{0,q} \\ H_{1,p} \cdot H_{1,q} \\ H_{2,p} \cdot H_{0,q} + H_{0,p} \cdot H_{2,q} \\ H_{2,p} \cdot H_{1,q} + H_{1,p} \cdot H_{2,q} \\ H_{2,p} \cdot H_{2,q} \end{pmatrix}^\mathsf{T}. \tag{96}$$

For a particular homography $\mathbf{H}$, the two constraints in Eqns. (86)–(87) can now be reformulated as a pair of linear equations,

$$\begin{pmatrix} \boldsymbol{v}_{0,1}(\mathbf{H}) \\ \boldsymbol{v}_{0,0}(\mathbf{H}) - \boldsymbol{v}_{1,1}(\mathbf{H}) \end{pmatrix} \cdot \boldsymbol{b} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \tag{97}$$

for the unknown vector $\boldsymbol{b}$ (defined in Eqn. (94)). To consider the estimated homographies $\mathbf{H}_i$ from *all* M views, the associated $2M$ equations are simply stacked in the usual way, i. e.,

$$\begin{pmatrix} \boldsymbol{v}_{0,1}(\mathbf{H}_0) \\ \boldsymbol{v}_{0,0}(\mathbf{H}_0) - \boldsymbol{v}_{1,1}(\mathbf{H}_0) \\ \hdashline \boldsymbol{v}_{0,1}(\mathbf{H}_1) \\ \boldsymbol{v}_{0,0}(\mathbf{H}_1) - \boldsymbol{v}_{1,1}(\mathbf{H}_1) \\ \hdashline \vdots \\ \hdashline \boldsymbol{v}_{0,1}(\mathbf{H}_i) \\ \boldsymbol{v}_{0,0}(\mathbf{H}_i) - \boldsymbol{v}_{1,1}(\mathbf{H}_i) \\ \hdashline \vdots \\ \hdashline \boldsymbol{v}_{0,1}(\mathbf{H}_{M-1}) \\ \boldsymbol{v}_{0,0}(\mathbf{H}_{M-1}) - \boldsymbol{v}_{1,1}(\mathbf{H}_{M-1}) \end{pmatrix} \cdot \boldsymbol{b} = \begin{pmatrix} 0 \\ 0 \\ \hdashline 0 \\ 0 \\ \hdashline \vdots \\ \hdashline 0 \\ 0 \\ \hdashline \vdots \\ \hdashline 0 \\ 0 \end{pmatrix} \qquad \text{or} \qquad \mathbf{V} \cdot \boldsymbol{b} = \mathbf{0}, \tag{98}$$

for short, with the matrix $\mathbf{V}$ of size $2M \times 6$. Again we have an overdetermined system of homogeneous linear equations, which we can readily solved by singular-value decomposition, as described earlier (see Sec. 3.2.1).

Once the vector $\boldsymbol{b} = (B_0, B_1, B_2, B_3, B_4, B_5)^\mathsf{T}$ and hence $\mathbf{B}$ is known, the camera intrinsics (i. e., the matrix $\mathbf{A}$) can be calculated. Note that the matrix $\mathbf{A}$ relates to $\mathbf{B}$ only by the (unknown) scale factor λ, i. e., $\mathbf{B} = \lambda \cdot (\mathbf{A}^{-1})^\mathsf{T} \cdot \mathbf{A}^{-1}$. An elegant (though not trivial) closed-form calculation of $\mathbf{A}$, proposed in [15], is

$$\alpha = \sqrt{w/(d \cdot B_0)}, \tag{99}$$

$$\beta = \sqrt{w/d^2 \cdot B_0}, \tag{100}$$

$$\gamma = \sqrt{w/(d^2 \cdot B_0)} \cdot B_1, \tag{101}$$

$$u_c = (B_1 B_4 - B_2 B_3)/d, \tag{102}$$

$$v_c = (B_1 B_3 - B_0 B_4)/d, \tag{103}$$

with

$$w = B_0 B_2 B_5 - B_1^2 B_5 - B_0 B_4^2 + 2 B_1 B_3 B_4 - B_2 B_3^2, \tag{104}$$

$$d = B_0 B_2 - B_1^2. \tag{105}$$

An alternative formulation for calculating $\mathbf{A}$ – based on numerical decomposition of $\mathbf{B}$ – is given in Sec. C.

3.4 Step 3: Extrinsic view parameters

Once the camera intrinsics are known, the extrinsic parameters $\mathbf{R}, \boldsymbol{t}$ for each view i can be calculated from the corresponding homography $\mathbf{H} = \mathbf{H}_i$. From Eqn. (77) we get

$$\boldsymbol{r}_0 = \lambda \cdot \mathbf{A}^{-1} \cdot \boldsymbol{h}_0, \qquad \boldsymbol{r}_1 = \lambda \cdot \mathbf{A}^{-1} \cdot \boldsymbol{h}_1, \qquad \boldsymbol{t} = \lambda \cdot \mathbf{A}^{-1} \cdot \boldsymbol{h}_2, \tag{106}$$

with the scale factor

$$\lambda = \frac{1}{\|\mathbf{A}^{-1} \cdot \boldsymbol{h}_0\|} = \frac{1}{\|\mathbf{A}^{-1} \cdot \boldsymbol{h}_1\|}, \tag{107}$$

and finally (since $\mathbf{R}$ must be orthonormal)

$$\boldsymbol{r}_2 = \boldsymbol{r}_0 \times \boldsymbol{r}_1. \tag{108}$$

Note that $\boldsymbol{h}$, $\boldsymbol{r}$, $\boldsymbol{t}$, and λ are different for each view i. The resulting 3×3 matrix $\mathbf{R} = (\boldsymbol{r}_0 \mid \boldsymbol{r}_1 \mid \boldsymbol{r}_2)$ is most likely *not* a proper rotation matrix. However, there are proven techniques for calculating the most similar "true" rotation matrix for a given 3×3 matrix (again based on singular-value decomposition, as described e. g. in [10, Sec. 2.6.5] and [14, Appendix C]).[14]

3.5 Step 4: Estimating radial lens distortion

All calculations so far assumed that the inner camera transformation follows the simple pinhole projection model. In particular, the distortions introduced by real lens systems were ignored so far. In this step, a simple non-linear lens distortion model is added to the projection pipeline and its parameters are calculated from the observed images. This is accomplished in two steps: First, the distortion parameters are estimated by linear least-squares fitting, minimizing the projection error. The lens distortion parameters are then refined (simultaneously with all other parameters) in a final, overall optimization step, described in Sec. 3.6.

At this point the (linear) camera intrinsics ($\mathbf{A}$) are *approximately* known, and the assumption is that *all* remaining projection errors can be attributed to lens distortion. Thus any inaccuracies in the previous steps will also affect the distortion estimates and the results obtained in this step may be far off the real values. Alternatively, one could omit this step altogether and rely on the overall refinement step (in Sec. 3.6) to calculate accurate distortion parameters (assuming zero distortion at the start).

[14] The `Rotation` class of the *Apache Commons Math* library provides a construction (among others) that builds a proper rotation from any 3×3 matrix that is sufficiently conditioned.

As mentioned, all remaining errors, i. e., the deviations between the *projected sensor points* $\boldsymbol{u}_{i,j}$ and the actually *observed sensor points* $\dot{\boldsymbol{u}}_{i,j}$,

$$\dot{\boldsymbol{d}}_{i,j} = \dot{\boldsymbol{u}}_{i,j} - \boldsymbol{u}_{i,j}, \tag{109}$$

are now attributed to lens distortion. Consequently, $\dot{\boldsymbol{d}}_{i,j}$ is referred to as the *observed distortion vector*.

As described in Sec. 2.5.2, lens distortion is modeled as a *radial displacement*, that is, the original (undistorted) projection $\boldsymbol{u}_{i,j}$ is warped to the distorted point $\tilde{\boldsymbol{u}}_{i,j}$ by

$$\begin{aligned}
\tilde{\boldsymbol{u}}_{i,j} &= \boldsymbol{u}_c + (\boldsymbol{u}_{i,j} - \boldsymbol{u}_c) \cdot [1 + D(r_{i,j}, \boldsymbol{k})] && (110)\\
&= \boldsymbol{u}_c + \boldsymbol{u}_{i,j} - \boldsymbol{u}_c + (\boldsymbol{u}_{i,j} - \boldsymbol{u}_c) \cdot D(r_{i,j}, \boldsymbol{k}) && (111)\\
&= \boldsymbol{u}_{i,j} + \underbrace{(\boldsymbol{u}_{i,j} - \boldsymbol{u}_c) \cdot D(r_{i,j}, \boldsymbol{k})}_{\boldsymbol{d}_{i,j}} = \boldsymbol{u}_{i,j} + \boldsymbol{d}_{i,j}. && (112)
\end{aligned}$$

The resulting *model distortion vector*,

$$\boldsymbol{d}_{i,j} = (\boldsymbol{u}_{i,j} - \boldsymbol{u}_c) \cdot D(r_{i,j}, \boldsymbol{k}), \tag{113}$$

is based on the estimated projection center $\boldsymbol{u}_c$ (in sensor coordinates) and the non-linear radial distortion function $D(r, \boldsymbol{k})$, as defined in Eqn. (32). The parameters $\boldsymbol{k} = (k_0, k_1)$ are to be estimated (see Fig. 1). Note that, in Eqn. (113), the *radius* $r_{i,j}$ passed to the function $D()$ is *not* calculated from the projected sensor points $\boldsymbol{u}_{i,j}$ but as the distance of the associated points $\boldsymbol{x}_{i,j}$ from the projection center (principal point) $\boldsymbol{x}_c = (0, 0)^\mathsf{T}$ in the "normalized" projection plane, that is,

$$r_{i,j} = \|\boldsymbol{x}_{i,j} - \boldsymbol{x}_c\| = \|\boldsymbol{x}_{i,j}\|. \tag{114}$$

For a positive function value $D(r_{i,j}, \boldsymbol{k})$ the sensor point $\boldsymbol{u}_{i,j}$ is shifted *outwards* (i. e., away from the projection center) to the distorted position $\tilde{\boldsymbol{u}}_{i,j}$, and *inwards* if the function value is *negative*.

The unknown distortion parameters $\boldsymbol{k}$ can be estimated by *minimizing the difference* between the *model* distortions $\boldsymbol{d}_{i,j}$ (Eqn. (113)) and the associated *observed* distortions $\dot{\boldsymbol{d}}_{i,j}$ (Eqn. (109)), that is,

$$\sum_{i,j} \|\boldsymbol{d}_{i,j} - \dot{\boldsymbol{d}}_{i,j}\| \rightarrow \min. \tag{115}$$

In other words, we are looking for a *least squares solution* to the over-determined system of equations $\boldsymbol{d}_{i,j} = \dot{\boldsymbol{d}}_{i,j}$ (for all point index pairs i, j), that is,

$$(\boldsymbol{u}_{i,j} - \boldsymbol{u}_c) \cdot D(r_{i,j}, \boldsymbol{k}) = \dot{\boldsymbol{u}}_{i,j} - \boldsymbol{u}_{i,j}, \tag{116}$$

to find the distortion parameters $\boldsymbol{k}$. By inserting from Eqns. (109)–(113) and expanding the function $D()$ from Eqn. (32), every observed point i, j contributes a pair of equations

$$\begin{aligned}
(\dot{u}_{i,j} - u_c) \cdot r_{i,j}^2 \cdot k_0 + (\dot{u}_{i,j} - u_c) \cdot r_{i,j}^4 \cdot k_1 &= (\dot{u}_{i,j} - u_{i,j}),\\
(\dot{v}_{i,j} - v_c) \cdot r_{i,j}^2 \cdot k_0 + (\dot{v}_{i,j} - v_c) \cdot r_{i,j}^4 \cdot k_1 &= (\dot{v}_{i,j} - v_{i,j}),
\end{aligned} \tag{117}$$

to the system. Note that (fortunately) these equations are *linear* in the unknowns k_0, k_1, i. e., they can be solved with standard linear algebra methods.[15] For this purpose, we rewrite Eqn. (117) in the familiar matrix notation as

$$\begin{pmatrix} (\dot{u}_{i,j} - u_c) \cdot r_{i,j}^2 & (\dot{u}_{i,j} - u_c) \cdot r_{i,j}^4 \\ (\dot{v}_{i,j} - v_c) \cdot r_{i,j}^2 & (\dot{v}_{i,j} - v_c) \cdot r_{i,j}^4 \end{pmatrix} \cdot \begin{pmatrix} k_0 \\ k_1 \end{pmatrix} = \begin{pmatrix} \dot{u}_{i,j} - u_{i,j} \\ \dot{v}_{i,j} - v_{i,j} \end{pmatrix}. \tag{118}$$

By stacking the equations for all MN points on top of each other, we end up with a system of $2MN$ linear equations,

$$\underbrace{\begin{pmatrix} (\dot{u}_{0,0}-u_c) \cdot r_{0,0}^2 & (\dot{u}_{0,0}-u_c) \cdot r_{0,0}^4 \\ (\dot{v}_{0,0}-v_c) \cdot r_{0,0}^2 & (\dot{v}_{0,0}-v_c) \cdot r_{0,0}^4 \\ (\dot{u}_{0,1}-u_c) \cdot r_{0,1}^2 & (\dot{u}_{0,1}-u_c) \cdot r_{0,1}^4 \\ (\dot{v}_{0,1}-v_c) \cdot r_{0,1}^2 & (\dot{v}_{0,1}-v_c) \cdot r_{0,1}^4 \\ \vdots & \vdots \\ (\dot{u}_{i,j}-u_c) \cdot r_{i,j}^2 & (\dot{u}_{i,j}-u_c) \cdot r_{i,j}^4 \\ (\dot{v}_{i,j}-v_c) \cdot r_{i,j}^2 & (\dot{v}_{i,j}-v_c) \cdot r_{i,j}^4 \\ \vdots & \vdots \\ (\dot{u}_{M-1,N-1}-u_c) \cdot r_{M-1,N-1}^2 & (\dot{u}_{M-1,N-1}-u_c) \cdot r_{M-1,N-1}^4 \\ (\dot{v}_{M-1,N-1}-v_c) \cdot r_{M-1,N-1}^2 & (\dot{v}_{M-1,N-1}-v_c) \cdot r_{M-1,N-1}^4 \end{pmatrix}}_{\mathbf{D}} \cdot \begin{pmatrix} k_0 \\ k_1 \end{pmatrix} = \underbrace{\begin{pmatrix} \dot{u}_{0,0}-u_{0,0} \\ \dot{v}_{0,0}-v_{0,0} \\ \dot{u}_{0,1}-u_{0,1} \\ \dot{v}_{0,1}-v_{0,1} \\ \vdots \\ \dot{u}_{i,j}-u_{i,j} \\ \dot{v}_{i,j}-v_{i,j} \\ \vdots \\ \dot{u}_{M-1,N-1}-u_{M-1,N-1} \\ \dot{v}_{M-1,N-1}-v_{M-1,N-1} \end{pmatrix}}_{\dot{\boldsymbol{d}}}, \tag{119}$$

or $\mathbf{D} \cdot \boldsymbol{k} = \dot{\boldsymbol{d}}$ for short, with $\boldsymbol{k} = (k_0, k_1)^\mathsf{T}$ as the vector of unknowns. The least-squares solution that minimizes $\|\mathbf{D} \cdot \boldsymbol{k} - \dot{\boldsymbol{d}}\|^2$ is found with the usual numerical methods (e. g., singular-value or QR-decomposition).[16]

3.6 Step 5: Refining all parameters

The last step of the calibration procedure is to optimize all calibration parameters, i. e., the camera intrinsics and the extrinsic parameters for all M views (with N observed points each), in a single (non-linear) system of equations. We define the vectors

$$\boldsymbol{a} = (\underbrace{\alpha, \beta, \gamma, u_c, v_c}_{\text{from } \mathbf{A}}, \underbrace{k_0, k_1}_{\boldsymbol{k}})^\mathsf{T}, \tag{120}$$

to collect the intrinsic camera parameters ($\boldsymbol{a}$, with 5 elements taken from the estimated matrix $\mathbf{A}$ and 2 elements from $\boldsymbol{k}$) and the extrinsic parameters

$$\boldsymbol{w}_i = (\underbrace{\rho_{i,x}, \rho_{i,y}, \rho_{i,z}}_{\text{from } \mathbf{R}_i}, \underbrace{t_{i,x}, t_{i,y}, t_{i,z}}_{\text{from } \boldsymbol{t}_i})^\mathsf{T}, \tag{121}$$

for $i = 0, \ldots, M-1$, ($\boldsymbol{w}_i$, with 3 elements taken from $\mathbf{R}_i$ and 3 elements from $\boldsymbol{t}_i$) for each view i (see below for the meaning of the rotation parameters ρ_i).

[15] Of course, the same approach can be used for higher-order distortion models with additional coefficients.

[16] The Java implementation described in Sec. 6 uses an instance of the *Apache Commons Math* `QRDecomposition` class and its associated solver.

3.6.1 Total projection error

Given the observed image points $\dot{\boldsymbol{u}}_{i,j} = (\dot{u}_{i,j}, \dot{v}_{i,j})^\mathsf{T}$, the goal is to minimize the *total projection error*

$$E(\boldsymbol{a}, \boldsymbol{w}) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \|\dot{\boldsymbol{u}}_{i,j} - \boldsymbol{u}_{i,j}\| \tag{122}$$

$$= \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \|\dot{\boldsymbol{u}}_{i,j} - P(\boldsymbol{a}, \boldsymbol{w}_i, \boldsymbol{X}_j)\|^2 \tag{123}$$

$$= \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |\dot{u}_{i,j} - u_{i,j}|^2 + |\dot{v}_{i,j} - v_{i,j}|^2 \tag{124}$$

$$= \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |\dot{u}_{i,j} - P_\mathrm{x}(\boldsymbol{a}, \boldsymbol{w}_i, \boldsymbol{X}_j)|^2 + |\dot{v}_{i,j} - P_\mathrm{y}(\boldsymbol{a}, \boldsymbol{w}_i, \boldsymbol{X}_j)|^2, \tag{125}$$

for the camera parameters $\boldsymbol{a}$ and all view parameters $\boldsymbol{w} = \boldsymbol{w}_0, \ldots, \boldsymbol{w}_{M-1}$.[17] This is a non-linear least-squares minimization problem which cannot be solved in closed form or by linear least-squares fitting. Iterative techniques are used in this case, as described in Sec. 3.6.3.

3.6.2 Parameterizing the extrinsic rotation matrices $\mathbf{R}_i$

Every 3D rotation matrix $\mathbf{R}$ consists of nine elements, despite the fact that it has only three degrees of freedom. Thus rotation matrices are subjected to strong constraints (see Sec. A.2.1 in the Appendix).

There are several ways to express arbitrary 3D rotations with only 3 parameters. As suggested in [15], we use the Euler-Rodrigues method [13, Ch. 6] [5, p. 585] that is based on a 3D vector

$$\boldsymbol{\rho} = (\rho_x, \rho_y, \rho_z) \tag{126}$$

that specifies both the 3D axis of rotation and the rotation angle θ as its magnitude, that is,

$$\theta = \|\boldsymbol{\rho}\| \qquad \text{and} \qquad \boldsymbol{\rho} = \frac{\boldsymbol{\rho}}{\|\boldsymbol{\rho}\|} \cdot \theta = \hat{\boldsymbol{\rho}} \cdot \theta, \tag{127}$$

where $\hat{\boldsymbol{\rho}}$ is the normalized (unit) vector for $\boldsymbol{\rho}$. A rotation matrix $\mathbf{R}$ can be easily converted to a Rodrigues vector $\boldsymbol{\rho}$ and vice versa; details are found in the literature.[18] In the algorithms described below, we use the notation

$$\mathbf{R} \leftarrow \textsc{ToRotationMatrix}(\boldsymbol{\rho}),$$
$$\boldsymbol{\rho} \leftarrow \textsc{ToRodriguesVector}(\mathbf{R}).$$

for converting between Rodrigues vectors and rotations matrices (see Sec. A.2.4 in the Appendix).

[17] With 5 intrinsic camera parameters, 2 lens distortion coefficients, and 6 parameters for each of the M views, this means optimizing $7 + M \cdot 6$ parameters.

[18] The `Rotation` class of *Apache Commons Math* provides convenient constructors to create a unique rotation from either a 3×3 rotation matrix or a direction vector and rotation angle.

3.6.3 Non-linear optimization

As outlined in Sec. 3.6.1, the goal of the overall refinement step is to determine the camera parameters $\boldsymbol{a}$ and the view parameters $\boldsymbol{w}_0, \ldots, \boldsymbol{w}_{M-1}$ that minimize the total projection error E (see Eqns. (122)–(125)). Non-linear optimization problems of this kind can only be solved with iterative techniques, such as the *Levenberg-Marquart* (LM) method [10, Sec. 15.5.2], which combines the Gauss-Newton method and the steepest gradient descent method. For a short introduction to the general approach see Sec. E in the Appendix.

To apply the LM-technique to the calibration refinement problem, we first concatenate the estimated intrinsic parameters $\boldsymbol{a}$ and all extrinsic parameters $\boldsymbol{w}_i$ (defined in defined in Eqns. (120–121) into a composite parameter vector

$$\begin{aligned} \mathtt{P} &= (\boldsymbol{a}^\mathsf{T} \mid \boldsymbol{w}_0^\mathsf{T} \mid \ldots \mid \boldsymbol{w}_{M-1}^\mathsf{T})^\mathsf{T} && (128) \\ &= (\underbrace{\alpha, \beta, \gamma, u_c, v_c, k_0, k_1}_{\boldsymbol{a}^\mathsf{T}}, \underbrace{\rho_{0,x}, \rho_{0,y}, \rho_{0,z}, t_{0,x}, t_{0,y}, t_{0,z}}_{\boldsymbol{w}_0^\mathsf{T}}, \ldots, \underbrace{\rho_{M-1,x}, \ldots, t_{M-1,z}}_{\boldsymbol{w}_{M-1}^\mathsf{T}})^\mathsf{T}, && (129) \end{aligned}$$

with $7 + 6M$ elements total (see Alg. 4.8).

The *sample positions* (denoted $\boldsymbol{x}_i$ in Sec. E.1, Eqn. (193)) are the 3D coordinates $\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1}$ of the points on the calibration target, doubled (one each for the x and y part of the projection) and repeated for each of the M views, that is,

$$\mathtt{X} = (\underbrace{\boldsymbol{X}_0, \boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1}, \boldsymbol{X}_{N-1}}_{\text{view } 0}, \ldots, \underbrace{\boldsymbol{X}_0, \boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1}, \boldsymbol{X}_{N-1}}_{\text{view } M-1})^\mathsf{T}, \tag{130}$$

The *sample values* (denoted y_i in Sec. E.1, Eqn. (194)) are the x/y-components of the observed image positions $\dot{\boldsymbol{u}}_{i,j} = (\dot{u}_{i,j}, \dot{v}_{i,j})$, that is,

$$\dot{\mathtt{Y}} = (\underbrace{\dot{u}_{0,0}, \dot{v}_{0,0}, \dot{u}_{0,1}, \ldots, \dot{u}_{0,N-1}, \dot{v}_{0,N-1}}_{\text{view } 0}, \ldots, \underbrace{\dot{u}_{M-1,0}, \dot{v}_{M-1,0}, \ldots, \dot{v}_{M-1,N-1}}_{\text{view } M-1})^\mathsf{T}. \tag{131}$$

Both $\mathtt{X}$ and $\dot{\mathtt{Y}}$ are of length $2MN$. Analogous to Eqn. (196), the model evaluation thus has the structure

$$\textsc{Val}(\mathcal{X}, \mathtt{P}) = \mathtt{Y} = \begin{pmatrix} u_{0,0} \\ v_{0,0} \\ \hdashline u_{0,1} \\ v_{0,1} \\ \hdashline \vdots \\ \hdashline u_{i,j} \\ v_{i,j} \\ \hdashline \vdots \\ \hdashline u_{M-1,N-1} \\ v_{M-1,N-1} \end{pmatrix} = \begin{pmatrix} P_\mathrm{x}(\boldsymbol{a}, \boldsymbol{w}_0, \boldsymbol{X}_0) \\ P_\mathrm{y}(\boldsymbol{a}, \boldsymbol{w}_0, \boldsymbol{X}_0) \\ \hdashline P_\mathrm{x}(\boldsymbol{a}, \boldsymbol{w}_0, \boldsymbol{X}_1) \\ P_\mathrm{y}(\boldsymbol{a}, \boldsymbol{w}_0, \boldsymbol{X}_1) \\ \hdashline \vdots \\ \hdashline P_\mathrm{x}(\boldsymbol{a}, \boldsymbol{w}_i, \boldsymbol{X}_j) \\ P_\mathrm{y}(\boldsymbol{a}, \boldsymbol{w}_i, \boldsymbol{X}_j) \\ \hdashline \vdots \\ \hdashline P_\mathrm{x}(\boldsymbol{a}, \boldsymbol{w}_{M-1}, \boldsymbol{X}_{N-1}) \\ P_\mathrm{y}(\boldsymbol{a}, \boldsymbol{w}_{M-1}, \boldsymbol{X}_{N-1}) \end{pmatrix}, \tag{132}$$

again with $2MN$ rows (see Eqn. (129) for the parameter vector $\mathtt{P}$ and Eqn. (24) for the definition of the projection functions $P_\mathrm{x}()$ and $P_\mathrm{y}()$). The corresponding

Jacobian function (see Eqn. (198)) has the form

$$\textsc{Jac}(\mathcal{X}, \mathtt{P}) = \mathtt{J} = \begin{pmatrix} \frac{\partial P_{\mathrm{x}}(\boldsymbol{a},\boldsymbol{w}_0,\boldsymbol{X}_0)}{\partial \mathtt{p}_0} & \frac{\partial P_{\mathrm{x}}(\boldsymbol{a},\boldsymbol{w}_0,\boldsymbol{X}_0)}{\partial \mathtt{p}_1} & \cdots & \frac{\partial P_{\mathrm{x}}(\boldsymbol{a},\boldsymbol{w}_0,\boldsymbol{X}_0)}{\partial \mathtt{p}_{K-1}} \\ \frac{\partial P_{\mathrm{y}}(\boldsymbol{a},\boldsymbol{w}_0,\boldsymbol{X}_0)}{\partial \mathtt{p}_0} & \frac{\partial P_{\mathrm{y}}(\boldsymbol{a},\boldsymbol{w}_0,\boldsymbol{X}_0)}{\partial \mathtt{p}_1} & \cdots & \frac{\partial P_{\mathrm{y}}(\boldsymbol{a},\boldsymbol{w}_0,\boldsymbol{X}_0)}{\partial \mathtt{p}_{K-1}} \\ \frac{\partial P_{\mathrm{x}}(\boldsymbol{a},\boldsymbol{w}_0,\boldsymbol{X}_1)}{\partial \mathtt{p}_0} & \frac{\partial P_{\mathrm{x}}(\boldsymbol{a},\boldsymbol{w}_0,\boldsymbol{X}_1)}{\partial \mathtt{p}_1} & \cdots & \frac{\partial P_{\mathrm{x}}(\boldsymbol{a},\boldsymbol{w}_0,\boldsymbol{X}_1)}{\partial \mathtt{p}_{K-1}} \\ \frac{\partial P_{\mathrm{y}}(\boldsymbol{a},\boldsymbol{w}_0,\boldsymbol{X}_1)}{\partial \mathtt{p}_0} & \frac{\partial P_{\mathrm{y}}(\boldsymbol{a},\boldsymbol{w}_0,\boldsymbol{X}_1)}{\partial \mathtt{p}_1} & \cdots & \frac{\partial P_{\mathrm{y}}(\boldsymbol{a},\boldsymbol{w}_0,\boldsymbol{X}_1)}{\partial \mathtt{p}_{K-1}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial P_{\mathrm{x}}(\boldsymbol{a},\boldsymbol{w}_i,\boldsymbol{X}_j)}{\partial \mathtt{p}_0} & \frac{\partial P_{\mathrm{x}}(\boldsymbol{a},\boldsymbol{w}_i,\boldsymbol{X}_j)}{\partial \mathtt{p}_1} & \cdots & \frac{\partial P_{\mathrm{x}}(\boldsymbol{a},\boldsymbol{w}_i,\boldsymbol{X}_j)}{\partial \mathtt{p}_{K-1}} \\ \frac{\partial P_{\mathrm{y}}(\boldsymbol{a},\boldsymbol{w}_i,\boldsymbol{X}_j)}{\partial \mathtt{p}_0} & \frac{\partial P_{\mathrm{y}}(\boldsymbol{a},\boldsymbol{w}_i,\boldsymbol{X}_j)}{\partial \mathtt{p}_1} & \cdots & \frac{\partial P_{\mathrm{y}}(\boldsymbol{a},\boldsymbol{w}_i,\boldsymbol{X}_j)}{\partial \mathtt{p}_{K-1}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial P_{\mathrm{x}}(\boldsymbol{a},\boldsymbol{w}_{M-1},\boldsymbol{X}_{N-1})}{\partial \mathtt{p}_{M-1,0}} & \frac{\partial P_{\mathrm{x}}(\boldsymbol{a},\boldsymbol{w}_{M-1},\boldsymbol{X}_{N-1})}{\partial \mathtt{p}_{M-1,1}} & \cdots & \frac{\partial P_{\mathrm{x}}(\boldsymbol{a},\boldsymbol{w}_{M-1},\boldsymbol{X}_{N-1})}{\partial \mathtt{p}_{M-1,K-1}} \\ \frac{\partial P_{\mathrm{y}}(\boldsymbol{a},\boldsymbol{w}_{M-1},\boldsymbol{X}_{N-1})}{\partial \mathtt{p}_{M-1,0}} & \frac{\partial P_{\mathrm{y}}(\boldsymbol{a},\boldsymbol{w}_{M-1},\boldsymbol{X}_{N-1})}{\partial \mathtt{p}_{M-1,1}} & \cdots & \frac{\partial P_{\mathrm{y}}(\boldsymbol{a},\boldsymbol{w}_{M-1},\boldsymbol{X}_{N-1})}{\partial \mathtt{p}_{M-1,K-1}} \end{pmatrix}. \tag{133}$$

The "stacked Jacobian" J in Eqn. (133) has two rows for each projected point i, j and one column for each of parameter $\mathtt{p}_k$, i. e., $2MN$ rows and $K = 7 + 6M$ columns. For example, with $M = 5$ views and $N = 256$ points each, J is of size 2560×37.

However, J is *sparse*, since the full parameter vector P is composed of the camera intrinsics $\boldsymbol{a}$ and the extrinsic parameters $\boldsymbol{w}_i$ for the M views (see Eqns. (128)–(129)). The camera parameters $\boldsymbol{a}$ affect all observations, thus the first 7 *columns* of J generally contain non-zero values. The first $2N$ rows of J correspond to view $i = 0$ and, in general, view i relates to rows $i \cdot 2N, \ldots, (i{+}1) \cdot 2N - 1$ of the matrix, for $i = 0, \ldots, M - 1$. However, the extrinsic parameters $\boldsymbol{w}_i$ of any view i are relevant for the observations made with that view only, thus the matrix has the block structure shown in Fig. 4.

Each block of columns in J corresponds to a particular segment of the parameter vector $\boldsymbol{p}$. For example, the leftmost (green) column represents the first 7 columns of J for the *intrinsic* (view-independent) camera parameters $\boldsymbol{a}$. All rows in the $\boldsymbol{a}$-segment of the Jacobian must be calculated, since the projections of all views change when any intrinsic camera parameter is modified. Since each view covers $2N$ rows of J, the size of each green block is $2N \times 7$.

However, in the matrix columns associated with the *extrinsic* parameters $\boldsymbol{w}_i$, none of the views – except view i itself – is affected by the values of $\boldsymbol{w}_i$. Thus in the columns for $\boldsymbol{w}_i$ only the matrix rows for view i must be calculated, all other derivatives are unaffected and thus zero. Since $\boldsymbol{w}_i$ extends over 6 matrix columns, the size of each diagonal (red) block is $2N \times 6$. This leaves many zero elements (gray blocks) and thus only a small part of the Jacobian matrix must actually be calculated. This is quite important, because the Jacobian must be recalculated in every iteration of the LM optimization process.

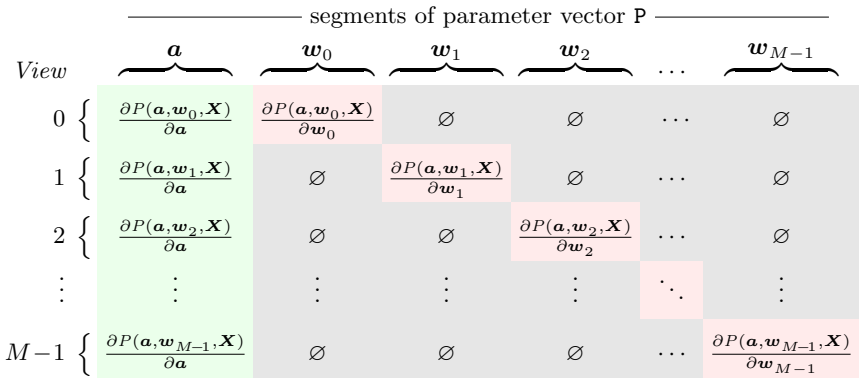


Figure 4: Structure of the "stacked" Jacobian J in Eqn. (133). The *green* blocks only depend on the intrinsic camera parameters $\boldsymbol{a}$. Each of the *pink* blocks depends only on the associated view parameters $\boldsymbol{w}_i$, and the values in the *gray* blocks are zero.

3.6.4 Calculating the Jacobian

The Jacobian matrix in Eqn. (133) consists of the partial derivatives of the projection function with respect to the individual parameters, evaluated for the given position $\boldsymbol{X}$. In the ideal case, the partial derivative functions are known in analytic form and can be directly evaluated.

Analytic calculation. The partial derivative functions of the projection mapping (including the non-linear radial lens distortion) can be obtained in analytic form, e. g., with Matlab, although the resulting expressions are quite involved. The Matlab-generated C code used in most implementations is not human-readable (many anonymous variables) and covers several pages. Although this is the most efficient way to calculate the Jacobian (running about twice as fast as the numeric method described below), any change in the projection model also affects the formulation of the derivatives. Thus the projection model is not implemented in a single place (i. e., by a single function or method) but needs to be replicated for the derivative calculation, which is also a potential source for errors.

Numeric calculation. It is also possible to estimate the partial derivatives numerically by finite difference approximation, as described in Sec. E.1.4 in the Appendix. Of course, only the elements of the Jacobian within the non-zero blocks (see Fig. 4) need to be calculated.

The numerical calculation of the Jacobian takes about 50% longer to execute than the analytical approach described above, even by exploiting the diagonal block structure of the matrix. However, it does not require any specific code for the projection model but simply invokes the same projection method that is used for calculating the "value" function $F()$. This is certainly of advantage, since the calculation of the Jacobian needs not be updated in the case the projection model changes.

In the associated Java implementation, the `LevenbergMarquardtOptimizer` class (provided by the *Apache Commons Math* library) is used to perform the numerical optimization. Details can be found in the source code (see class `NonlinearOptimizer` in package `imagingbook.extras.calibration.zhang`).

4 Summary

The complete calibration process is summarized in Algorithms 4.1–4.8.

Algorithm 4.1 Camera calibration algorithm by Zhang (overview). It is assumed that each observed image point $\dot{\boldsymbol{u}}_{i,j}$ corresponds to the associated model point $\boldsymbol{X}_j$ and all views were taken with the same camera. Details are found in Algs. 4.2–4.8.

1: CALIBRATE($\mathcal{X}, \dot{\mathcal{U}}$)
Input: $\mathcal{X} = (\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1})$, an ordered sequence of 3D points on the planar target, with $\boldsymbol{X}_j = (X_j, Y_j, 0)^\mathsf{T}$; $\dot{\mathcal{U}} = (\dot{\boldsymbol{U}}_0, \ldots, \dot{\boldsymbol{U}}_{M-1})$, a sequence of views, each view $\dot{\boldsymbol{U}}_i = (\dot{\boldsymbol{u}}_{i,0}, \ldots, \dot{\boldsymbol{u}}_{i,N-1})$ is an ordered sequence of observed image points $\dot{\boldsymbol{u}}_{i,j} = (\dot{u}_{i,j}, \dot{v}_{i,j})^\mathsf{T}$. Returns the estimated intrinsic parameters $\mathbf{A}, \boldsymbol{k}$ of the camera and the extrinsic transformations $\mathcal{W} = (\mathbf{W}_0, \ldots, \mathbf{W}_{N-1})$, with $\mathbf{W}_i = (\mathbf{R}_i \mid \boldsymbol{t}_i)$, for each view.

2: $\mathcal{H}_{\text{init}} \leftarrow$ GETHOMOGRAPHIES($\mathcal{X}, \dot{\mathcal{U}}$) ▷ Step 1 (Sec. 3.2)
3: $\mathbf{A}_{\text{init}} \leftarrow$ GETCAMERAINTRINSICS($\mathcal{H}_{\text{init}}$) ▷ Step 2 (Sec. 3.3)
4: $\mathcal{W}_{\text{init}} \leftarrow$ GETEXTRINSICS($\mathbf{A}, \mathcal{H}_{\text{init}}$) ▷ Step 3 (Sec. 3.4)
5: $\boldsymbol{k}_{\text{init}} \leftarrow$ ESTLENSDISTORTION($\mathbf{A}_{\text{init}}, \mathcal{W}_{\text{init}}, \mathcal{X}, \dot{\mathcal{U}}$) ▷ Step 4 (Sec. 3.5)
6: $\langle \mathbf{A}, \boldsymbol{k}, \mathcal{W} \rangle \leftarrow$ REFINEALL($\mathbf{A}_{\text{init}}, \boldsymbol{k}_{\text{init}}, \mathcal{W}_{\text{init}}, \mathcal{X}, \dot{\mathcal{U}}$) ▷ Step 5 (Sec. 3.6)
7: **return** $\langle \mathbf{A}, \boldsymbol{k}, \mathcal{W} \rangle$

Algorithm 4.2 Estimation of view homographies.

GetHomographies($\mathcal{X}$,$\dot{\mathcal{U}}$)
Input: $\mathcal{X} = (\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1})$, the model points; $\dot{\mathcal{U}} = (\dot{\boldsymbol{U}}_0, \ldots, \dot{\boldsymbol{U}}_{M-1})$, the associated sensor points in M views. Returns a sequence of estimated homographies $\mathcal{H} = (\mathbf{H}_0, \ldots, \mathbf{H}_{M-1})$, one for each of the M views.

$M \leftarrow |\dot{\mathcal{U}}|$ ▷ number of views
$\mathcal{H} \leftarrow ( )$
for $i \leftarrow 0, \ldots, M-1$ **do** ▷ for each view i
  $\mathbf{H}_{\text{init}} \leftarrow$ EstimateHomography$(\mathcal{X}, \dot{\boldsymbol{U}}_i)$ ▷ see below
  $\mathbf{H} \leftarrow$ RefineHomography$(\mathbf{H}_{\text{init}}, \mathcal{X}, \dot{\boldsymbol{U}}_i)$ ▷ see Alg. 4.3
  $\mathcal{H} \leftarrow \mathcal{H} \frown (\mathbf{H})$
return $\mathcal{H}$ ▷ $\mathcal{H} = (\mathbf{H}_0, \ldots, \mathbf{H}_{M-1})$

EstimateHomography($\boldsymbol{P}, \boldsymbol{Q}$) ▷ replace A, B
Input: $\boldsymbol{P} = (\boldsymbol{p}_0, \ldots, \boldsymbol{p}_{N-1})$, $\boldsymbol{Q} = (\boldsymbol{q}_0, \ldots, \boldsymbol{q}_{N-1})$, with $\boldsymbol{p}_j, \boldsymbol{q}_j \in \mathbb{R}^2$.
Returns the estimated homography matrix $\mathbf{H}$, such that $\boldsymbol{q}_j = \mathbf{H} \cdot \boldsymbol{p}_j$.

$N \leftarrow |\boldsymbol{P}|$ ▷ number of points in $\boldsymbol{P}, \boldsymbol{Q}$
$\mathbf{N}_P \leftarrow$ GetNormalisationMatrix$(\boldsymbol{P})$
$\mathbf{N}_Q \leftarrow$ GetNormalisationMatrix$(\boldsymbol{Q})$
$\mathbf{M} \leftarrow$ new matrix of size $2N \times 9$
for $j \leftarrow 0, \ldots, N-1$ **do**
  $k \leftarrow 2 \cdot j$
  Normalize:
  $\boldsymbol{p}' \leftarrow \hom^{-1}(\mathbf{N}_P \cdot \hom(\boldsymbol{p}_j))$ ▷ $\boldsymbol{p}' = (x'_p, y'_p)^\intercal$
  $\boldsymbol{q}' \leftarrow \hom^{-1}(\mathbf{N}_Q \cdot \hom(\boldsymbol{q}_j))$ ▷ $\boldsymbol{q}' = (x'_q, y'_q)^\intercal$
  $\mathbf{M}_{k,*} \leftarrow (x'_p, y'_p, 1, 0, 0, 0, -x'_p x'_q, -y'_p x'_q - x'_q)$ ▷ row vec. k
  $\mathbf{M}_{k+1,*} \leftarrow (0, 0, 0, x'_p, y'_p, 1, -x'_p y'_q, -y'_p y'_q, -y'_q)$ ▷ row vec. $k+1$
Solve the homogeneous system (e. g. by singular value decomposition):
$\boldsymbol{h} \leftarrow \text{Solve}(\mathbf{M} \cdot \boldsymbol{h} = \mathbf{0})$ ▷ $\boldsymbol{h} = (h_0, \ldots, h_8)$
$\mathbf{H}' \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 \\ h_3 & h_4 & h_5 \\ h_6 & h_7 & h_8 \end{pmatrix}$
$\mathbf{H} \leftarrow \mathbf{N}_Q^{-1} \cdot \mathbf{H}' \cdot \mathbf{N}_P$ ▷ de-normalize $\mathbf{H}'$, see Eqn. (68)
return $\mathbf{H}$

GetNormalisationMatrix($\mathtt{X}$)
Input: $\mathtt{X} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{N-1})$, with $\boldsymbol{x}_j = (x_j, y_j)^\intercal$.

$N \leftarrow |\mathtt{X}|$
$\bar{x} \leftarrow \frac{1}{N} \sum_{j=0}^{N-1} x_j, \qquad \sigma_x^2 \leftarrow \frac{1}{N} \sum_{j=0}^{N-1} (x_j - \bar{x})^2$
$\bar{y} \leftarrow \frac{1}{N} \sum_{j=0}^{N-1} y_j, \qquad \sigma_y^2 \leftarrow \frac{1}{N} \sum_{j=0}^{N-1} (y_j - \bar{y})^2$
$s_x \leftarrow \sqrt{2/\sigma_x^2}$
$s_y \leftarrow \sqrt{2/\sigma_y^2}$
$\mathbf{N}_\mathtt{X} \leftarrow \begin{pmatrix} s_x & 0 & -s_x \bar{x} \\ 0 & s_y & -s_y \bar{y} \\ 0 & 0 & 1 \end{pmatrix}$ ▷ see Eqn. (181)
return $\mathbf{N}_\mathtt{X}$

Algorithm 4.3 Refinement of a single view homography by minimizing the projection error in the sensor image using non-linear (Levenberg-Marquart) optimization.

1: RefineHomography($\mathbf{H}, \mathcal{X}, \dot{\boldsymbol{U}}$)
Input: $\mathbf{H} = (H_{m,n})$, the initial 3×3 homography matrix; $\mathcal{X} = (\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1})$, with $\boldsymbol{X}_j = (X_j, Y_j)^\mathsf{T}$, the model points; $\dot{\boldsymbol{U}} = (\dot{\boldsymbol{u}}_0, \ldots, \dot{\boldsymbol{u}}_{N-1})$, with $\dot{\boldsymbol{u}}_j = (\dot{u}_j, \dot{v}_j)^\mathsf{T}$, the observed sensor points for a single camera view. Returns the numerically optimized homography $\mathbf{H}'$.
2: $N \leftarrow |\mathcal{X}|$
3: $\mathtt{X} \leftarrow (\boldsymbol{X}_0, \boldsymbol{X}_0, \boldsymbol{X}_1, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N-1}, \boldsymbol{X}_{N-1})$ ▷ $|\mathtt{X}| = 2N$
4: $\dot{\mathtt{Y}} \leftarrow (\dot{u}_0, \dot{v}_0, \dot{u}_1, \dot{v}_1, \ldots, \dot{u}_{N-1}, \dot{v}_{N-1})$ ▷ $|\mathtt{Y}| = 2N$
5: $\mathtt{h} \leftarrow (H_{0,0}, H_{0,1}, H_{0,2}, H_{1,0}, H_{1,2}, H_{1,2}, H_{2,0}, H_{2,1}, H_{2,2})$ ▷ flatten $\mathbf{H}$
6: $\mathtt{h}' \leftarrow$ Optimize(Val, Jac, $\mathtt{X}, \dot{\mathtt{Y}}, \mathtt{h}$) ▷ LM-optimization
7: $\mathbf{H}' \leftarrow \frac{1}{\mathtt{h}'_8} \cdot \begin{pmatrix} \mathtt{h}'_0 & \mathtt{h}'_1 & \mathtt{h}'_2 \\ \mathtt{h}'_3 & \mathtt{h}'_4 & \mathtt{h}'_5 \\ \mathtt{h}'_6 & \mathtt{h}'_7 & \mathtt{h}'_8 \end{pmatrix}$ ▷ compose $\mathbf{H}'$ from $\mathtt{h}'$ and normalize
8: **return** $\mathbf{H}'$

9: Val($\mathcal{X}, \mathtt{h}$) ▷ *value* function, invoked by Optimize()
Input: $\mathcal{X} = (\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1})$, the model points; $\mathtt{h} = (\mathtt{h}_0, \ldots, \mathtt{h}_8)$, parameter vector holding the 9 elements of the associated homography matrix $\mathbf{H}$. Returns a vector with $2N$ values.
10: $N \leftarrow |\mathcal{X}|$
11: $\mathtt{Y} \leftarrow$ new vector of length $2N$
12: **for** $j \leftarrow 0, \ldots, N-1$ **do**
13: $(X, Y) \leftarrow \boldsymbol{X}_j$ ▷ see Eqns. (70)–(71)
14: $s \leftarrow \mathtt{h}_6 \cdot X + \mathtt{h}_7 \cdot Y + \mathtt{h}_8$ ▷ $w = (\mathtt{h}_6, \mathtt{h}_7, \mathtt{h}_8) \cdot (X, Y, 1)^\mathsf{T}$
15: $\begin{pmatrix} u \\ v \end{pmatrix} \leftarrow \frac{1}{w} \cdot \begin{pmatrix} \mathtt{h}_0 & \mathtt{h}_1 & \mathtt{h}_2 \\ \mathtt{h}_3 & \mathtt{h}_4 & \mathtt{h}_5 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$ ▷ $\boldsymbol{u}_j = \hom^{-1}(\mathbf{H} \cdot \hom(\boldsymbol{X}_j))$
16: $\mathtt{Y}_{2j} \leftarrow u, \quad \mathtt{Y}_{2j+1} \leftarrow v$
17: **return** $\mathtt{Y}$

18: Jac($\mathcal{X}, \mathtt{h}$) ▷ *Jacobian* function, invoked by Optimize()
Input: $\mathcal{X} = (\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1})$, the model points; $\mathtt{h} = (\mathtt{h}_0, \ldots, \mathtt{h}_8)$, parameter vector with the 9 elements of the associated homography matrix $\mathbf{H}$. Returns the Jacobian matrix of size $2N \times 9$.
19: $N \leftarrow |\mathcal{X}|$
20: $\mathtt{J} \leftarrow$ new matrix of size $2N \times 9$
21: **for** $j \leftarrow 0, \ldots, N-1$ **do**
22: $(X, Y) \leftarrow \boldsymbol{X}_j$
23: $s_x \leftarrow \mathtt{h}_0 \cdot X + \mathtt{h}_1 \cdot Y + \mathtt{h}_2$ ▷ see Eqns. (73)–(76)
24: $s_y \leftarrow \mathtt{h}_3 \cdot X + \mathtt{h}_4 \cdot Y + \mathtt{h}_5$
25: $w \leftarrow \mathtt{h}_6 \cdot X + \mathtt{h}_7 \cdot Y + \mathtt{h}_8$
26: $\mathtt{J}_{2j,*} \leftarrow (\frac{X}{w}, \frac{Y}{w}, \frac{1}{w}, 0, 0, 0, \frac{-s_x \cdot X}{w^2}, \frac{-s_x \cdot Y}{w^2}, \frac{-s_x}{w^2})$ ▷ fill row $2j$
27: $\mathtt{J}_{2j+1,*} \leftarrow (0, 0, 0, \frac{X}{w}, \frac{Y}{w}, \frac{1}{w}, \frac{-s_y \cdot X}{w^2}, \frac{-s_y \cdot Y}{w^2}, \frac{-s_y}{w^2})$ ▷ fill row $2j+1$
28: **return** $\mathtt{J}$

Algorithm 4.4 Calculation of intrinsic camera parameters (Version A).

1: GetCameraIntrinsics($\mathcal{H}$)
Input: $\mathcal{H} = (\mathbf{H}_0, \ldots, \mathbf{H}_{M-1})$, a sequence of homography matrices. Returns the (common) intrinsic camera matrix $\mathbf{A}$.
2: $M = |\mathcal{H}|$
3: $\mathbf{V} \leftarrow$ new matrix of size $2M \times 6$
4: **for** $i \leftarrow 0, \ldots, M-1$ **do**
5: $\mathbf{H} \leftarrow \mathcal{H}(i)$
6: $\mathbf{V}_{2i,*} \leftarrow \boldsymbol{v}_{0,1}(\mathbf{H})$ ▷ fill row $2i$ (see def. below)
7: $\mathbf{V}_{2i+1,*} \leftarrow \boldsymbol{v}_{0,0}(\mathbf{H}) - \boldsymbol{v}_{1,1}(\mathbf{H})$ ▷ fill row $2i+1$
Find the least-squares solution to the homogen. system (e. g., by SVD):
8: $\boldsymbol{b} \leftarrow \text{Solve}(\mathbf{V} \cdot \boldsymbol{b} = \mathbf{0})$ ▷ $\boldsymbol{b} = (B_0, \ldots, B_5)$
9: $w \leftarrow B_0 B_2 B_5 - B_1^2 B_5 - B_0 B_4^2 + 2B_1 B_3 B_4 - B_2 B_3^2$ ▷ Eqn. (104)
10: $d \leftarrow B_0 B_2 - B_1^2$ ▷ Eqn. (105)
11: $\alpha \leftarrow \sqrt{w/(d \cdot B_0)}$ ▷ Eqn. (99)
12: $\beta \leftarrow \sqrt{w/d^2 \cdot B_0}$ ▷ Eqn. (100)
13: $\gamma \leftarrow \sqrt{w/(d^2 \cdot B_0)} \cdot B_1$ ▷ Eqn. (101)
14: $u_c \leftarrow (B_1 B_4 - B_2 B_3)/d$ ▷ Eqn. (102)
15: $v_c \leftarrow (B_1 B_3 - B_0 B_4)/d$ ▷ Eqn. (103)
16: $\mathbf{A} \leftarrow \begin{pmatrix} \alpha & \gamma & u_c \\ 0 & \beta & v_c \\ 0 & 0 & 1 \end{pmatrix}$
17: **return A**

18: Definition: $\boldsymbol{v}_{p,q}(\mathbf{H}) := \begin{pmatrix} H_{0,p} \cdot H_{0,q} \\ H_{0,p} \cdot H_{1,q} + H_{1,p} \cdot H_{0,q} \\ H_{1,p} \cdot H_{1,q} \\ H_{2,p} \cdot H_{0,q} + H_{0,p} \cdot H_{2,q} \\ H_{2,p} \cdot H_{1,q} + H_{1,p} \cdot H_{2,q} \\ H_{2,p} \cdot H_{2,q} \end{pmatrix}^{\mathsf{T}}$

Algorithm 4.5 Calculation of intrinsic camera parameters (Version B, using Cholesky decomposition). See Alg. 4.4 for the definition of $\boldsymbol{v}_{p,q}(\mathbf{H})$.

GetCameraIntrinsics($\mathcal{H}$)
Input: $\mathcal{H} = (\mathbf{H}_0, \ldots, \mathbf{H}_{M-1})$, a sequence of homography matrices.
Returns the (common) intrinsic camera matrix $\mathbf{A}$.
  $M = |\mathcal{H}|$
  $\mathbf{V} \leftarrow$ new matrix of size $2M \times 6$
  for $i \leftarrow 0, \ldots, M-1$ **do**
    $\mathbf{H} \leftarrow \mathcal{H}(i)$
    $\mathbf{V}_{2i,*} \leftarrow \boldsymbol{v}_{0,1}(\mathbf{H})$ ▷ set row vector $2i$ (see below)
    $\mathbf{V}_{2i+1,*} \leftarrow \boldsymbol{v}_{0,0}(\mathbf{H}) - \boldsymbol{v}_{1,1}(\mathbf{H})$ ▷ set row vector $2i+1$
  Solve the homog. system (e. g. by singular value decomposition):
  $\boldsymbol{b} \leftarrow \text{Solve}(\mathbf{V} \cdot \boldsymbol{b} = \mathbf{0})$ ▷ $\boldsymbol{b} = (B_0, \ldots, B_5)$
  $\mathbf{B} \leftarrow \begin{pmatrix} B_0 & B_1 & B_3 \\ B_1 & B_2 & B_4 \\ B_3 & B_4 & B_5 \end{pmatrix}$
  if $(B_0 < 0 \vee B_2 < 0 \vee B_5 < 0)$ **then**
    $\mathbf{B} \leftarrow -\mathbf{B}$ ▷ make sure $\mathbf{B}$ is positive definite
  $\mathbf{L} \leftarrow \text{Solve}(\mathbf{L} \cdot \mathbf{L}^\intercal = \mathbf{B})$ ▷ by Cholesky decomposition (see Sec. C)
  $\mathbf{A} \leftarrow (\mathbf{L}^{-1})^\intercal \cdot \mathbf{L}_{2,2}$ ▷ $\mathbf{L}_{2,2} \in \mathbb{R}$ is the element of $\mathbf{L}$ at position $(2,2)$
  return $\mathbf{A}$

Algorithm 4.6 Calculation of extrinsic view parameters.

GetExtrinsics($\mathbf{A}, \mathcal{H}$)
Input: $\mathcal{H} = (\mathbf{H}_0, \ldots, \mathbf{H}_{M-1})$, a sequence of homography matrices.
Returns a sequence of extrinsic view parameters $\mathcal{W} = (\mathbf{W}_0, \ldots, \mathbf{W}_{M-1})$, with $\mathbf{W}_i = (\mathbf{R}_i \mid \boldsymbol{t}_i)$.
  $\mathcal{W} \leftarrow ( )$
  for $i \leftarrow 0, \ldots, M-1$ **do**
    $\mathbf{H} \leftarrow \mathcal{H}(i)$
    $\mathbf{W} \leftarrow$ EstimateViewTransform($\mathbf{A}, \mathbf{H}$) ▷ $\mathbf{W} = (\mathbf{R} \mid \boldsymbol{t})$
    $\mathcal{W} \leftarrow \mathcal{W} \frown (\mathbf{W})$
  return $\mathcal{W}$

EstimateViewTransform($\mathbf{A}, \mathbf{H}$)
  $\boldsymbol{h}_0 \leftarrow \mathbf{H}_{*,0}$ ▷ $\boldsymbol{h}_0 = (H_{0,0}, H_{1,0}, H_{2,0})$
  $\boldsymbol{h}_1 \leftarrow \mathbf{H}_{*,1}$ ▷ $\boldsymbol{h}_1 = (H_{0,1}, H_{1,1}, H_{2,1})$
  $\boldsymbol{h}_2 \leftarrow \mathbf{H}_{*,2}$ ▷ $\boldsymbol{h}_2 = (H_{0,2}, H_{1,2}, H_{2,2})$
  $\kappa \leftarrow 1/\|\mathbf{A}^{-1} \cdot \boldsymbol{h}_0\|$
  $\mathbf{r}_0 \leftarrow \kappa \cdot \mathbf{A}^{-1} \cdot \boldsymbol{h}_0$
  $\mathbf{r}_1 \leftarrow \kappa \cdot \mathbf{A}^{-1} \cdot \boldsymbol{h}_1$
  $\mathbf{r}_2 \leftarrow \mathbf{r}_0 \times \mathbf{r}_1$ ▷ 3D cross (vector) product
  $\boldsymbol{t} \leftarrow \kappa \cdot \mathbf{A}^{-1} \cdot \boldsymbol{h}_2$ ▷ translation vector
  $\tilde{\mathbf{R}} \leftarrow (\mathbf{r}_0 \mid \mathbf{r}_1 \mid \mathbf{r}_2)$ ▷ initial rotation matrix
  $\mathbf{R} \leftarrow$ MakeTrueRotationMatrix($\tilde{\mathbf{R}}$) ▷ see end of Sec. 3.4.
  return $(\mathbf{R} \mid \boldsymbol{t})$

Algorithm 4.7 Estimation of the radial lens distortion parameters.

1: EstLensDistortion($\mathbf{A}, \mathcal{W}, \mathcal{X}, \dot{\mathcal{U}}$)
Input: $\mathbf{A}$, the estimated intrinsic camera parameters; $\mathcal{W} = (\mathbf{W}_0, \ldots, \mathbf{W}_{M-1})$, with $\mathbf{W}_i = (\mathbf{R}_i \mid \boldsymbol{t}_i)$, the estimated extrinsic parameters (camera views); $\mathcal{X} = (\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1})$, the target model points; $\dot{\mathcal{U}} = (\dot{\boldsymbol{U}}_0, \ldots, \dot{\boldsymbol{U}}_{M-1})$, the observed sensor points, with $\dot{\boldsymbol{U}}_i = (\dot{\boldsymbol{u}}_{i,0}, \ldots, \dot{\boldsymbol{u}}_{i,N-1})$ being the points for view i. Returns the vector of estimated lens distortion coefficients.

2: $M \leftarrow |\mathcal{W}|$ ▷ number of views
3: $N \leftarrow |\mathcal{X}|$ ▷ number of model points
4: $(u_c, v_c) \leftarrow (\mathbf{A}_{0,2}, \mathbf{A}_{1,2})$ ▷ proj. center (in sensor coord.)

Set up matrix $\mathbf{D}$ and vector $\dot{\boldsymbol{d}}$ (see Eqn. (119)):
5: $\mathbf{D} \leftarrow$ new matrix of size $2MN \times 2$
6: $\dot{\boldsymbol{d}} \leftarrow$ new vector of length $2MN$
7: $l \leftarrow 0$
8: **for** $i \leftarrow 0, \ldots, M-1$ **do** ▷ view index i
9: **for** $j \leftarrow 0, \ldots, N-1$ **do** ▷ for each model point
10: $(x, y) \leftarrow \check{P}(\mathbf{W}_i, \boldsymbol{X}_j)$ ▷ normalized proj. (see Eqn. (20))
11: $r \leftarrow \sqrt{x^2 + y^2}$ ▷ radius in the normalized proj.
12: $(u, v) \leftarrow P(\mathbf{A}, \mathbf{W}_i, \boldsymbol{X}_j)$ ▷ project to sensor (see Eqn. (24))
13: $(d_u, d_v) \leftarrow (u - u_c, v - v_c)$
14: $\mathbf{D}_{2l,*} \leftarrow (d_u \cdot r^2, d_u \cdot r^4)$ ▷ fill row $2l$ of matrix $\mathbf{D}$
15: $\mathbf{D}_{2l+1,*} \leftarrow (d_v \cdot r^2, d_v \cdot r^4)$ ▷ fill row $2l+1$ of matrix $\mathbf{D}$
16: $(\dot{u}, \dot{v}) \leftarrow \dot{\boldsymbol{u}}_{i,j}$ ▷ observed image point $\dot{\boldsymbol{u}}_{i,j}$
17: $\dot{\boldsymbol{d}}_{2l} \leftarrow (\dot{u} - u)$ ▷ set element $2l$ of vector $\dot{\boldsymbol{d}}$
18: $\dot{\boldsymbol{d}}_{2l+1} \leftarrow (\dot{v} - v)$ ▷ set element $2l + 1$ of vector $\dot{\boldsymbol{d}}$
19: $l \leftarrow l + 1$
20: $\boldsymbol{k} \leftarrow \text{Solve}(\mathbf{D} \cdot \boldsymbol{k} = \dot{\boldsymbol{d}})$ ▷ lin. least-squares solution, e. g., by SVD
21: **return** $\boldsymbol{k}$ ▷ $\boldsymbol{k} = (k_0, k_1)^\mathsf{T}$

Algorithm 4.8 Overall, non-linear refinement. Goal: find the intrinsic and extrinsic parameters that minimize $E = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \|\dot{\boldsymbol{u}}_{i,j} - P(\boldsymbol{a}, \boldsymbol{w}_i, \boldsymbol{X}_j)\|^2$.

1: RefineAll($\mathbf{A}, \boldsymbol{k}, \mathcal{W}, \mathcal{X}, \dot{\mathcal{U}}$)

Input: $\mathbf{A}$: camera intrinsics; $\boldsymbol{k}$: lens distortion coefficients; $\mathcal{W} = (\mathbf{W}_i)$: extrinsic view parameters; $\mathcal{X} = (\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1})$: the target model points; $\dot{\mathcal{U}} = (\dot{\boldsymbol{u}}_{i,j})$: the observed image points, Returns refined estimates $\langle \mathbf{A}_{\text{opt}}, \boldsymbol{k}_{\text{opt}}, \mathcal{W}_{\text{opt}} \rangle$ for the camera intrinsics, distortion parameters, and the camera view parameters, respectively.

2: $\mathtt{P}_{\text{init}} \leftarrow$ ComposeParameterVector($\mathbf{A}, \boldsymbol{k}, \mathcal{W}$) ▷ see below

3: $\mathtt{X} \leftarrow (\underbrace{\boldsymbol{X}_0, \boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1}, \boldsymbol{X}_{N-1}}_{\text{for view } 0}, \ldots, \underbrace{\boldsymbol{X}_0, \boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1}, \boldsymbol{X}_{N-1}}_{\text{for view } M-1})^\mathsf{T}$

4: $\dot{\mathtt{Y}} \leftarrow (\underbrace{\dot{u}_{0,0}, \dot{v}_{0,0}, \ldots, \dot{u}_{0,N-1}, \dot{v}_{0,N-1}}_{\text{for view } 0}, \ldots, \underbrace{\dot{u}_{M-1,0}, \ldots, \dot{v}_{M-1,N-1}}_{\text{for view } M-1})^\mathsf{T}$

5: $\mathtt{P} \leftarrow$ Optimize(Val, Jac, $\mathtt{X}, \dot{\mathtt{Y}}, \mathtt{P}_{\text{init}}$) ▷ LM-optim., see Eqn. (201)

6: **return** DecomposeParameterVector(P) ▷ $= \langle \mathbf{A}_{\text{opt}}, \boldsymbol{k}_{\text{opt}}, \mathcal{W}_{\text{opt}} \rangle$

7: ComposeParameterVector($\mathbf{A}, \boldsymbol{k}, \mathcal{W}$)
Input: $\mathbf{A} = \left(\begin{smallmatrix} \alpha & \gamma & u_c \\ 0 & \beta & v_c \end{smallmatrix}\right)$: the estimated camera intrinsics; $\boldsymbol{k} = (k_0, k_1)$: the estimated lens distortion coefficients; $\mathcal{W} = (\mathbf{W}_0, \ldots, \mathbf{W}_{M-1})$, with $\mathbf{W}_i = (\mathbf{R}_i \mid \boldsymbol{t}_i)$: extrinsic view parameters. Returns a parameter vector P of length $7 + M \cdot 6$.

8: $\boldsymbol{a} \leftarrow (\alpha, \beta, \gamma, u_c, v_c, k_0, k_1)$ ▷ see Eqn. (120)

9: $\mathtt{P} \leftarrow \boldsymbol{a}, \quad M \leftarrow |\mathcal{W}|$

10: **for** view index $i \leftarrow 0, \ldots, M-1$ **do**

11: $(\mathbf{R} \mid \boldsymbol{t}) \leftarrow \mathbf{W}_i$ ▷ $\boldsymbol{t} = (t_x, t_y, t_z)^\mathsf{T}$

12: $\boldsymbol{\rho} \leftarrow$ ToRodriguesVector($\mathbf{R}$) ▷ $\boldsymbol{\rho} = (\rho_x, \rho_y, \rho_z)$, see Alg. 4.10

13: $\boldsymbol{w} \leftarrow \boldsymbol{\rho} \backsim (\boldsymbol{t}^\mathsf{T})$ ▷ $\boldsymbol{w} = (\rho_x, \rho_y, \rho_z, t_x, t_y, t_z)$

14: $\mathtt{P} \leftarrow \mathtt{P} \backsim \boldsymbol{w}$

15: **return** P

16: DecomposeParameterVector(P)
Input: P, parameter vector of length $7 + M \cdot 6$ (with M being the number of views). Returns the associated camera matrix $\mathbf{A}$, the lens distortion coefficients $\boldsymbol{k}$, and the view transformations $\mathcal{W} = (\mathbf{W}_0, \ldots, \mathbf{W}_{M-1})$.

17: $(\alpha, \beta, \gamma, u_c, v_c, k_0, k_1) \leftarrow (\mathtt{p}_0, \ldots, \mathtt{p}_6)$

18: $\mathbf{A} \leftarrow \begin{pmatrix} \alpha & \gamma & u_c \\ 0 & \beta & v_c \\ 0 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{k} \leftarrow (k_0, k_1)\,, \quad \mathcal{W} \leftarrow (\,), \quad M \leftarrow |\mathcal{W}|$

19: **for** view index $i \leftarrow 0, \ldots, M-1$ **do**

20: $m \leftarrow 7 + 6 \cdot i$

21: $\boldsymbol{\rho} \leftarrow (\mathtt{p}_m, \ldots, \mathtt{p}_{m+2})$ ▷ $\boldsymbol{\rho} = (\rho_x, \rho_y, \rho_z)$

22: $\boldsymbol{t} \leftarrow (\mathtt{p}_{m+3}, \ldots, \mathtt{p}_{m+5})^\mathsf{T}$ ▷ $\boldsymbol{t} = (t_x, t_y, t_z)^\mathsf{T}$

23: $\mathbf{R} \leftarrow$ ToRotationMatrix($\boldsymbol{\rho}$) ▷ see Alg. 4.10

24: $\mathbf{W} \leftarrow (\mathbf{R} \mid \boldsymbol{t})$

25: $\mathcal{W} \leftarrow \mathcal{W} \backsim (\mathbf{W})$

26: **return** $\langle \mathbf{A}, \boldsymbol{k}, \mathcal{W} \rangle$

Algorithm 4.9 Value and Jacobian functions referenced in Alg. 4.8. The function $\textsc{Val}(\mathcal{X}, \mathtt{P})$ calculates the value of the optimization model for the target points in $\mathcal{X}$ and the model parameters $\mathtt{P}$. The function $\textsc{Jac}(\mathcal{X}, \mathtt{P})$ calculates the associated Jacobian matrix, as specified in Eqn. (133).

$\textsc{Val}(\mathcal{X}, \mathtt{P})$ ▷ the *value* function
Input: $\mathcal{X} = (\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1})$, the target model points; $\mathtt{P} = (\mathtt{p}_0, \ldots, \mathtt{p}_{K-1})$, the vector of camera and all view parameters. Returns a vector of length $2MN$ containing the projected point coordinates for all views.

$M \leftarrow |\mathcal{W}|$ ▷ number of views
$N \leftarrow |\mathcal{X}|$ ▷ number of model points
$\boldsymbol{a} \leftarrow (\mathtt{p}_0, \ldots, \mathtt{p}_6)$ ▷ get the 7 camera parameters from $\mathtt{P}$
$\mathtt{Y} \leftarrow (\,)$
for $i \leftarrow 0, \ldots, M-1$ **do** ▷ view index
  $m \leftarrow 7 + 6 \cdot i$
  $\boldsymbol{w} \leftarrow (\mathtt{p}_m, \ldots, \mathtt{p}_{m+5})$ ▷ extract view parameters $\boldsymbol{w}_i$
  for $j \leftarrow 0, \ldots, N-1$ **do** ▷ point index
    $(u, v) \leftarrow P(\boldsymbol{a}, \boldsymbol{w}, \boldsymbol{X}_j)$ ▷ project the model point $\boldsymbol{X}_j$
    $\mathtt{Y} \leftarrow \mathtt{Y} \frown (u, v)$
return $\mathtt{Y}$ ▷ $\mathtt{Y} = (u_{0,0}, v_{0,0}, \ldots, u_{M-1,N-1}, v_{M-1,N-1})$

$\textsc{Jac}(\mathcal{X}, \mathtt{P})$ ▷ the *Jacobian* function
Input: $\mathcal{X} = (\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{N-1})$, the target model points; $\mathtt{P} = (\mathtt{p}_0, \ldots, \mathtt{p}_{K-1})$, the vector of camera and all view parameters. Returns the Jacobian matrix of size $2MN \times K$ (with $K = 7 + 6M$). Partial derivatives can be calculated analytically or numerically (see Sec. 3.6.4 for details).

$M \leftarrow |\mathcal{W}|$ ▷ number of views
$N \leftarrow |\mathcal{X}|$ ▷ number of model points
$K \leftarrow |\mathtt{P}|$ ▷ number of parameters
$\boldsymbol{a} \leftarrow (\mathtt{p}_0, \ldots, \mathtt{p}_6)$ ▷ get the 7 camera parameters from $\mathtt{P}$
$\mathtt{J} \leftarrow$ new matrix of size $2MN \times K$
for $k \leftarrow 0, \ldots, K-1$ **do** ▷ parameter index
  $r \leftarrow 0$
  for $i \leftarrow 0, \ldots, M-1$ **do** ▷ view index
    $m \leftarrow 7 + 6 \cdot i$
    $\boldsymbol{w} \leftarrow (\mathtt{p}_m, \ldots, \mathtt{p}_{m+5})$ ▷ extract view parameters $\boldsymbol{w}_i$
    for $j \leftarrow 0, \ldots, N-1$ **do** ▷ point index
      $\mathtt{J}_{r+0,k} \leftarrow \frac{\partial P_\mathrm{x}(\boldsymbol{a}, \boldsymbol{w}, \boldsymbol{X}_j)}{\partial \mathtt{p}_k}$ ▷ partial deriv. of P_x w.r.t. $\mathtt{p}_k$
      $\mathtt{J}_{r+1,k} \leftarrow \frac{\partial P_\mathrm{y}(\boldsymbol{a}, \boldsymbol{w}, \boldsymbol{X}_j)}{\partial \mathtt{p}_k}$ ▷ partial deriv. of P_y w.r.t. $\mathtt{p}_k$
      $r \leftarrow r + 2$
return $\mathtt{J}$

Algorithm 4.10 Conversions between a rotation matrix and the associated Rodrigues rotation vector. Note that the function ToRodriguesVector($\mathbf{R}$) is usually implemented via quaternions, while the direct method shown below was adopted from [11]. The corresponding Java code is based on the implementation provided by the *Apache Commons Math* library. See also Sec. A.2.4 in the Appendix.

ToRodriguesVector($\mathbf{R}$)
Input: $\mathbf{R} = (R_{i,j})$, a proper 3D rotation matrix. Returns the associated Rodrigues rotation vector ($\boldsymbol{\rho}$).

$\boldsymbol{p} \leftarrow 0.5 \cdot \begin{pmatrix} R_{2,1} - R_{1,2} \\ R_{0,2} - R_{2,0} \\ R_{1,0} - R_{0,1} \end{pmatrix}$
$c \leftarrow 0.5 \cdot (\text{trace}(\mathbf{R}) - 1)$
if $\|\boldsymbol{p}\| = 0$ **then**
  if $c = 1$ **then** ▷ **Case 1**
    $\boldsymbol{\rho} = (0, 0, 0)$
  else if $c = -1$ **then** ▷ **Case 2**
    $\mathbf{R}_+ \leftarrow \mathbf{R} + \mathbf{I}$
    $\boldsymbol{v} \leftarrow$ column vector of $\mathbf{R}_+$ with max. norm
    $\boldsymbol{u} \leftarrow \frac{1}{\|\boldsymbol{v}\|} \cdot \boldsymbol{v}$ ▷ rotation axis $\boldsymbol{u} = (u_0, u_1, u_2)$
    if $(u_0 < 0) \vee (u_0 = 0 \wedge u_1 < 0) \vee (u_0 = u_1 = 0 \wedge u_2 < 0)$ **then**
      $\boldsymbol{u} \leftarrow -\boldsymbol{u}$ ▷ switch sign for correct hemisphere
    $\boldsymbol{\rho} = \pi \cdot \boldsymbol{u}$ ▷ $\theta = \pi$
  else
    $\boldsymbol{\rho} = \text{nil}$ ▷ this should never happen
else ▷ **Case 3**
  $\boldsymbol{u} \leftarrow \frac{1}{\|\boldsymbol{p}\|} \cdot \boldsymbol{p}$ ▷ rotation axis
  $\theta \leftarrow \tan^{-1}\left(\frac{\|\boldsymbol{p}\|}{c}\right)$ ▷ use `Math.atan2(`$\|\boldsymbol{p}\|, c$`)` or equiv.
  $\boldsymbol{\rho} \leftarrow \theta \cdot \boldsymbol{u}$
return $\boldsymbol{\rho}$

ToRotationMatrix($\boldsymbol{\rho}$)
Input: $\boldsymbol{\rho} = (\rho_x, \rho_y, \rho_z)$, a 3D Rodrigues rotation vector. Returns the associated rotation matrix ($\mathbf{R}$).

$\theta \leftarrow \|\boldsymbol{\rho}\|$
$\hat{\boldsymbol{\rho}} \leftarrow \frac{1}{\|\boldsymbol{\rho}\|} \cdot \boldsymbol{\rho}$ ▷ unit vector $\hat{\boldsymbol{\rho}} = (\hat{\rho}_x, \hat{\rho}_y, \hat{\rho}_z)$
$\mathbf{W} \leftarrow \begin{pmatrix} 0 & -\hat{\rho}_z & \hat{\rho}_y \\ \hat{\rho}_z & 0 & -\hat{\rho}_x \\ -\hat{\rho}_y & \hat{\rho}_x & 0 \end{pmatrix}$
$\mathbf{R} \leftarrow \mathbf{I} + \mathbf{W} \cdot \sin(\theta) + \mathbf{W} \cdot \mathbf{W} \cdot (1 - \cos(\theta))$
return $\mathbf{R}$

5 Image Rectification

Removing the camera's lens distortion from a real image is an important task, e. g., in the context of augmented reality systems. Graphic APIs (such as OpenGL, DirectX etc.) do not consider lens distortion, i. e., the virtual cameras used to render images are purely "pinhole".

5.1 Removing lens distortion

We can remove the lens distortion from a given real image I by geometric transformation to a new image I', in which each point $\boldsymbol{u}' = (u', v')^\mathsf{T}$ is related to the original coordinates $\boldsymbol{u} = (u, v)^\mathsf{T}$ as

$$\boldsymbol{u}' = T(\boldsymbol{u}). \tag{134}$$

Here T is the geometric 2D mapping $I \mapsto I'$ which only depends on the intrinsic camera parameters (the extrinsic view parameters are of no relevance here). For rendering the new image I' by the usual *target-to-source mapping* (see [2, Sec. 21.2.2]) we are primarily interested in the *inverse* transformation T^{-1},

$$T^{-1} : I' \mapsto I, \tag{135}$$

which (luckily) makes everything easier, since it avoids inverting the radial distortion function $D()$. To get from the new image I' to the observed image I, the associated geometric transformation T^{-1} consists of the following steps:

Step 1: Moving backwards in the projection chain[19] from the rectified sensor points (u', v'), the corresponding points on the normalized image plane are obtained be inverting the (known) internal camera mapping $\mathbf{A}$ (see Eqn. (37)) as

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} \alpha & \gamma & u_c \\ 0 & \beta & v_c \\ 0 & 0 & 1 \end{pmatrix}^{-1}}_{\mathbf{A}} \cdot \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{1}{\alpha} & -\frac{\gamma}{\alpha\beta} & \frac{-bu_c+\gamma v_c}{\alpha\beta} \\ 0 & \frac{1}{\beta} & -\frac{v_c}{\beta} \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{A}^{-1}} \cdot \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} \tag{136}$$

and hence

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \frac{1}{\alpha} & -\frac{\gamma}{\alpha\beta} & \frac{-bu_c+\gamma v_c}{\alpha\beta} \\ 0 & \frac{1}{\beta} & -\frac{v_c}{\beta} \end{pmatrix} \cdot \begin{pmatrix} u' \\ v' \end{pmatrix}, \tag{137}$$

or simply (a kind of *pseudo-inverse*)

$$\boldsymbol{x}' = \mathbf{A}^{-1} \cdot \boldsymbol{u}'. \tag{138}$$

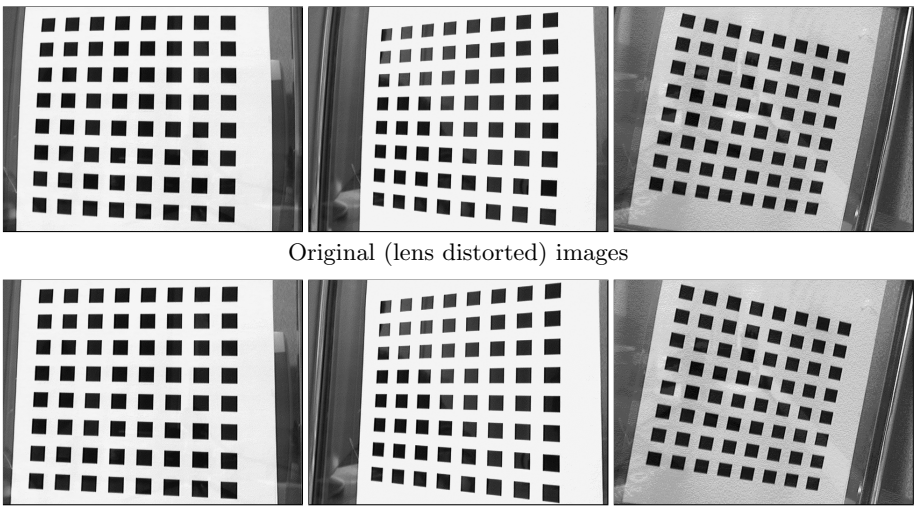
Since there should be no lens distortion for the image I', $\boldsymbol{x}'$ is implicitly considered the *undistorted* position in the normalized projection plane.

Step 2: Next, moving *forward* to the distorted image I, we apply the radial warping (see Eqn. (31))

$$\tilde{\boldsymbol{x}} = \mathrm{warp}(\boldsymbol{x}', \boldsymbol{k}) = \boldsymbol{x}' \cdot [1 + D(\|\boldsymbol{x}'\|, \boldsymbol{k})], \tag{139}$$

with the camera's lens distortion parameters $\boldsymbol{k} = (k_0, k_1)$.

[19] See Fig. 3.



Original (lens distorted) images

Rectified images

Figure 5: Image rectification example.

Step 3: Finally, we again apply the internal camera mapping (Eqn. (37)),

$$\boldsymbol{u} = \mathbf{A} \cdot \tilde{\boldsymbol{x}}, \tag{140}$$

to obtain the coordinates $\boldsymbol{u}$ in the observed image I.

In summary, the required geometric mapping T^{-1} from the rectified image I' to the distorted image I can be written as

$$\boldsymbol{u} = T^{-1}(\boldsymbol{u}') = \mathbf{A} \cdot \text{warp}(\mathbf{A}^{-1} \cdot \boldsymbol{u}'). \tag{141}$$

Now the undistorted image I' can be easily rendered from the distorted image I using target-to-source rendering [2, Sec. 21.2] as follows:

1: **for** all image coordinates $\boldsymbol{u}'$ of I' **do** ▷ $\boldsymbol{u}' \in \mathbb{Z}^2$
2: $\quad \boldsymbol{u} \leftarrow T^{-1}(\boldsymbol{u}')$ ▷ $\boldsymbol{u} \in \mathbb{R}^2$
3: $\quad val \leftarrow \text{Interpolate}(I, \boldsymbol{u})$ ▷ interpolate I at pos. $\boldsymbol{u}$
4: $\quad I'(\boldsymbol{u}') \leftarrow val$

5.2 Simulating different camera intrinsics

For some reason it may be interesting to simulate a camera b with different parameters than the camera a used for the original images. In this case, the geometric mapping in Eqn. (141) changes to

$$T_{ab}^{-1}(\boldsymbol{u}') = \mathbf{A}_a \cdot \text{warp}_a(\text{warp}_b^{-1}(\mathbf{A}_b^{-1} \cdot \boldsymbol{u}')), \tag{142}$$

where warp_b^{-1} denotes the inverse radial lens distortion for camera b, and $\mathbf{A}_a, \mathbf{A}_b$ are the intrinsic matrices for the two cameras (which may be the same, of course). Thus, given an image I_a that was taken with camera a we can transform

it to a new image I_b that would show the same scene if taken with a different camera b with the same view position and orientation, i. e.,

$$T_{ab}^{-1} : I_b \mapsto I_a, \tag{143}$$

with I_a and I_b being the source and target image, respectively.

The only additional task in Eqn. (142) is the inversion of the radial lens distortion function warp($\boldsymbol{x}$). From Eqn. (31) we get

$$\text{warp}(\boldsymbol{x}, \boldsymbol{k}) = \tilde{\boldsymbol{x}} = \boldsymbol{x} \cdot [1 + D(\|\boldsymbol{x}\|, \boldsymbol{k})] \tag{144}$$

$$= \boldsymbol{x} \cdot \frac{\|\boldsymbol{x}\| \cdot [1 + D(\|\boldsymbol{x}\|, \boldsymbol{k})]}{\|\boldsymbol{x}\|} = \boldsymbol{x} \cdot \frac{\tilde{r}}{r} = \boldsymbol{x} \cdot \frac{f_{\text{rad}}(r)}{r}, \tag{145}$$

where (see Eqns. 30 and 32)

$$f_{\text{rad}}(r) = r \cdot [1 + D(r, \boldsymbol{k})] = r + k_0 \cdot r^3 + k_1 \cdot r^5. \tag{146}$$

Thus, given a point $\tilde{\boldsymbol{x}}$ with radius $\tilde{r}$, the *inverse warping* is defined as

$$\text{warp}^{-1}(\tilde{\boldsymbol{x}}, \boldsymbol{k}) = \boldsymbol{x} = \tilde{\boldsymbol{x}} \cdot \frac{r}{\tilde{r}} = \tilde{\boldsymbol{x}} \cdot \frac{f_{\text{rad}}^{-1}(\tilde{r})}{\tilde{r}} = \tilde{\boldsymbol{x}} \cdot \frac{f_{\text{rad}}^{-1}(\|\tilde{\boldsymbol{x}}\|)}{\|\tilde{\boldsymbol{x}}\|}. \tag{147}$$

5.2.1 Inverting the radial distortion

Calculating the inverse of the warping function $f_{\text{rad}}^{-1}(\tilde{r})$ for a given distorted radius $\tilde{r}$ means to find the argument value(s) r, such that $f_{\text{rad}}(r) = \tilde{r}$ or

$$f_{\text{rad}}(r) - \tilde{r} = r + k_0 \cdot r + k_1 \cdot r^5 - \tilde{r} = 0. \tag{148}$$

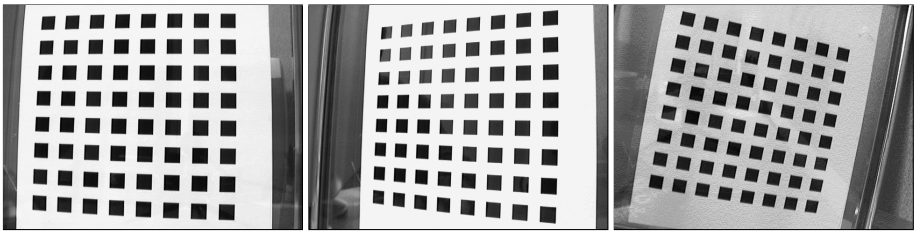
Thus, for a given (fixed) value $\tilde{r}$, the solution is a root of the polynomial in r,

$$g(r) = -\tilde{r} + r + k_0 \cdot r^3 + k_1 \cdot r^5, \tag{149}$$

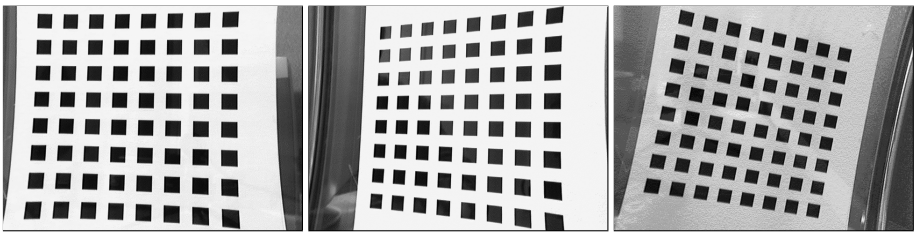
for $r \geq 0$. Unfortunately (at least to the knowledge of the author), the solution does not come in closed form but must be found with numerical techniques, such as the Newton-Raphson method. Since the radial mapping $f_{\text{rad}}(r)$ is usually close to the identity function, we can use $r_{\text{init}} = \tilde{r}$ as an *ad hoc* start value for the root finder.[20] The following code segment (contained in class `Camera`) shows the concrete implementation of $f_{\text{rad}}^{-1}(\tilde{r})$ using a numerical solver from the ACM library:

```
double fRadInv(double R) { // R is the distorted radius
  NewtonRaphsonSolver solver = new NewtonRaphsonSolver();
  int maxEval = 20;
  double k0 = K[0];
  double k1 = K[1];
  double[] coefficients = {-R, 1, 0, k0, 0, k1};
  PolynomialFunction g = new PolynomialFunction(coefficients);
  double rInit = R;
  double r = solver.solve(maxEval, g, rInit);
  return r;      // the undistorted radius
}
```

[20] More intelligent approaches can be imagined. This implementation uses the ACM `NewtonRaphsonSolver` class which only requires an initial start value. This start value may be too far away for root solvers that require an initial min/max bracket, with function values of opposite sign at the ends of the bracket (such as `LaguerreSolver` and `BrentSolver`).



Images taken with the original camera ($k_0 = -0.2286$, $k_1 = 0.1904$).



Transformed images with modified distortion parameters ($k_0 = -0.1$, $k_1 = 2.0$).

Figure 6: Modified lens distortion example. The original images (top row) are taken with camera a. For camera b (bottom row) only the lens distortion parameters k_0, k_1 were changed. All other intrinsic camera parameters are identical to camera a.

The solution is typically found after only a few iterations. Note that the above `fRadInv()` method is called for each image pixel, thus an efficient implementation is important.

An example of images being transformed to a modified camera are shown in Fig. 6. In this case the two cameras are identical except for the lens distortion parameters.

6 Java/ImageJ Implementation

This section describes a Java implementation of Zhang's calibration method that closely follows the above description (being analogously structured and using similar symbols wherever possible). It consists of a small Java API and several *ImageJ*[21] plugins demonstrating its use.[22] The API requires *Apache Commons Math*[23] as the only external library. Note that this implementation performs only the geometric part of the calibration and does not include any functionality for the detection of target points in the calibration images.

6.1 API description

Only the most important classes and methods are listed below. For additional information consult the JavaDoc documentation or the source files.

[21] http://rsbweb.nih.gov/ij/index.html

[22] The source code for this library is openly available at www.imagingbook.com.

[23] http://commons.apache.org/math/

Class `Calibrator` (package `calibration.lib.zhang`)

`Calibrator(Parameters params, Point2D[] model)`
: Constructor. Takes a parameter object (of type `Calibrator.Parameters`) and a sequence of 2D model points $(X_j, Y_j, 0)$.

`void addView(Point2D[] pts)`
: Adds a new view set, i. e., a sequence of observed image points (u_j, v_j). These points must be in correspondence with the model points passed to the constructor, i. e., they must have the same count and ordering.

`Camera calibrate()`
: Performs the actual calibration (using the supplied model and view sets) and returns a `Camera` object that specifies all intrinsic camera parameters.

`Camera getInitialCamera()`
: Returns the initial estimate of the camera parameters (before refinement).

`Camera getFinalCamera()`
: Returns the final camera parameters (same as `calibrate()`).

`ViewTransform[] getInitialViews()`
: Returns the initial estimates of the view transformations (extrinsic camera parameters) for each view set.

`ViewTransform[] getFinalViews()`
: Returns the final view transformations (extrinsic camera parameters) for each view set.

Class `Camera` (package `calibration.lib.zhang`)

`Camera(double alpha, double beta, double gamma, double uc, double vc, double k0, double k1))`
: Constructor. Takes the intrinsic parameters α, β, γ, u_c, v_c and the lens distortion parameters k_0, k_1. All camera objects are immutable.

`Camera (RealMatrix A, double[] K)`
: Constructor. Creates a standard camera with the intrinsic transformation matrix `A` and a vector of distortion parameters `K` $= (k_0, k_1, \ldots)$.

`Camera (double[] s)`
: Constructor. Takes the camera parameters as a single vector $s = (\alpha, \beta, \gamma, u_c, v_c, k_0, k_1)$. This constructor is primarily used internally for non-linear optimization.

6.2 ImageJ Demo plugins

The distribution includes Zhang's original set of test images (packaged as Java resources), which are used by the following ImageJ demo plugins. Other image data sets can be easily incorporated.

`Open_Test_Images`
: Opens Zhang's standard calibration images (bundled with the `EasyCalib` program) as a stack of RGB images. The image data are stored as a

resource in the local Java class tree. This plugin also demonstrates the use of the resource access mechanism.

`Demo_Zhang_Projection`
: This plugin projects opens an image stack containing the 5 Zhang test images and projects the model points into each view, using the (known) camera and view parameters. All data are part of Zhang's demo data set that comes with the `EasyCalib` program. No calibration is performed.

`Demo_Zhang_Projection_Overlay` Same as above, but all graphic elements are drawn as non-destructive vector overlays (look closely!). The complete stack with overly can be saved as a TIFF file.

`Demo_Zhang_Calibration`
: This plugin performs Zhang's camera calibration on the pre-calculated point data for the N given target views. Based on the estimated intrinsic and extrinsic (view) parameters, the corner points of the 3D target model are then projected onto the corresponding calibration images (a stack). All rendering is done by pixel drawing (no graphic overlays).

`Demo_Rectification`
: This plugin projects opens an image stack containing the 5 Zhang test images and removes the lens distortion based on the calibrated camera parameters. The resulting rectified frames are shown in a new image stack.

`Demo_Replace_Camera`
: This plugin projects opens an image stack containing the 5 Zhang test images (assumed to be taken with camera A) and re-renders the images by mapping them to a new camera B. In this example, only the lens distortion coefficients are modified but in principle all intrinsic parameters of camera B could be changed.

`Demo_View_Interpolation`
: This plugin performs interpolation of views, given a sequence of key views. Translations (3D camera positions) are interpolated linearly. Pairs of rotations are interpolated by linear mixture of the corresponding quaternion representations.[24]

[24] See http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-17-quaternions/#How_do_I_interpolate_between_2_quaternions__.

Appendix

A 3D/2D Geometry

Throughout this document we use

$$\boldsymbol{x} = (x, y, z)^\mathsf{T} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{150}$$

to denote a 3D vector or the position of a 3D point and

$$\boldsymbol{u} = (u, v)^\mathsf{T} = \begin{pmatrix} u \\ v \end{pmatrix} \tag{151}$$

as the 2D position of an image point on the sensor plane.

A.1 Homogeneous coordinates

To convert a n-dimensional *cartesian* point $\boldsymbol{x} = (x_0, \dots, x_{n-1})^\mathsf{T}$ to homogeneous coordinates, we use the notation[25]

$$\underline{\boldsymbol{x}} = \hom(\boldsymbol{x}) = \begin{pmatrix} \underline{x}_0 \\ \vdots \\ \underline{x}_{n-1} \\ \underline{x}_n \end{pmatrix} = \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \\ 1 \end{pmatrix}, \tag{152}$$

Similarly, for converting a homogeneous vector $\underline{\boldsymbol{x}} = (\underline{x}_0, \dots, \underline{x}_n)^\mathsf{T}$ back to Cartesian coordinates we write

$$\boldsymbol{x} = \hom^{-1}(\underline{\boldsymbol{x}}) = \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix} = \frac{1}{\underline{x}_n} \cdot \begin{pmatrix} \underline{x}_0 \\ \vdots \\ \underline{x}_{n-1} \end{pmatrix}, \tag{153}$$

assuming that $\underline{x}_n \neq 0$.

Two homogeneous points $\underline{\boldsymbol{x}}_1$, $\underline{\boldsymbol{x}}_2$ are *equivalent* ($\equiv$), if they map to the same Cartesian point, i. e.,

$$\underline{\boldsymbol{x}}_1 \equiv \underline{\boldsymbol{x}}_2 \iff \hom^{-1}(\underline{\boldsymbol{x}}_1) = \hom^{-1}(\underline{\boldsymbol{x}}_2). \tag{154}$$

Moreover, since $\hom^{-1}(\underline{\boldsymbol{x}}) = \hom^{-1}(s \cdot \underline{\boldsymbol{x}})$ for any nonzero factor $s \in \mathbb{R}$, a scaled homogeneous point is equivalent to the original point, i. e.,

$$\underline{\boldsymbol{x}} \equiv s \cdot \underline{\boldsymbol{x}}. \tag{155}$$

Homogeneous coordinates can be used for vector spaces of arbitrary dimension, including 2D coordinates.

[25] The operator hom( ) is introduced here for convenience and clarity. For some reason no suitable standard operators seems to exist.

A.2 Rigid body transformations in 3D

When an object moves in 3D without changing its size or shape, it is subject to a rigid transformation, changing position and orientation. Let us assume that a rigid 3D object is represented by the point set $\boldsymbol{X}_0, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N-1}$. Under rigid body transformation every (non-homogeneous) 3D point $\boldsymbol{X}_i = (X_i, Y_i, Z_i)^\mathsf{T}$ on that object is mapped to a new point

$$\boldsymbol{X}'_i = \mathbf{R} \cdot \boldsymbol{X}_i + \boldsymbol{t} \; . \tag{156}$$

The rigid motion is modeled in (156) as a 3D rotation (about the origin of the coordinate system) followed by a translation. $\mathbf{R}$ is an (orthonormal[26]) 3D rotation matrix

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \tag{157}$$

(as described below) and $\boldsymbol{t}$ is a 3D translation vector

$$\boldsymbol{t} = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} . \tag{158}$$

A.2.1 Rotations in 3D

The matrix $\mathbf{R}$ in (157), which describes arbitrary rotation in 3D, is composed of three individual rotations $\mathbf{R}_x, \mathbf{R}_y, \mathbf{R}_z$ about the X, Y, and Z axes, respectively: A rotation about the X-axis by an angle θ_x:

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{pmatrix} . \tag{159}$$

A rotation about the Y-axis by an angle θ_y:

$$\mathbf{R}_y = \begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{pmatrix} . \tag{160}$$

A rotation about the Z-axis by an angle θ_z:

$$\mathbf{R}_z = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix} . \tag{161}$$

The complete matrix $\mathbf{R}$ for an arbitrary rotation in 3D is obtained as

$$\mathbf{R} = \mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \tag{162}$$

$$= \begin{pmatrix} \cos\theta_y \cos\theta_z & -\cos\theta_y \sin\theta_z & \sin\theta_y \\ \sin\theta_x \sin\theta_y \cos\theta_z + \cos\theta_x \sin\theta_z & \cos\theta_x \cos\theta_z - \sin\theta_x \sin\theta_y \sin\theta_z & -\sin\theta_x \cos\theta_y \\ -\cos\theta_x \sin\theta_y \cos\theta_z + \sin\theta_x \sin\theta_z & \sin\theta_x \cos\theta_z + \cos\theta_x \sin\theta_y \sin\theta_z & \cos\theta_x \cos\theta_y \end{pmatrix} . \tag{163}$$

[26] A quadratic matrix $\mathbf{R}$ is *orthonormal* if $\mathbf{R} \cdot \mathbf{R}^\mathsf{T} = \mathbf{R}^\mathsf{T} \cdot \mathbf{R} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix.

Notice that the *order of rotations* in (162) is important, i.e., the result of $\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z$ is different to $\mathbf{R}_z\mathbf{R}_y\mathbf{R}_x$ for the same set of angles θ_x, θ_y, θ_z (also see [12, p. 333]). Using homogeneous coordinates, the rigid rotation by $\mathbf{R}$ can be expressed as $\underline{\boldsymbol{X}}'_i = \mathbf{M}_\mathrm{R} \cdot \underline{\boldsymbol{X}}_i$ or

$$\begin{pmatrix} X'_i \\ Y'_i \\ Z'_i \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{M}_\mathrm{R}} \cdot \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{pmatrix}. \tag{164}$$

Obviously, homogeneous coordinates are not required to express a pure rotation alone.

A.2.2 Translation in 3D

The 3D translation required in (156),

$$\boldsymbol{X}'_i = \boldsymbol{X}_i + \boldsymbol{t} = \begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}. \tag{165}$$

can also be specified as a matrix multiplication with homogeneous coordinates in the form $\underline{\boldsymbol{X}}'_i = \mathbf{M}_\mathrm{T} \cdot \underline{\boldsymbol{X}}_i$, or

$$\begin{pmatrix} X'_i \\ Y'_i \\ Z'_i \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{M}_\mathrm{T}} \cdot \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{pmatrix}. \tag{166}$$

A.2.3 Complete rigid motion

The complete rigid transformation, combining rotation and translation as in (156), in homogeneous coordinates is then obtained as

$$\underline{\boldsymbol{X}}'_i = \mathbf{M}_\mathrm{T} \cdot \mathbf{M}_\mathrm{R} \cdot \underline{\boldsymbol{X}}_i \tag{167}$$

which, by inserting from Eqn. (164) and Eqn. (166), expands to

$$\begin{pmatrix} X'_i \\ Y'_i \\ Z'_i \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{M}_\mathrm{T}} \cdot \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{M}_\mathrm{R}} \cdot \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{pmatrix}. \tag{168}$$

Thus, using homogeneous coordinates, the complete rigid body (rb) motion can be expressed as a single matrix multiplication

$$\underline{\boldsymbol{X}}'_i = \mathbf{M}_\mathrm{rb} \cdot \underline{\boldsymbol{X}}_i, \tag{169}$$

where

$$\mathbf{M}_{\mathrm{rb}} = \mathbf{M}_{\mathrm{T}} \cdot \mathbf{M}_{\mathrm{R}} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & \boldsymbol{t} \\ \mathbf{0}_3^{\mathsf{T}} & 1 \end{pmatrix}. \tag{170}$$

Due to the use of homogenous coordinates, rigid body motion can be expressed as a linear transformation.[27] The transformation in Eqn. (170) has 12 variables but only 6 degrees of freedom: 3 for the rotations $\theta_x, \theta_y, \theta_z$ contained in $\mathbf{R}$, and t_x, t_y, t_z in the translation vector $\boldsymbol{t}$.

A.2.4 Converting rotations

The usual method for parameterizing the rotation matrix with 3 scalar quantities is based on the Euler-Rodrigues representation (see Sec. 3.6.2). Conversion between rotation matrices and Rodrigues rotation vectors is usually accomplished via the *quaternion* representation. This also applies to the *Apache Commons Math* library, which is used in the actual Java implementation. The following "recipe" is provided for completeness and was adapted from the description in [11]. For a concise summary see the corresponding procedures in Alg. 4.10.

Rotation matrix to Rodrigues vector. Given a 3×3 rotation matrix[28] $\mathbf{R} = (R_{i,j})$, the corresponding Rodrigues vector $\boldsymbol{\rho}$ can be calculated as follows [11]. First we define the quantities

$$\boldsymbol{p} = \frac{1}{2} \cdot \begin{pmatrix} R_{2,1} - R_{2,1} \\ R_{0,2} - R_{2,0} \\ R_{1,0} - R_{0,1} \end{pmatrix}, \qquad c = \frac{\mathrm{trace}(\mathbf{R}) - 1}{2} \tag{171}$$

and differentiate the following three cases:

Case 1: If $\|\boldsymbol{p}\| = 0$ and $c = 1$, then $\boldsymbol{\rho} = \mathbf{0}$.

Case 2: If $\|\boldsymbol{p}\| = 0$ and $c = -1$, let $\boldsymbol{v}$ be the *maximum* (non-zero) norm column vector of the matrix $\mathbf{R}_+ = \mathbf{R} + \mathbf{I}$, then

$$\boldsymbol{u} = \frac{1}{\|\boldsymbol{v}\|} \cdot \boldsymbol{v} \qquad \text{and} \qquad \boldsymbol{\rho} = \pi \cdot S(\boldsymbol{u}), \tag{172}$$

with $S(\,)$ as defined in Eqn. (174) below.

Case 3: Otherwise (if $\|\boldsymbol{p}\| \neq 0$),

$$\boldsymbol{u} = \frac{1}{\|\boldsymbol{p}\|} \cdot \boldsymbol{p}, \qquad \theta = \tan^{-1}\left(\frac{\|\boldsymbol{p}\|}{c}\right), \qquad \text{and} \qquad \boldsymbol{\rho} = \theta \cdot \boldsymbol{u}. \tag{173}$$

To avoid singularities, $\tan^{-1}(\|\boldsymbol{p}\|/c)$ in Eqn. (173) should be calculated with the standard Java method `Math.atan2(`$\|\boldsymbol{p}\|, c$`)` (or equivalent). The function

[27] Translation by itself is not linear transformation in non-homogenous coordinates.

[28] A proper rotation $\mathbf{R}$ must satisfy $\mathbf{R}^{\mathsf{T}} \cdot \mathbf{R} = \mathbf{R} \cdot \mathbf{R}^{\mathsf{T}} = \mathbf{I}$ and $\det(\mathbf{R}) = 1$.

$S(\boldsymbol{x})$, used in Eqn. (172), conditionally flips the signs of the coordinates of the supplied 3D vector $\boldsymbol{x} = (x_0, x_1, x_2)$:

$$S(\boldsymbol{x}) = \begin{cases} -\boldsymbol{x} & \text{if } (x_0 < 0) \vee (x_0 = 0 \wedge x_1 < 0) \vee (x_0 = x_1 = 0 \wedge x_2 < 0), \\ \boldsymbol{x} & \text{otherwise.} \end{cases} \tag{174}$$

Rodrigues vector to Rotation matrix. Given a 3D rotation specified by the Rodrigues vector $\boldsymbol{\rho} = (\rho_x, \rho_y, \rho_z)$, as defined in Eqn. (126), the corresponding rotation matrix $\mathbf{R}$ can be found as follows. First, we extract the rotation angle θ (with $|\theta| < \pi$) and the unit direction vector $\hat{\boldsymbol{\rho}}$ by

$$\theta = \|\boldsymbol{\rho}\| \qquad \text{and} \qquad \hat{\boldsymbol{\rho}} = (\hat{\rho}_x, \hat{\rho}_y, \hat{\rho}_z) = \frac{\boldsymbol{\rho}}{\|\boldsymbol{\rho}\|}. \tag{175}$$

The associated rotation matrix is defined as

$$\mathbf{R} = \mathbf{I} + \mathbf{W} \cdot \sin(\theta) + \mathbf{W} \cdot \mathbf{W} \cdot (1 - \cos(\theta)), \tag{176}$$

where $\mathbf{I}$ is the 3×3 identity matrix, and

$$\mathbf{W} = \begin{pmatrix} 0 & -\hat{\rho}_z & \hat{\rho}_y \\ \hat{\rho}_z & 0 & -\hat{\rho}_x \\ -\hat{\rho}_y & \hat{\rho}_x & 0 \end{pmatrix} \tag{177}$$

is the *cross-product matrix*.[29] The expanded rotation matrix is[30]

$$\mathbf{R} = \begin{pmatrix} \cos\theta + \hat{\rho}_x^2(1-\cos\theta) & -\hat{\rho}_z \sin\theta + \hat{\rho}_x\hat{\rho}_y(1-\cos\theta) & \hat{\rho}_y \sin\theta + \hat{\rho}_x\hat{\rho}_z(1-\cos\theta) \\ \hat{\rho}_z \sin\theta + \hat{\rho}_x\hat{\rho}_y(1-\cos\theta) & \cos\theta + \hat{\rho}_y^2(1-\cos\theta) & -\hat{\rho}_x \sin\theta + \hat{\rho}_y\hat{\rho}_z(1-\cos\theta) \\ -\hat{\rho}_y \sin\theta + \hat{\rho}_x\hat{\rho}_z(1-\cos\theta) & \hat{\rho}_x \sin\theta + \hat{\rho}_y\hat{\rho}_z(1-\cos\theta) & \cos\theta + \hat{\rho}_z^2(1-\cos\theta) \end{pmatrix}. \tag{178}$$

B Normalizing 2D point sets

In general, normalizing a given 2D point set $\mathtt{X} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{N-1})$ is accomplished by shifting and scaling all points such that the centroid of the transformed set is aligned with the origin and its diameter has a predefined size. The elements of the normalized point set,

$$\mathtt{X}' = \text{normalize}(\mathtt{X}) = (\boldsymbol{x}'_0, \ldots, \boldsymbol{x}'_{N-1}) \tag{179}$$

are obtained as $\boldsymbol{x}'_j = \mathbf{N}_\mathtt{X} \cdot \boldsymbol{x}_j$, using homogeneous coordinates, i. e.,

$$\boldsymbol{x}'_j = \text{hom}^{-1}\left[\mathbf{N}_\mathtt{X} \cdot \text{hom}(\boldsymbol{x}_j)\right]. \tag{180}$$

$\mathbf{N}_\mathtt{X}$ is a dedicated normalization matrix for the point set $\mathtt{X}$, typically with the structure

$$\mathbf{N}_\mathtt{X} = \begin{pmatrix} s_x & 0 & -s_x\bar{x} \\ 0 & s_y & -s_y\bar{y} \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and the inverse} \quad \mathbf{N}_\mathtt{X}^{-1} = \begin{pmatrix} \frac{1}{s_x} & 0 & \bar{x} \\ 0 & \frac{1}{s_y} & \bar{y} \\ 0 & 0 & 1 \end{pmatrix}, \tag{181}$$

[29] Note that $\mathbf{W} \cdot \boldsymbol{\rho} = \mathbf{0}$ and thus applying $\mathbf{R}_{\boldsymbol{\rho}}$ to an arbitrary point $\lambda \cdot \boldsymbol{\rho}$ on the rotation axis yields exactly the same point.

[30] http://mathworld.wolfram.com/RodriguesRotationFormula.html

where $\bar{\boldsymbol{x}} = (\bar{x}, \bar{y})^\mathsf{T} = \frac{1}{N} \cdot \sum_{j=0}^{N-1} \boldsymbol{x}_j$ is the centroid of the original point set. For calculating the scale factor s, several methods can be found in the literature, two of which are given below.[31]

Method 1: This method scales the point set (uniformly along both axes) such that the average distance of the points from the origin is equal to $\sqrt{2}$.[32] No rotation is applied. In this case,

$$s_x = s_y = \sqrt{2} \cdot N \cdot \Big(\sum_{j=0}^{N-1} \|\boldsymbol{x}_j - \bar{\boldsymbol{x}}\| \Big)^{-1}. \tag{182}$$

Method 2: Here the scaling is applied non-uniformly in x- and y-direction, such that the variances along both axis get normalized, i. e.,

$$s_x = \sqrt{2/\sigma_x^2} \qquad \text{and} \qquad s_y = \sqrt{2/\sigma_y^2}, \tag{183}$$

with the variances $\sigma_x^2 = \frac{1}{N} \sum_{j=0}^{N-1} (x_j - \bar{x})^2$ and $\sigma_y^2 = \frac{1}{N} \sum_{j=0}^{N-1} (y_j - \bar{y})^2$, respectively.

Note that none of the above methods is optimal for point sets with high eccentricity that is not aligned with the coordinate axis. There are probably better methods for normalizing such points sets that also include rotations to align the dominant orientations with coordinate axes, e. g., by principal component analysis (PCA).

C Extraction of camera intrinsics by Cholesky decomposition

In Sec. 3.3, the intrinsic camera parameters $\mathbf{A}$ are obtained from the previously calculated matrix

$$\mathbf{B} = \lambda \cdot (\mathbf{A}^{-1})^\mathsf{T} \cdot \mathbf{A}^{-1} = \underbrace{[\sqrt{\lambda} \cdot (\mathbf{A}^{-1})^\mathsf{T}]}_{\mathbf{L}} \cdot \underbrace{[\sqrt{\lambda} \cdot \mathbf{A}^{-1}]}_{\mathbf{L}^\mathsf{T}} \tag{184}$$

in closed form (see Eqns. (99–105)), where $\lambda \in \mathbb{R}$ is an unknown scale factor. An alternative approach using matrix decomposition is shown here. The inverse of $\mathbf{A}$ has the structure

$$\mathbf{A}^{-1} = \begin{pmatrix} a_0 & a_1 & a_2 \\ 0 & a_3 & a_4 \\ 0 & 0 & 1 \end{pmatrix}, \tag{185}$$

and therefore $\sqrt{\lambda} \cdot (\mathbf{A}^{-1}) = \mathbf{L}^\mathsf{T}$ is an *upper* triangular matrix and $\sqrt{\lambda} \cdot (\mathbf{A}^{-1})^\mathsf{T} = \mathbf{L}$ is a *lower* triangular matrix. Given the matrix $\mathbf{B} = \mathbf{L} \cdot \mathbf{L}^\mathsf{T}$, the *Cholesky decomposition*[33] can be used to determine $\mathbf{L}$ uniquely in the form

$$\mathbf{L} = \mathrm{Chol}(\mathbf{B}), \tag{186}$$

[31] Method 2 is used in the implementation.

[32] The relevance of factor $\sqrt{2}$ is unclear to the author. Replacing it by 1 should not affect the numerical stability.

[33] http://en.wikipedia.org/wiki/Cholesky_decomposition

if $\mathbf{B}$ is a *symmetric*, *positive definite* matrix.[34]

From $\mathbf{L}^\intercal = \sqrt{\lambda} \cdot \mathbf{A}^{-1}$ we see that $\mathbf{L}^\intercal$ is identical to $\mathbf{A}^{-1}$, except for the scale factor $s = \sqrt{\lambda}$. Since the lower-right element of $\mathbf{A}^{-1}$ must be 1, the result is

$$\mathbf{A}^{-1} = \frac{1}{s} \cdot \mathbf{L}^\intercal \qquad \text{and thus} \qquad \mathbf{A} = s \cdot (\mathbf{L}^\intercal)^{-1} = s \cdot (\mathbf{L}^{-1})^\intercal, \tag{187}$$

with $s = \mathbf{L}^\intercal_{2,2} = \mathbf{L}_{2,2}$.

While $\mathbf{B}$ is certainly symmetric, it is not necessarily *positive definite*, since λ may be negative. A simple test for positive definiteness is to check if all diagonal elements are non-negative. If not, we apply the Cholesky decomposition to $-\mathbf{B}$. The following code segment shows how this may be accomplished with the *Apache Commons Math* API:

```
RealMatrix B; // obtained from vector b
if (B.getEntry(0,0)<0 || B.getEntry(1,1)<0 || B.getEntry(2,2)<0) {
    B = B.scalarMultiply(-1); // make sure B is positive definite
}
CholeskyDecomposition cd = new CholeskyDecomposition(B);
RealMatrix L = CD.getL();
RealMatrix A =
  MatrixUtils.inverse(L).transpose().scalarMultiply(L.getEntry(2,2));
```

A more elegant (and safer) approach is to use the Cholesky decomposition itself to verify that $\mathbf{B}$ is positive definite. If not, the constructor throws a `NonPositiveDefiniteMatrixException`, which can be caught to repeat the decomposition on $-\mathbf{B}$, as shown in this example:

```
RealMatrix B;
CholeskyDecomposition cd = null;
try {cd = new CholeskyDecomposition(B);}
catch (NonPositiveDefiniteMatrixException e)
    {cd = new CholeskyDecomposition(B.scalarMultiply(-1));} // try -B
RealMatrix L = cd.getL();
RealMatrix A = ...
```

D Calculating the focal length f

The intrinsic parameters α, β, γ in Eqn. (15) can only be determined up to an unknown scale factor, i. e., the absolute size of the imaging system (and the focal length f in particular) cannot be determined from intrinsic parameters alone. Once the intrinsic parameters

$$\alpha = f s_x, \qquad \beta = f s_y, \qquad \gamma = f s_\theta$$

are known, the focal length f can be obtained by setting one of the scale parameters to a constant value. For example, setting the horizontal scale parameter $s_x = 1$ we get

$$f = \frac{\alpha}{s_x} = \alpha \qquad \text{and thus} \qquad s_y = \frac{\beta}{f} = \frac{\beta}{\alpha}\,. \tag{188}$$

[34] http://en.wikipedia.org/wiki/Positive-definite_matrix

The resulting value f is the focal length measured in terms of horizontal pixel units.

The actual (physical) focal length is then obtained by multiplying f (in pixel units) with the actual horizontal pixel spacing Δ_x, i. e.,

$$f_{real} = f \cdot \Delta_x \,.$$

Δ_x is usually known and constant. For example, given a typical 3×2 sensor chip of size 22.5×15 mm with 3000×2000 (6 million) square pixels, the resulting pixel spacing is

$$\Delta_x = \Delta_y = \frac{15\,\text{mm}}{2000} = 0.0075\,\text{mm}$$

Assuming $s_x = s_y = 1$ and $\alpha = f = 3200$, the resulting physical focal length is $f_{real} = 3200 \cdot 0.0075 = 24\,\text{mm}$.

E Non-linear optimization with the Levenberg-Marquart method

For easier understanding, it may be helpful to introduce some common terminology used in optimization.The general problem can be summarized as follows:

> Given is a set of m empirical "observations" $\{\langle \boldsymbol{x}_i, \dot{y}_i \rangle\}$, each composed of an input coordinate $\boldsymbol{x}_i \in \mathbb{R}^p$ and an associated (measured) scalar output value $\dot{y}_i \in \mathbb{R}$. It is assumed that the relation between the $\boldsymbol{x}_i$ and $\dot{y}_i$ values can be modeled by a function f (or even multiple functions f_i) with a common set of parameters $\boldsymbol{p} \in \mathbb{R}^n$. The goal is to find the parameters that give the best fit between the observed values $\dot{y}_i$ and the output values $y_i = f(\boldsymbol{x}_i, \boldsymbol{p})$ "predicted" by the model function(s).

Note that the set of observations may contain multiple measurements for the *same* input position $\boldsymbol{x}$. In the remaining part of this section, the following symbols are used:[35]

m ... the number of empirical observations (index i),
n ... dimensionality of the parameter vector $\boldsymbol{p}$ (index j),
p ... dimensionality of the input coordinates $\boldsymbol{x}_i$,
q ... dimensionality of the output values $\boldsymbol{y}_i$ (see Sec. E.1.6).

E.1 Setting up the "model"

E.1.1 Special case: single model function

To keep things simple, we start with the special case that the relation between the input coordinates $\boldsymbol{x}_i$ and the observed values $\dot{y}_i$ is modeled by a single

[35] To keep this description as general as possible, the notation used in this chapter is intentionally different (though similar) to the notation used in the remaining parts of the document. Thus the symbols used here are not directly related to the symbols used to describe camera calibration. In particular, the symbols $\boldsymbol{x}$ and y do *not* denote 2D image coordinates but have different meanings in this context!

function $f()$, that is,

$$\dot{y}_i \approx f(\boldsymbol{x}_i, \boldsymbol{p}), \tag{189}$$

for all observations $i = 0, \ldots, m-1$, with the n-dimensional parameter vector $\boldsymbol{p}$. The "predicted" output value, obtained by applying the model function $f()$ to the input vector $\boldsymbol{x}_i$, is denoted

$$y_i = f(\boldsymbol{x}_i, \boldsymbol{p}). \tag{190}$$

In general, the predicted value y_i is not the same as the observed value $\dot{y}_i$, i. e., $y_i \neq \dot{y}_i$. For a given parameter vector $\boldsymbol{p}$, the deviation (error) between the associated predicted and observed output values is commonly quantified as

$$e_i(\boldsymbol{p}) = |\dot{y}_i - y_i|^2 = |\dot{y}_i - f(\boldsymbol{x}_i, \boldsymbol{p})|^2, \tag{191}$$

and the goal is to find the parameters $\boldsymbol{p}$ that *minimize* the overall "least-squares" error

$$E(\boldsymbol{p}) = \sum_{i=0}^{m-1} e_i(\boldsymbol{p}) = \sum_{i=0}^{m-1} |\dot{y}_i - y_i|^2 = \sum_{i=0}^{m-1} |\dot{y}_i - f(\boldsymbol{x}_i, \boldsymbol{p})|^2, \tag{192}$$

over all m observations $\langle \boldsymbol{x}_i, \dot{y}_i \rangle$. To simplify the notation, we can represent the set of observations by two vectors $\mathtt{X}, \dot{\mathtt{Y}}$, where

$$\mathtt{X} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{m-1})^\intercal \tag{193}$$

is the vector of sample *positions* and

$$\dot{\mathtt{Y}} = (\dot{y}_0, \ldots, \dot{y}_{m-1})^\intercal. \tag{194}$$

is the vector of the associated sample *values*.[36] The "least-squares" error (Eqn. (192)) to be minimzied can now be conveniently written in the form

$$E(\boldsymbol{p}) = \|\dot{\mathtt{Y}} - \mathtt{Y}\|^2 = \|\dot{\mathtt{Y}} - F(\mathtt{X}, \boldsymbol{p})\|^2, \tag{195}$$

where the vector of "predicted" values $\mathtt{Y}$ is obtained by applying the model function $f(\boldsymbol{x}, \boldsymbol{p})$ to all m sample positions $\boldsymbol{x}_i \in \mathtt{X}$, that is,

$$\mathtt{Y} = F(\mathtt{X}, \boldsymbol{p}) = \begin{pmatrix} y_0 \\ \vdots \\ y_{m-1} \end{pmatrix} = \begin{pmatrix} f(\boldsymbol{x}_0, \boldsymbol{p}) \\ \vdots \\ f(\boldsymbol{x}_{m-1}, \boldsymbol{p}) \end{pmatrix}. \tag{196}$$

E.1.2 General case: multiple model functions

In general, the underlying model function $f()$ needs not be the same for every sample (as in Eqn. (196)). Instead, a *different* model function $f_i(\boldsymbol{x}, \boldsymbol{p})$ may be assigned to each sample, as long as every model function accepts the same parameter vector $\boldsymbol{p}$. We can thus rewrite Eqn. (196) in the general form

$$F(\mathtt{X}, \boldsymbol{p}) = \mathtt{Y} = \begin{pmatrix} y_0 \\ \vdots \\ y_{m-1} \end{pmatrix} = \begin{pmatrix} f_0(\boldsymbol{x}_0, \boldsymbol{p}) \\ \vdots \\ f_{m-1}(\boldsymbol{x}_{m-1}, \boldsymbol{p}) \end{pmatrix}. \tag{197}$$

[36] Note that $\mathtt{X}$ is a vector of vectors, i. e., actually a matrix.

In optimization language, the function $F(\mathtt{X}, \boldsymbol{p})$ in Eqn. (197) is called the "value function". It evaluates the "model" for all sample positions $\boldsymbol{x}_i \in \mathtt{X}$ with the common parameters $\boldsymbol{p}$ and returns the results as a vector $\mathtt{Y} = (y_0, \ldots, y_{m-1})^\mathsf{T}$, with $y_i = f_i(\boldsymbol{x}_i, \boldsymbol{p})$.

E.1.3 The Jacobian function $J()$

In addition to the value function $F()$, the Levenberg-Marquart method requires the *Jacobian* of the involved model functions. The function $J(\mathtt{X}, \boldsymbol{p})$, defined below, returns the $m \times n$ Jacobian matrix[37] of the multi-dimensional, scalar-valued function F for the (fixed) input coordinates $\mathtt{X} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{m-1})$ and the variable parameters $\boldsymbol{p} \in \mathbb{R}^n$. With the input position $\boldsymbol{x}_i$ fixed, each component function $f_i(\boldsymbol{x}_i, \boldsymbol{p})$ only depends on the n-dimensional parameter vector $\boldsymbol{p}$, and can thus be treated as a n-dimensional function. For a given input $\mathtt{X}$, the Jacobian function at a specific "point" (parameter vector) $\boldsymbol{p}$ is

$$J(\mathtt{X}, \boldsymbol{p}) = \mathtt{J} = \begin{pmatrix} \frac{\partial f_0(\boldsymbol{x}_0, \boldsymbol{p})}{\partial p_0} & \frac{\partial f_0(\boldsymbol{x}_0, \boldsymbol{p})}{\partial p_1} & \cdots & \frac{\partial f_0(\boldsymbol{x}_0, \boldsymbol{p})}{\partial p_{n-1}} \\ \frac{\partial f_1(\boldsymbol{x}_1, \boldsymbol{p})}{\partial p_0} & \frac{\partial f_1(\boldsymbol{x}_1, \boldsymbol{p})}{\partial p_1} & \cdots & \frac{\partial f_1(\boldsymbol{x}_1, \boldsymbol{p})}{\partial p_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{m-1}(\boldsymbol{x}_{m-1}, \boldsymbol{p})}{\partial p_0} & \frac{\partial f_{m-1}(\boldsymbol{x}_{m-1}, \boldsymbol{p})}{\partial p_1} & \cdots & \frac{\partial f_{m-1}(\boldsymbol{x}_{m-1}, \boldsymbol{p})}{\partial p_{n-1}} \end{pmatrix}. \tag{198}$$

The element (i, j) of the Jacobian is the first partial derivative of the component function $f_i(\boldsymbol{x}, \boldsymbol{p})$ with respect to the single parameter p_j. This scalar quantity indicates, how much the output value of the component function $f_i()$ increases (or decreases) at the specified position $\boldsymbol{x} = \boldsymbol{x}_i$ and the parameters settings $\boldsymbol{p}$, when only the single parameter p_j is modified and everything else remains fixed. This information is essential for the LM optimizer to efficiently explore the parameter space in search of the optimal solution.

E.1.4 Calculation of partial derivatives

Depending on the characteristics of the model functions in F, the partial derivatives of the Jacobian may be expressible as analytical functions, which is the ideal situation. If this is too complicated or impossible, the partial derivatives can be estimated *numerically* by applying a minor variation (δ) to the relevant parameter (p_j) and measuring the resulting change of the associated function output.

Numeric calculation. It is also possible to estimate the partial derivatives numerically by finite difference approximation in the form

$$\frac{\partial f_i(\boldsymbol{x}, \boldsymbol{p})}{\partial p_j} \approx \frac{f_i(\boldsymbol{x}, \boldsymbol{p} + \delta_j \cdot \mathbf{e}_j) - f_k(\boldsymbol{x}, \boldsymbol{p})}{\delta}, \tag{199}$$

which varies the single parameter p_j by a small amount δ_j and measures the resulting change of the component function $f_i()$ at the given position $\boldsymbol{x}$. Here $\mathbf{e}_j$ denotes a unit vector with value 1 at position j and all other elements zero.

[37] The elements of a function's Jacobian matrix are again functions.

$\delta_j \in \mathbb{R}$ is a small, positive quantity of change (typ. $\approx 10^{-8}$) applied to parameter p_j. For numerical reasons, δ_j is usually not set to a fixed value but is adapted to the magnitude of the affected parameter p_j in the form

$$\delta_j = \sqrt{\epsilon} \cdot \max(|p_j|, 1)\,, \tag{200}$$

with the constant $\epsilon = 2.2^{-16}$ (and thus $\sqrt{\epsilon} \approx 1.5 \cdot 10^{-8}$), assuming `double` arithmetic.[38]

Note that in Eqn. (199) all other parameters ($p_k, k \neq j$) as well as the coordinate vector $\boldsymbol{x}$ remain unchanged.

E.1.5 Invoking the LM optimization

With $\mathtt{X}$, $\dot{\mathtt{Y}}$, F, and J set up, as described above, the LM optimization procedure can be invoked the form

$$\boldsymbol{p}_{\text{opt}} \leftarrow \text{Optimize}(F, J, \mathtt{X}, \dot{\mathtt{Y}}, \boldsymbol{p}_0), \tag{201}$$

where $\boldsymbol{p}_0$ is the initial parameter vector or "starting point". The actual work is done by the LM optimizer, usually implemented in numerical libraries with a similar call signature. The result $\boldsymbol{p}_{\text{opt}}$ is a parameter vector that minimizes the least-squares deviation between the values predicted by the model (represented by F) and the observed values in $\dot{\mathtt{Y}}$, for the set of input coordinates given in $\mathtt{X}$. Note that the result is a *local* optimum, i. e., much depends on $\boldsymbol{p}_0$ being a good "initial guess'.

E.1.6 Handling multi-dimensional output values

While the formulation in Eqn. (197) applies to scalar output values (y_i), it can be used directly to handle multi-dimensional (i. e., vector-valued) output data as well. Let us assume that our empirical observations $\{\langle \boldsymbol{x}_i, \dot{\boldsymbol{y}}_i \rangle\}$ contain vector-valued input data $\boldsymbol{x}_i \in \mathbb{R}^p$ (as before) as well as vector-valued output data $\dot{\boldsymbol{y}}_i = (\dot{y}_{i,0}, \ldots, \dot{y}_{i,q-1}) \in \mathbb{R}^q$. In this case the associated model function is multi-dimensional and vector-valued, i. e.,

$$\boldsymbol{f}_i \colon \mathbb{R}^p \mapsto \mathbb{R}^q. \tag{202}$$

The trick is to split this function into q scalar-valued component functions $f_{i,0}, \ldots, f_{i,q-1}$ (which is always possible), such that

$$\boldsymbol{y}_i = \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{p}) = \begin{pmatrix} y_{i,0} \\ \vdots \\ y_{i,q-1} \end{pmatrix} = \begin{pmatrix} f_{i,0}(\boldsymbol{x}_i, \boldsymbol{p}) \\ \vdots \\ f_{i,q-1}(\boldsymbol{x}_i, \boldsymbol{p}) \end{pmatrix}. \tag{203}$$

Each pair $\langle \boldsymbol{x}_i, \dot{y}_{i,j} \rangle$ can now be treated as a single scalar-valued observation and $y_{i,j} \in \mathbb{R}$ as the associated prediction value. The position and value vectors (corresponding to Eqn. (193) and Eqn. (194), respectively), are expanded to

$$\mathtt{X} = (\underbrace{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_0}_{q \text{ times}}, \underbrace{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_1}_{q \text{ times}}, \ldots, \underbrace{\boldsymbol{x}_{m-1}, \ldots, \boldsymbol{x}_{m-1}}_{q \text{ times}})^{\mathsf{T}}, \tag{204}$$

$$\dot{\mathtt{Y}} = (\underbrace{\dot{y}_{0,0}, \ldots, \dot{y}_{0,q-1}}_{\dot{\boldsymbol{y}}_0}, \underbrace{\dot{y}_{1,0}, \ldots, \dot{y}_{1,q-1}}_{\dot{\boldsymbol{y}}_1}, \ldots, \underbrace{\dot{y}_{m-1,0}, \ldots, \dot{y}_{m-1,q-1}}_{\dot{\boldsymbol{y}}_{m-1}})^{\mathsf{T}}, \tag{205}$$

[38] See [10, Sec. 5.7] and http://en.wikipedia.org/wiki/Numerical_differentiation for details.

and the vector of "predicted" values (see Eqn. (197)) becomes

$$\mathtt{Y} = (\underbrace{y_{0,0}, \ldots, y_{0,q-1}}_{\boldsymbol{y}_0 = \boldsymbol{f}_0(\boldsymbol{x}_0, \boldsymbol{p})}, \underbrace{y_{1,0}, \ldots, y_{1,q-1}}_{\boldsymbol{y}_1 = \boldsymbol{f}_1(\boldsymbol{x}_1, \boldsymbol{p})}, \ldots, \underbrace{y_{m-1,0}, \ldots, y_{m-1,q-1}}_{\boldsymbol{y}_{m-1} = \boldsymbol{f}_{m-1}(\boldsymbol{x}_{m-1}, \boldsymbol{p})})^\mathsf{T}. \tag{206}$$

The approximation error is calculated in exactly the same way as defined for scalar-valued outputs (see Eqn. (195)), that is,

$$E(\boldsymbol{p}) = \sum_{i=0}^{m-1} \|\dot{\boldsymbol{y}}_i - \boldsymbol{y}_i\|^2 = \sum_{i=0}^{m-1} \sum_{j=0}^{q-1} |\dot{y}_{i,j} - y_{i,j}|^2 = \|\dot{\mathcal{Y}} - \mathtt{Y}\|^2 = \|\dot{\mathtt{Y}} - F(\mathtt{X}, \boldsymbol{p})\|^2. \tag{207}$$

By using a scalar-valued component function for each output dimension, mappings to output values of any dimensionality can be handled. The *length* of the resulting data vectors is $m \cdot q$ (the number of observations multiplied by the dimensionality of the output values).

Example: Assume we wish to optimize a vector-valued model transformation $\boldsymbol{f}\colon \mathbb{R}^3 \mapsto \mathbb{R}^2$ (i. e., $p = 3$, $q = 2$), which maps from 3D input positions $\boldsymbol{x}_i$ to 2D output values $\boldsymbol{y}_i = (y_{i,\mathrm{x}}, y_{i,\mathrm{y}})^\mathsf{T}$, such that

$$\boldsymbol{y}_i = \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{p}) \qquad \text{or} \qquad \begin{pmatrix} y_{i,\mathrm{x}} \\ y_{i,\mathrm{y}} \end{pmatrix} = \begin{pmatrix} f_{i,\mathrm{x}}(\boldsymbol{x}_i, \boldsymbol{p}) \\ f_{i,\mathrm{y}}(\boldsymbol{x}_i, \boldsymbol{p}) \end{pmatrix} = \begin{pmatrix} f_{\mathrm{x}}(\boldsymbol{x}_i, \boldsymbol{p}) \\ f_{\mathrm{y}}(\boldsymbol{x}_i, \boldsymbol{p}) \end{pmatrix}. \tag{208}$$

The two scalar-valued component functions are assumed to be the same for all samples, i. e., $f_{i,\mathrm{x}} = f_{\mathrm{x}}$ and $f_{i,\mathrm{y}} = f_{\mathrm{y}}$, for all $i = 0, \ldots, n-1$. Given m empirical observations $\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle$ for this mapping, with $\boldsymbol{y}_i = (y_{i,\mathrm{x}}, y_{i,\mathrm{y}})^\mathsf{T}$, the resulting data vectors could be arranged as

$$\mathtt{X} = \begin{pmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{m-1} \\ \boldsymbol{x}_{m-1} \end{pmatrix}, \quad \dot{\mathtt{Y}} = \begin{pmatrix} \dot{y}_{0,\mathrm{x}} \\ \dot{y}_{0,\mathrm{y}} \\ \dot{y}_{1,\mathrm{x}} \\ \dot{y}_{1,\mathrm{y}} \\ \vdots \\ \dot{y}_{m-1,\mathrm{x}} \\ \dot{y}_{m-1,\mathrm{y}} \end{pmatrix}, \quad \mathtt{Y} = \begin{pmatrix} y_{0,\mathrm{x}} \\ y_{0,\mathrm{y}} \\ y_{1,\mathrm{x}} \\ y_{1,\mathrm{y}} \\ \vdots \\ y_{m-1,\mathrm{x}} \\ y_{m-1,\mathrm{y}} \end{pmatrix} = \begin{pmatrix} f_{\mathrm{x}}(\boldsymbol{x}_0, \boldsymbol{p}) \\ f_{\mathrm{y}}(\boldsymbol{x}_0, \boldsymbol{p}) \\ f_{\mathrm{x}}(\boldsymbol{x}_1, \boldsymbol{p}) \\ f_{\mathrm{y}}(\boldsymbol{x}_1, \boldsymbol{p}) \\ \vdots \\ f_{\mathrm{x}}(\boldsymbol{x}_{m-1}, \boldsymbol{p}) \\ f_{\mathrm{y}}(\boldsymbol{x}_{m-1}, \boldsymbol{p}) \end{pmatrix}. \tag{209}$$

Note that the *ordering* of the vector elements in Eqn. (209) is not important; e. g., we could have arranged them equivalently in the form

$$\mathtt{X} = \begin{pmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{m-1} \\ \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{m-1} \end{pmatrix}, \quad \dot{\mathtt{Y}} = \begin{pmatrix} \dot{y}_{0,\mathrm{x}} \\ \dot{y}_{1,\mathrm{x}} \\ \vdots \\ \dot{y}_{m-1,\mathrm{x}} \\ \dot{y}_{0,\mathrm{x}} \\ \dot{y}_{1,\mathrm{x}} \\ \vdots \\ \dot{y}_{m-1,\mathrm{y}} \end{pmatrix}, \quad \mathtt{Y} = \begin{pmatrix} y_{0,\mathrm{x}} \\ y_{1,\mathrm{x}} \\ \vdots \\ y_{m-1,\mathrm{x}} \\ y_{0,\mathrm{y}} \\ y_{1,\mathrm{y}} \\ \vdots \\ y_{m-1,\mathrm{y}} \end{pmatrix} = \begin{pmatrix} f_{\mathrm{x}}(\boldsymbol{x}_0, \boldsymbol{p}) \\ f_{\mathrm{x}}(\boldsymbol{x}_1, \boldsymbol{p}) \\ \vdots \\ f_{\mathrm{x}}(\boldsymbol{x}_{m-1}, \boldsymbol{p}) \\ f_{\mathrm{y}}(\boldsymbol{x}_0, \boldsymbol{p}) \\ f_{\mathrm{y}}(\boldsymbol{x}_1, \boldsymbol{p}) \\ \vdots \\ f_{\mathrm{y}}(\boldsymbol{x}_{m-1}, \boldsymbol{p}) \end{pmatrix}. \tag{210}$$

While only two component functions $(f_{\mathrm{x}}, f_{\mathrm{y}})$ were used in this example, there could (as mentioned earlier) be a different function in each line of Eqn. (209) or Eqn. (210).

References

[1] I. N. Bronstein, K. A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, fifth edition, 2000.

[2] W. Burger and M. J. Burge. *Digital Image Processing – An Algorithmic Introduction Using Java*. Springer, London, second edition, 2016.

[3] D. Claus and A. W. Fitzgibbon. A rational function lens distortion model for general cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 213–219, June 2005.

[4] W. W. Hager. *Applied Numerical Linear Algebra*. Prentice Hall, 1988.

[5] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2013.

[6] R. I. Hartley. In defense of the eight-point algorithm. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(6):580–593, 1997.

[7] Intel Corporation. *Open Source Computer Vision Library reference Manual*, 2001. http://developer.intel.com.

[8] R. Jain, R. Kasturi, and B. G. Schunck. *Machine Vision*. McGraw-Hill, 1995.

[9] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer, 2004.

[10] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes*. Cambridge University Press, third edition, 2007.

[11] C. Tomasi. Vector representation of rotations. Computer Science 527 Course Notes, Duke University, https://www.cs.duke.edu/courses/fall13/compsci527/notes/rodrigues.pdf, 2013.

[12] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, 1998.

[13] J. Vince. *Matrix Transforms for Computer Games and Animation*. Springer, 2012.

[14] Z. Zhang. A flexible new technique for camera calibration. Technical Report MSR-TR-98-71, Microsoft Research, 1998.

[15] Z. Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.