J Math Imaging Vis
DOI 10.1007/s10851-017-0756-y

Algorithms for the Orthographic-*n*-Point Problem

Carsten Steger[1]

Received: 15 July 2016 / Accepted: 9 August 2017

Abstract We examine the orthographic-*n*-point problem (O*n*P), which extends the perspective-*n*-point problem to telecentric cameras. Given a set of 3D points and their corresponding 2D points under orthographic projection, the O*n*P problem is the determination of the pose of the 3D point cloud with respect to the telecentric camera. We show that the O*n*P problem is equivalent to the unbalanced orthogonal Procrustes problem for non-coplanar 3D points and to the sub-Stiefel Procrustes problem for coplanar 3D points. To solve the O*n*P problem, we apply existing algorithms for the respective Procrustes problems and also propose novel algorithms. Furthermore, we evaluate the algorithms to determine their robustness and speed and conclude which algorithms are preferable in real applications. Finally, we evaluate which algorithm is most suitable as a minimal solver in a RANSAC scheme.

Keywords Orthographic-*n*-point problem · Pose · Exterior orientation · Telecentric lens · Minimal solver · Procrustes problems

1 Introduction

The perspective *n*-point problem (P*n*P) determines the pose in the camera coordinate system, i.e., a rigid 3D transformation, of a set of *n* points from their corresponding images obtained with a perspective camera. At least three point correspondences are required to obtain a finite number of solutions. The P*n*P problem has been researched extensively for perspective cameras. Numerous approaches have been proposed for three (P3P), four (P4P), five (P5P), and a general number of point correspondences (P*n*P); see, e.g., [15,19,22,28,40,70] (P3P), [34,59,67,70] (P4P), [59,67,70] (P5P), [2,17,18,23,27,32,39,48,49,56,61,72] (P*n*P), and references therein. The P*n*P problem originated in photogrammetry, where it is called space resection (or simply resection); see, e.g., [52, Chapter 11.1.3], [51, Chapter 4.2.3], [21, Chapter 12.2.4].

If a camera with a telecentric lens is used, the projection is orthographic instead of perspective [64, Chapter 3.9], [63]. If we want to determine the pose of an object with a telecentric camera, this leads to a problem that we will call the orthographic *n*-point problem (O*n*P). This problem occurs in several applications. For example, it can be used to determine the initial values of the pose of a calibration object during camera calibration. Furthermore, some visual inspection algorithms require that the pose of an object is determined using a telecentric camera (e.g., to determine whether some components that have been mounted onto a printed circuit board are in the correct 3D orientation after reflow). In addition, it can be used to determine the pose of an object in object localization or recognition algorithms.

All of the above P*n*P algorithms assume a perspective camera. Consequently, none of them can solve the O*n*P problem. The approaches that might appear to come closest are the P*n*P algorithms described in [16] for non-coplanar 3D points and [55] for coplanar 3D points. They use a scaled orthographic projection (also known as weak perspective). An extension to paraperspective is described in [35]. In all of these approaches, the affine camera model that is used serves as an approximation to the true perspective projection of the

The original version of this article was revised: The references to step numbers in some of the algorithms were corrected.

✉ Carsten Steger
steger@mvtec.com

[1] MVTec Software GmbH, Arnulfstraße 205, 80634 München, Germany

Published online: 30 August 2017

camera. Since a finite projection center is still required by these algorithms, they cannot be extended to the OnP problem. For example, the similar triangles that are used in the derivation of the algorithms in [16,55] do not exist for true orthographic projection. Furthermore, orthogonality of the 3D rotation matrix is only enforced after the algorithms in [16,35,55] have computed their solutions, which may lead to suboptimal solutions.

In principle, telecentric cameras also can be regarded as specialized instances of generalized cameras, in which the camera geometry is modeled by providing an explicit ray or line in 3D for every image point. For telecentric cameras, all rays are, of course, parallel. Several PnP algorithms have been proposed for generalized cameras [11,54,62]. However, in all of the approaches the case of parallel rays is excluded explicitly [11,54] or implicitly [62]. Therefore, they do not provide a solution to the OnP problem.

Since the OnP problem seems to be largely unexplored, we propose several algorithms to solve the OnP problem in this paper. Some of the proposed algorithms are based on existing Procrustes problem solvers, while others are novel. In addition, we will perform an extensive performance evaluation of the proposed algorithms to determine which algorithms exhibit the best tradeoff between robustness and speed and are, therefore, suitable for practical use.

2 Problem Definition

2.1 Camera Model for Telecentric Cameras

To be able to define the OnP problem, we first discuss the camera model for telecentric cameras that we will use. Our presentation is based on the description in [63].[1]

In our model, a point $\boldsymbol{p}_\mathrm{o} = (x_\mathrm{o}, y_\mathrm{o}, z_\mathrm{o})^\top$ given in the object coordinate system is transformed into a point $\boldsymbol{p}_\mathrm{c} = (x_\mathrm{c}, y_\mathrm{c}, z_\mathrm{c})^\top$ in the camera coordinate system by a rigid 3D transformation:

$$\boldsymbol{p}_\mathrm{c} = \mathbf{R}\boldsymbol{p}_\mathrm{o} + \boldsymbol{t}, \tag{1}$$

where $\boldsymbol{t} = (t_x, t_y, t_z)^\top$ is a translation vector and $\mathbf{R}$ is a rotation matrix. To solve the OnP problem, we must determine $\mathbf{R}$ and $\boldsymbol{t}$.

Next, the point $\boldsymbol{p}_\mathrm{c}$ is projected into the image plane. For telecentric lenses, the projection is given by:

$$\begin{pmatrix} x_\mathrm{u} \\ y_\mathrm{u} \end{pmatrix} = m \begin{pmatrix} x_\mathrm{c} \\ y_\mathrm{c} \end{pmatrix}, \tag{2}$$

[1] We will not discuss the object-side and bilateral telecentric tilt lens camera models that are described in [63]. The essential feature that we will use, i.e., that we can transform pixel coordinates to rectified metric coordinates, can also be performed with these camera models.

where m is the magnification of the lens. Note that (2) is independent of z_c and therefore also of t_z. Consequently, we obviously cannot recover t_z. Since t_z does not influence the projection, we can set it to an arbitrary value, e.g., to 0. The independence of z_c additionally shows that only the first two rows of $\mathbf{R}$ influence the projected points. In contrast to $\boldsymbol{t}$, $\mathbf{R}$ is determined uniquely from its first two rows: since $\mathbf{R}$ is orthogonal, the third row of $\mathbf{R}$ can be reconstructed as the vector product of the first two rows.

The undistorted point $(x_\mathrm{u}, y_\mathrm{u})^\top$ is then distorted to a point $(x_\mathrm{d}, y_\mathrm{d})^\top$. We support two distortion models: the division model [6,20,43–47] and the polynomial model [8,9]. In the division model, the undistorted point $(x_\mathrm{u}, y_\mathrm{u})^\top$ is computed from the distorted point $(x_\mathrm{d}, y_\mathrm{d})^\top$ as follows:

$$\begin{pmatrix} x_\mathrm{u} \\ y_\mathrm{u} \end{pmatrix} = \frac{1}{1 + \kappa r_\mathrm{d}^2} \begin{pmatrix} x_\mathrm{d} \\ y_\mathrm{d} \end{pmatrix}, \tag{3}$$

where $r_\mathrm{d}^2 = x_\mathrm{d}^2 + y_\mathrm{d}^2$. In the polynomial model, the undistorted point is computed by:

$$\begin{pmatrix} x_\mathrm{u} \\ y_\mathrm{u} \end{pmatrix} = \begin{pmatrix} x_\mathrm{d}(1 + K_1 r_\mathrm{d}^2 + K_2 r_\mathrm{d}^4 + K_3 r_\mathrm{d}^6) \\ \quad + (P_1(r_\mathrm{d}^2 + 2x_\mathrm{d}^2) + 2P_2 x_\mathrm{d} y_\mathrm{d}) \\ y_\mathrm{d}(1 + K_1 r_\mathrm{d}^2 + K_2 r_\mathrm{d}^4 + K_3 r_\mathrm{d}^6) \\ \quad + (2P_1 x_\mathrm{d} y_\mathrm{d} + P_2(r_\mathrm{d}^2 + 2y_\mathrm{d}^2)) \end{pmatrix}. \tag{4}$$

The distortion in the division model can be inverted analytically, while that of the polynomial model cannot. As we will see below, in this paper we are only interested in undistorting points. Therefore, the analytical undistortions in (3) and (4) are exactly what we need.

Finally, the distorted point $(x_\mathrm{d}, y_\mathrm{d})^\top$ is transformed into the image coordinate system:

$$\begin{pmatrix} x_\mathrm{i} \\ y_\mathrm{i} \end{pmatrix} = \begin{pmatrix} \frac{x_\mathrm{d}}{s_x} + c_x \\ \frac{y_\mathrm{d}}{s_y} + c_y \end{pmatrix}. \tag{5}$$

Here, s_x and s_y denote the pixel pitch on the sensor in the horizontal and vertical direction, respectively, and $(c_x, c_y)^\top$ is the principal point.

In this paper, we assume that the interior orientation of the camera has been calibrated, e.g., using the approach in [63], i.e., that m, κ or $(K_1, K_2, K_3, P_1, P_2)$, s_x, s_y, c_x, and c_y are known. With this, it is possible to transform an image point $(x_\mathrm{i}, y_\mathrm{i})^\top$ back into a 2D point $(x_\mathrm{c}, y_\mathrm{c})^\top$ in the camera coordinate system. First, we invert (5):

$$\begin{pmatrix} x_\mathrm{d} \\ y_\mathrm{d} \end{pmatrix} = \begin{pmatrix} s_x(x_\mathrm{i} - c_x) \\ s_y(y_\mathrm{i} - c_y) \end{pmatrix}. \tag{6}$$

Next, we apply (3) or (4). Finally, we invert (2):

$$\begin{pmatrix} x_\mathrm{c} \\ y_\mathrm{c} \end{pmatrix} = \frac{1}{m} \begin{pmatrix} x_\mathrm{u} \\ y_\mathrm{u} \end{pmatrix}. \tag{7}$$

The point $(x_\mathrm{c}, y_\mathrm{c})^\top$ is obviously given in the same units, e.g., meters, as the point $\boldsymbol{p}_\mathrm{o}$. To distinguish this projected 2D point from the 3D point $\boldsymbol{p}_\mathrm{c}$, we define $\boldsymbol{p}_\mathrm{p} = (x_\mathrm{c}, y_\mathrm{c})^\top$.

2.2 Procrustes Problems

From the above discussion, we can see that the defining equation for the OnP problem for a single point correspondence is

$$\boldsymbol{p}_\mathrm{p} = \mathbf{R}_{2\times 3}\boldsymbol{p}_\mathrm{o} + \boldsymbol{t}_2, \tag{8}$$

where $\mathbf{R}_{2\times 3}$ denotes the first two rows of $\mathbf{R}$ and $\boldsymbol{t}_2$ denotes the first two rows of $\boldsymbol{t}$. As noted above, the third row of $\mathbf{R}$ can be computed as the vector product of the two rows of $\mathbf{R}_{2\times 3}$, while the element t_z of $\boldsymbol{t}$ cannot be determined and can be set to 0.

To simplify the discussion below, from now on, we will omit the subscripts from $\mathbf{R}_{2\times 3}$ and $\boldsymbol{t}_2$. Consequently, from now on $\mathbf{R}$ will denote a 2×3 matrix with orthogonal rows ($\mathbf{R}\mathbf{R}^\top = \mathbf{I}_2$) and $\boldsymbol{t}$ will denote $(t_x, t_y)^\top$. Furthermore, to simplify the notation, we define $\boldsymbol{x}' = \boldsymbol{p}_\mathrm{o}$ and $\boldsymbol{y}' = \boldsymbol{p}_\mathrm{p}$.

To solve the OnP problem for n point correspondences $\boldsymbol{x}'_i \leftrightarrow \boldsymbol{y}'_i$, we minimize the following reprojection error over $\mathbf{R}$ and $\boldsymbol{t}$:

$$\varepsilon^2 = \sum_{i=1}^{n} \|\mathbf{R}\boldsymbol{x}'_i + \boldsymbol{t} - \boldsymbol{y}'_i\|_2^2. \tag{9}$$

Proposition 1 *The translation $\boldsymbol{t}$ in (9) is given by*

$$\boldsymbol{t} = \bar{\boldsymbol{y}} - \mathbf{R}\bar{\boldsymbol{x}}, \tag{10}$$

where

$$\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}' \quad \textit{and} \quad \bar{\boldsymbol{y}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{y}'. \tag{11}$$

Proof The proof is analogous to the proofs in [27, Sections II.B and III.B]. □

As a result of Proposition 1, the OnP problem reduces to the following orthogonal Procrustes problem:

$$\min_{\mathbf{R}} \sum_{i=1}^{n} \|\mathbf{R}\boldsymbol{x}_i - \boldsymbol{y}_i\|_2^2, \tag{12}$$

where $\boldsymbol{x}_i = \boldsymbol{x}'_i - \bar{\boldsymbol{x}}$ and $\boldsymbol{y}_i = \boldsymbol{y}'_i - \bar{\boldsymbol{y}}$.

As is customary for Procrustes problems [25], we can stack the points $\boldsymbol{x}_i^\top$ into an $n \times 3$ matrix $\mathbf{X}$ and the points $\boldsymbol{y}_i^\top$ into an $n \times 2$ matrix $\mathbf{Y}$ and can write (12) as

$$\min_{\mathbf{Q}} = \|\mathbf{X}\mathbf{Q} - \mathbf{Y}\|_\mathrm{F}^2, \tag{13}$$

where $\mathbf{Q} = \mathbf{R}^\top$ and $\|\mathbf{M}\|_\mathrm{F}$ denotes the Frobenius norm of $\mathbf{M}$.

Proposition 2 *Every Procrustes problem* $\min_{\mathbf{Q}} \|\mathbf{X}\mathbf{Q} - \mathbf{Y}\|_\mathrm{F}^2$, *where* $\mathbf{Q}$ *is a* $p \times q$ *matrix,* $\mathbf{X}$ *is an* $n \times p$ *matrix,* $\mathbf{Y}$ *is an* $n \times q$ *matrix,* $p \geq q$, *and* $n \geq p$, *can be reduced to an equivalent Procrustes problem* $\min_{\mathbf{Q}} \|\mathbf{X}'\mathbf{Q} - \mathbf{Y}'\|_\mathrm{F}^2$, *where* $\mathbf{X}'$ *is a* $p \times p$ *matrix and* $\mathbf{Y}'$ *is a* $p \times q$ *matrix.*

Proof The proof is adapted from [10, Section 2] and [24, Chapter 5.3.3]. Let us compute the QR decomposition of $\mathbf{X}$: $\mathbf{X} = \mathbf{S}\mathbf{T}$, where $\mathbf{S}$ has orthogonal columns ($\mathbf{S}^\top\mathbf{S} = \mathbf{I}_p$) and $\mathbf{T}$ is upper triangular, i.e., $\mathbf{T}$ can be written as $\mathbf{T} = (\mathbf{U}^\top, \mathbf{0}^\top)^\top$, where $\mathbf{U}$ is a $p \times p$ upper triangular matrix. Then, $\mathbf{T} = \mathbf{S}^\top\mathbf{X}$. Furthermore, $\mathbf{V} = \mathbf{S}^\top\mathbf{Y} = (\mathbf{V}_1^\top, \mathbf{V}_2^\top)^\top$, where $\mathbf{V}_1$ is $p \times q$ and $\mathbf{V}_2$ is $(n-p) \times q$. By using the fact that the Frobenius norm is invariant to rotations, we obtain:

$$\begin{aligned} \|\mathbf{X}\mathbf{Q} - \mathbf{Y}\|_\mathrm{F}^2 &= \|\mathbf{S}^\top(\mathbf{X}\mathbf{Q} - \mathbf{Y})\|_\mathrm{F}^2 && (14)\\ &= \|\mathbf{T}\mathbf{Q} - \mathbf{V}\|_\mathrm{F}^2 && (15)\\ &= \|\mathbf{U}\mathbf{Q} - \mathbf{V}_1\|_\mathrm{F}^2 + \|\mathbf{0}\mathbf{Q} - \mathbf{V}_2\|_\mathrm{F}^2 && (16)\\ &= \|\mathbf{U}\mathbf{Q} - \mathbf{V}_1\|_\mathrm{F}^2 + \|\mathbf{V}_2\|_\mathrm{F}^2. && (17) \end{aligned}$$

Since $\|\mathbf{V}_2\|_\mathrm{F}^2$ is constant, (13) is equivalent to the reduced Procrustes problem

$$\min_{\mathbf{Q}} \|\mathbf{X}'\mathbf{Q} - \mathbf{Y}'\|_\mathrm{F}^2, \tag{18}$$

where $\mathbf{X}' = \mathbf{U}$ and $\mathbf{Y}' = \mathbf{V}_1$ are computed from $\mathbf{X}$ and $\mathbf{Y}$ via the QR decomposition of $\mathbf{X}$ as described above. □

2.3 The OnP Problem for Non-coplanar 3D Points

We now apply the results of Sect. 2.2 to the OnP problem for non-coplanar points.

Obviously, we need at least three points $\boldsymbol{x}_i$ in general position, i.e., not collinear, to have a finite number of solutions. Since three points are always coplanar, we assume $n \geq 4$. The case of coplanar points will be discussed in Sect. 2.4. We assume that $\mathbf{X}$ has full rank.

Proposition 2 shows that $\mathbf{X}$ can be reduced to a 3×3 matrix and $\mathbf{Y}$ to a 3×2 matrix.

Definition 1 The Stiefel manifold $\mathcal{V}_{p,q}$ is the set of all $p \times q$ matrices $\mathbf{Q}$ with orthogonal columns ($p \geq q$), i.e., $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}_q$ [65]. An element $\mathbf{Q} \in \mathcal{V}_{p,q}$ is called a Stiefel matrix. We denote $\mathcal{V}_{3,2}$ by $\mathbb{S}$.

The discussion in Sect. 2.2 shows that the OnP problem for non-coplanar 3D points is equivalent to the following Procrustes problem:

$$\min_{\mathbf{Q}\in\mathbb{S}} = \|\mathbf{X}\mathbf{Q} - \mathbf{Y}\|_{\mathrm{F}}^2. \tag{19}$$

The Procrustes problem in (19) is called the projection Procrustes problem (e.g., [25]) or the unbalanced Procrustes problem (e.g., [57,71]). We will discuss algorithms that can be used to solve (19) in Sect. 3.

Remark 1 The Stiefel manifold $\mathbb{S}$ obviously has three degrees of freedom: the matrices $\mathbf{Q}$ have six degrees of freedom and the orthogonality requirement $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}_2$ provides three equations that the elements of $\mathbf{Q}$ must fulfill.

Remark 2 Instead of the direct representation of the rotation by its matrix elements and the constraints $\mathbf{R}\mathbf{R}^\top = \mathbf{I}_2$, we can also parameterize the rotation by a unit quaternion $\boldsymbol{q} = (q_0, q_1, q_2, q_3)^\top$ ($\|\boldsymbol{q}\|_2^2 = 1$). Then, the matrix $\mathbf{R}$ is given by

$$\mathbf{R} = \begin{pmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) \end{pmatrix}. \tag{20}$$

2.4 The OnP Problem for Coplanar 3D Points

Like for non-coplanar points, we must have at least three points $\boldsymbol{x}_i$ in general position, i.e., not collinear, to have a finite number of solutions.

Since the points $\boldsymbol{x}_i$ are coplanar, without loss of generality we can assume that they lie in the plane $z = 0$. This means that the third column of $\mathbf{X}$ will be $\boldsymbol{0}$. This shows that for coplanar points the third row of $\mathbf{Q}$ (i.e., the third column of $\mathbf{R}$) cannot be determined from the Procrustes problem alone. Therefore, we may omit the third column of $\mathbf{X}$ and the third row of $\mathbf{Q}$ in (13).

Definition 2 A sub-Stiefel matrix $\mathbf{Q}_\mathrm{s}$ is a $p \times p$ matrix that is obtained by deleting the last row of a $(p+1) \times p$ Stiefel matrix $\mathbf{Q}$:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_\mathrm{s} \\ \boldsymbol{q}^\top \end{pmatrix} \tag{21}$$

for some $\boldsymbol{q} \in \mathbb{R}^p$ [10]. We denote the set of 2×2 sub-Stiefel matrices by $\mathbb{S}_\mathrm{s}$.

Proposition 2 and [10, Lemma 4.1][2] show that, for coplanar points, $\mathbf{X}$ and $\mathbf{Y}$ can be reduced to 2×2 matrices.

[2] [10, Lemma 4.1] shows that multiplying a sub-Stiefel matrix from the left or right by an orthogonal matrix results in a sub-Stiefel matrix.

From the discussion in Sect. 2.2, it follows that the OnP problem for coplanar 3D points is equivalent to the following Procrustes problem:

$$\min_{\mathbf{Q}_\mathrm{s}\in\mathbb{S}_\mathrm{s}} = \|\mathbf{X}\mathbf{Q}_\mathrm{s} - \mathbf{Y}\|_{\mathrm{F}}^2. \tag{22}$$

Algorithms that can be used to solve (22) will be discussed in Sect. 4.

Proposition 3 *If* $\mathbf{Q}_\mathrm{s} \in \mathbb{S}_\mathrm{s}$, *then*

$$\mathrm{tr}(\mathbf{Q}_\mathrm{s}^\top\mathbf{Q}_\mathrm{s}) - (\det\mathbf{Q}_\mathrm{s})^2 = \|\mathbf{Q}_\mathrm{s}\|_{\mathrm{F}}^2 - (\det\mathbf{Q}_\mathrm{s})^2 = 1. \tag{23}$$

Proof Suppose $\mathbf{Q}_\mathrm{s} \in \mathbb{S}_\mathrm{s}$. Then, by Definition 2 there is a matrix $\mathbf{Q} \in \mathbb{S}$ of which $\mathbf{Q}_\mathrm{s}$ is the upper 2×2 matrix. Let us denote the elements of $\mathbf{Q}$ by q_{ij}. Since the columns of $\mathbf{Q}$ are orthogonal, we have:

$$q_{11}^2 + q_{21}^2 + q_{31}^2 = 1 \tag{24}$$
$$q_{12}^2 + q_{22}^2 + q_{32}^2 = 1 \tag{25}$$
$$q_{11}q_{12} + q_{21}q_{22} + q_{31}q_{32} = 0. \tag{26}$$

To eliminate q_{31} and q_{32}, which do not occur in $\mathbf{Q}_\mathrm{s}$, we can solve (24) and (25) for q_{31} and q_{32}, respectively:

$$q_{31} = \pm\sqrt{1 - q_{11}^2 - q_{21}^2} \tag{27}$$
$$q_{32} = \pm\sqrt{1 - q_{12}^2 - q_{22}^2}. \tag{28}$$

Moving the term $q_{31}q_{32}$ to the right-hand side and substituting (27) and (28) into (26) results in

$$q_{11}q_{12} + q_{21}q_{22} = \pm\sqrt{1 - q_{11}^2 - q_{21}^2}\sqrt{1 - q_{12}^2 - q_{22}^2}. \tag{29}$$

To remove the sign ambiguity on the right-hand side, we can square both sides to obtain

$$(q_{11}q_{12}+q_{21}q_{22})^2 = (1-q_{11}^2-q_{21}^2)(1-q_{12}^2-q_{22}^2). \tag{30}$$

By bringing all terms to the left-hand side and simplifying, we obtain

$$q_{11}^2 + q_{12}^2 + q_{21}^2 + q_{22}^2 - (q_{11}q_{22} - q_{12}q_{21})^2 - 1 = 0. \tag{31}$$

This proves the proposition. □

Remark 3 Proposition 3 shows that the set $\mathbb{S}_\mathrm{s}$ has three degrees of freedom: the matrices $\mathbf{Q}_\mathrm{s}$ have four degrees of freedom and the requirement (23) provides one equation that the elements of $\mathbf{Q}_\mathrm{s}$ must fulfill.

Remark 4 Theorems 4.2 and 4.4 of [10] give two additional characterizations of sub-Stiefel matrices. We will not make use of these characterizations in this paper.

Proposition 4 *Given a sub-Stiefel matrix* $\mathbf{Q}_\mathrm{s} \in \mathbb{S}_\mathrm{s}$, *there are two possible solutions for the associated* $\mathbf{Q} \in \mathbb{S}$.

Proof The possible solutions for q_{31} and q_{32} are given by (27) and (28). We can select one solution (q_{31}, q_{32}) arbitrarily based on (26). It is obvious from (24)–(26) that $(-q_{31}, -q_{32})$ will be a second solution. It is also obvious that the other two candidates $(q_{31}, -q_{32})$ and $(-q_{31}, q_{32})$ cannot be solutions because if (q_{31}, q_{32}) fulfills (26) then neither $(q_{31}, -q_{32})$ nor $(-q_{31}, q_{32})$ can fulfill (26) (unless $(q_{31}, q_{32}) = (0, 0)$). □

Corollary 1 *The OnP problem for coplanar points has two possible poses that correspond to the solution* $\mathbf{Q}_\mathrm{s}$.

Remark 5 Corollary 1 is also proved for three point correspondences under weak perspective in [38, Proposition 2]. Furthermore, [38] shows that the two solutions correspond to the well-known Necker reversal. This is also true for the O*n*P problem.

Remark 6 A method to reconstruct a full rotation matrix from a general $p \times p$ sub-Stiefel matrix is given in [10, Theorem 4.3]. As in Proposition 4, there are two possible solutions.

Remark 7 If the two solutions in Proposition 4 are represented by Euler angles as $\mathbf{R} = \mathbf{R}_x(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_z(\gamma)$, the two solutions are related in a very simple manner: if one solution is given by (α, β, γ), the second solution is given by $(-\alpha, -\beta, \gamma)$.

Remark 8 Like for non-coplanar points (see Remark 2), we can parameterize the rotation by a unit quaternion $\boldsymbol{q} = (q_0, q_1, q_2, q_3)^\top$ $(\|\boldsymbol{q}\|_2^2 = 1)$, resulting in the matrix

$$\mathbf{R} = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 \end{pmatrix}. \tag{32}$$

Proposition 5 *Given a quaternion* $\boldsymbol{q} = (q_0, q_1, q_2, q_3)^\top$ *that solves the OnP problem for coplanar points, the second solution of the OnP problem derived in Proposition 4 and Corollary 1 is given by* $\boldsymbol{q}' = (q_0, -q_1, -q_2, q_3)^\top$.

Proof According to the proof of Proposition 4, the second solution has a matrix $\mathbf{R}$ for which the left 2×2 submatrix is identical to that of the first solution, whereas the third column is negated. Substituting $\boldsymbol{q}$ and $\boldsymbol{q}'$ into (20) proves the proposition. □

Corollary 2 *Based on Proposition 5 and the fact that* $\boldsymbol{q}$ *and* $-\boldsymbol{q}$ *represent the same rotation, there are four equivalent solutions to the OnP problem for coplanar points if the rotation is parameterized by quaternions. Uniqueness can be achieved, for example, by requiring* $q_0 \geq 0$ and $q_1 \geq 0$.

Remark 9 The translation $\boldsymbol{t}$ in Proposition 1 is identical for both solutions in Proposition 4.

3 Algorithms for Solving the O*n*P Problem for Non-coplanar Points

As discussed in Sect. 2.3, the O*n*P problem for non-coplanar points is equivalent to the unbalanced Procrustes problem. Consequently, we can use algorithms that have been proposed to solve this problem. The two main requirements for the algorithms we have are robustness and speed. Since the O*n*P problem may exhibit multiple local minima, the algorithm should ideally find the global minimum. Furthermore, the algorithm should be as fast as possible. The robustness and speed of the algorithms will be evaluated in Sect. 5.

An analytic solution for the unbalanced Procrustes problem was given by Cliff [13] (see also [25, Chapter 5.1]). However, this problem does not use the least-squares criterion we use in the O*n*P problem formulation. Instead, it uses an inner-product criterion to find the optimal rotation. Therefore, it solves a different problem than the one we are interested in.

Analytic solutions for the unbalanced $p \times q$ Procrustes problem using least squares as the optimization criterion are only known for $p = q$ (the balanced Procrustes problem) or $q = 1$. In our case, $p = 3$ and $q = 2$. Therefore, we must resort to iterative algorithms that may converge to local minima. This explains our requirements that an ideal algorithm should converge to the global minimum and should be as fast as possible.

One candidate algorithm we found is the algorithm of Green and Gower [25, Algorithm 5.1], which is also described in [66]. We will describe this algorithm in Sect. 3.1. We selected this algorithm as a candidate because the results in [71] indicated good performance.

Another candidate is the algorithm of Koschat and Swayne [41] (see also [25, Algorithm 5.2]). This algorithm will be described in Sect. 3.2. We selected this algorithm as a candidate because the results in [12] seemed to indicate reasonable performance.

There are further algorithms that have been proposed [7,53,57,71]. The results in [71] indicate that the algorithms proposed by Park [57] and Bojanczyk and Lutoborski [7] are significantly slower than the algorithm of Green and Gower. Furthermore, the algorithm proposed by Zhang and Du [71] is also slightly slower than the algorithm of Green and Gower. Finally, the results in [12] indicate that the algorithm of Mooijaart and Commandeur [53] is significantly slower than the

algorithm of Koschat and Swayne. Therefore, we did not consider any of these algorithms.

In our search for fast and robust algorithms, we also implemented a Levenberg–Marquardt algorithm (Sect. 3.3) and two algorithms that iteratively solve systems of polynomial equations that are based on Lagrange multipliers of the Procrustes problem (Sects. 3.4 and 3.5).

Finally, to check whether the above algorithms converge to the global optimum, we also implemented a solver based on an algorithm that is capable of finding the global optimum of polynomial optimization problems with polynomial equality and inequality constraints (Sect. 3.6).

3.1 The Algorithm of Green and Gower

The algorithm of Green and Gower is described in [25, Algorithm 5.1] (see also [66]). We extend it by reducing the number of point correspondences to three, as described in Proposition 2. Applied to the OnP problem for non-coplanar points, it can be described as follows:

1. Reduce $\mathbf{X}$ and $\mathbf{Y}$ to $\mathbf{X}'$ and $\mathbf{Y}'$ as described in Proposition 2.
2. Extend $\mathbf{Y}'$ to a 3×3 matrix by adding a $\boldsymbol{0}$ column on the right.
3. Set the result matrix $\mathbf{Q}$ to $\mathbf{I}_3$.
4. Use a balanced orthogonal Procrustes algorithm to determine the 3D rotation matrix $\mathbf{Q}'$ based on $\mathbf{X}'$ and the extended $\mathbf{Y}'$.
5. Replace $\mathbf{X}'$ by $\mathbf{X}'\mathbf{Q}'$.
6. Replace $\mathbf{Q}$ by $\mathbf{Q}\mathbf{Q}'$.
7. If the norm of the difference between the last column of $\mathbf{X}'$ and the last column of the extended $\mathbf{Y}'$ is below a threshold, go to step 9.
8. Replace the last column of the extended $\mathbf{Y}'$ by the last column of the updated $\mathbf{X}'$ and go to step 4.
9. Compute $\mathbf{R} = \mathbf{Q}^\top$ and $\boldsymbol{t}$ by (10). Set $t_z = 0$ (cf. Sect. 2.2).

There are numerous algorithms to solve the balanced orthogonal Procrustes problem in step 4 above [3,26,27,36,37,60,68,69], [25, Chapter 4]. We use the algorithm proposed in [68] because it ensures that a rotation matrix is returned (as opposed to a general orthogonal matrix, which also could include a reflection).

We also note that ten Berge and Knol [66] proposed a different initialization of the extension of $\mathbf{Y}'$ in step 2 above. They claimed that their modification helps the algorithm of Green and Gower to avoid local minima. This does not correspond to our experience. When we used the modified initial value in the experiments reported in Sect. 5.1, a significantly decreased robustness resulted in some of the experiments, i.e., the algorithm converged to local minima significantly more often. Therefore, we do not use the modified initialization proposed in [66].

3.2 The Algorithm of Koschat and Swayne

The algorithm of Koschat and Swayne was originally proposed in [41]. Our implementation is based on the modifications that are proposed in [25, Algorithm 5.2]. Since the inner loop of the algorithm does not use the matrices $\mathbf{X}$ and $\mathbf{Y}$ directly, the runtime of the inner loop does not depend on the number of point correspondences. Therefore, there is no need to reduce the number of point correspondences to three by Proposition 2. When applied to the OnP problem, the algorithm can be described as follows:

1. Initialize $\mathbf{Q}$ by the algorithm of Cliff (see [13] or [25, Chapter 5.1]).
2. Set $\rho^2 = \|\mathbf{X}\|_\mathrm{F}$.
3. Set $\mathbf{A} = \rho^2\mathbf{I} - \mathbf{X}^\top\mathbf{X}$ and $\mathbf{B} = \mathbf{Y}^\top\mathbf{X}$.
4. Set $\mathbf{Q}_\mathrm{o} = \mathbf{Q}$.
5. Set $\mathbf{Z} = \mathbf{B} + \mathbf{Q}_\mathrm{o}^\top\mathbf{A}$.
6. Compute the SVD of $\mathbf{Z}$: $\mathbf{Z} = \mathbf{U}\mathbf{S}\mathbf{V}^\top$.
7. Set $\mathbf{Q} = \mathbf{V}\mathbf{U}^\top$.
8. If $\|\mathbf{Q} - \mathbf{Q}_\mathrm{o}\|_\mathrm{F}$ is greater than a threshold, go to step 4.
9. Compute $\mathbf{R} = \mathbf{Q}^\top$. Compute the third row of $\mathbf{R}$ as the vector product of the first two rows. Compute $\boldsymbol{t}$ by (10). Set $t_z = 0$ (cf. Sect. 2.2).

3.3 The Levenberg–Marquardt Algorithm

The Levenberg–Marquardt algorithm we implemented is an adaptation of the algorithm described in [29, Appendix 6]. In contrast to the additive augmentation of the normal equations that is used in [29, Appendix 6], we use the multiplicative augmentation described in [58, Chapter 15.5]. The rotation matrix is parameterized by Euler angles, as described in Remark 7. First, the problem is reduced to three point correspondences by Proposition 2. We then use the same algorithm that is used in Sect. 3.4 to compute the initial estimate of the rotation matrix and initialize the Euler angles α, β, and γ from this rotation matrix. To terminate the Levenberg–Marquardt algorithm, we use the criteria that are described in [58, Chapter 15.5]. After the Levenberg–Marquardt algorithm has converged, we compute $\mathbf{R}$ from the Euler angles, $\boldsymbol{t}$ by (10), and set $t_z = 0$ (cf. Sect. 2.2).

3.4 The Iterative Polynomial System Solver Based on a Direct Representation of the Rotation

The OnP problem (12) for non-coplanar points can also be regarded as a constrained minimization problem. Let us denote the function to be minimized as

$$f(\mathbf{R}) = \sum_{i=1}^{n} \|\mathbf{R}\boldsymbol{x}_i - \boldsymbol{y}_i\|_2^2. \tag{33}$$

The constraint $\mathbf{R}\mathbf{R}^\top = \mathbf{I}_2$ can be written explicitly as

$$\boldsymbol{h}(\mathbf{R}) = \begin{pmatrix} r_{11}^2 + r_{12}^2 + r_{13}^2 - 1 \\ r_{21}^2 + r_{22}^2 + r_{23}^2 - 1 \\ r_{11}r_{21} + r_{12}r_{22} + r_{13}r_{23} \end{pmatrix} = \boldsymbol{0}. \tag{34}$$

Thus, the constrained minimization problem is given by

$$\begin{aligned} &\text{minimize } f(\mathbf{R}) \\ &\text{subject to } \boldsymbol{h}(\mathbf{R}) = \boldsymbol{0}. \end{aligned} \tag{35}$$

This is a problem of minimizing a polynomial function with polynomial constraints, which we solve using the following approach: the first-order necessary conditions for this problem are given by (see [50, Chapter 11.3]):

$$\nabla f(\mathbf{R}) + \boldsymbol{\lambda}^\top \nabla \boldsymbol{h}(\mathbf{R}) = \boldsymbol{0} \tag{36}$$

$$\boldsymbol{h}(\mathbf{R}) = \boldsymbol{0}, \tag{37}$$

where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \lambda_3)^\top \in \mathbb{R}^3$ are Lagrange multipliers.

Computing (36) explicitly using an approach analogous to that in [27, Section III.B] results in the following six equations:

$$a_{11}r_{11} + a_{12}r_{12} + a_{13}r_{13} + \lambda_1 r_{11} + \lambda_3 r_{21} - b_{11} = 0 \tag{38}$$

$$a_{11}r_{21} + a_{12}r_{22} + a_{13}r_{23} + \lambda_3 r_{11} + \lambda_2 r_{21} - b_{12} = 0 \tag{39}$$

$$a_{12}r_{11} + a_{22}r_{12} + a_{23}r_{13} + \lambda_1 r_{12} + \lambda_3 r_{22} - b_{21} = 0 \tag{40}$$

$$a_{12}r_{21} + a_{22}r_{22} + a_{23}r_{23} + \lambda_3 r_{12} + \lambda_2 r_{22} - b_{22} = 0 \tag{41}$$

$$a_{13}r_{11} + a_{23}r_{12} + a_{33}r_{13} + \lambda_1 r_{13} + \lambda_3 r_{23} - b_{31} = 0 \tag{42}$$

$$a_{13}r_{21} + a_{23}r_{22} + a_{33}r_{23} + \lambda_3 r_{13} + \lambda_2 r_{23} - b_{32} = 0, \tag{43}$$

where

$$\mathbf{A} = \mathbf{X}^\top \mathbf{X} \tag{44}$$

$$\mathbf{B} = \mathbf{X}^\top \mathbf{Y} \tag{45}$$

(using the notation of the stacked point matrices $\mathbf{X}$ and $\mathbf{Y}$ in (13)). In matrix notation, (38)–(43) can be written as

$$\mathbf{A}\mathbf{R}^\top + \mathbf{R}^\top \mathbf{L} = \mathbf{B}, \tag{46}$$

where

$$\mathbf{L} = \begin{pmatrix} \lambda_1 & \lambda_3 \\ \lambda_3 & \lambda_2 \end{pmatrix}. \tag{47}$$

This shows that (36) and (37) result in the nine equations (38)–(43) and (34). They all are polynomials of degree two in the nine unknowns $\mathbf{R}$ and $\boldsymbol{\lambda}$.

By Bézout's theorem [14, Chapter 3.3], the polynomial system has at most $2^9 = 512$ distinct solutions. We used the software Bertini [4] on some of the test problems that are reported in Sect. 5.1 to verify the correctness of the above formulation.[3] These experiments also showed that for the random noise experiment, there are typically four real finite solutions of the polynomial system. For the random point correspondence experiment, we found between four and twelve real finite solutions. The number of solutions was always even. A closer inspection of the data of the random point correspondence experiment showed that some of the solutions correspond to saddle points.

To solve the system of polynomial equations, we used the solver generator described in [42] to generate a solver for this problem.[4] In addition, we wrote a wrapper code around the solver that extracted the solution that had the minimum error of (33). When we integrated the solver into the experimental framework in Sect. 5.1, the experiments showed that the generated solver never returned a better solution (i.e., a solution with a smaller error) than the best solution of any of the other solvers. Furthermore, the experiments showed that the generated solver frequently failed to find any solution, sometimes in more than 70% of the random test cases, independent of the scenario (random noise, outliers, and random point correspondences). This is in stark contrast to the fact that (33), as a continuous function on a compact domain, by the extreme value theorem always has a global minimum. Furthermore, it contrasts with the fact that all the other solvers returned at least a local minimum of (33). Therefore, we did not consider the generated solver any further.

Since one of the goals we strive for is speed, we solve the nine polynomial equations by Newton's method. If we denote the nine equations as a function $\boldsymbol{z}(\boldsymbol{p})$, where $\boldsymbol{p}$ denotes the nine parameters in $\mathbf{R}$ and $\boldsymbol{\lambda}$, we start with an initial value $\boldsymbol{p}_0$ and iterate $\boldsymbol{p}_{i+1} = \boldsymbol{p}_i - (\nabla \boldsymbol{z}(\boldsymbol{p}_i))^{-1} \boldsymbol{z}(\boldsymbol{p}_i)$ until convergence.

Note that $\boldsymbol{z}(\boldsymbol{p})$ and $\nabla \boldsymbol{z}(\boldsymbol{p})$ only depend on $\mathbf{A}$ and $\mathbf{B}$. Therefore, there is no speed advantage if the number of point correspondences is reduced to three. Hence, we do not perform this step.

To obtain the initial value $\boldsymbol{p}_0$, we use the following heuristic: suppose the points $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are related perfectly by an orthogonal matrix $\mathbf{R}$. Then, we would have $\boldsymbol{\lambda} = \boldsymbol{0}$. Therefore, (46) reduces to $\mathbf{A}\mathbf{R}^\top = \mathbf{B}$, i.e., $\mathbf{R}^\top = \mathbf{A}^{-1}\mathbf{B}$. This means that, under the above hypothesis, $\mathbf{R}$ is simply given by the solution of the unrestricted Procrustes problem.

In reality, the points $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are not related perfectly by an orthogonal matrix. Consequently, the matrix $\mathbf{R}$ computed above will not be orthogonal. We therefore project $\mathbf{R}$ onto

[3] Bertini guarantees that all solutions are found. However, the runtime on a single instance of the problem is in the order of minutes.

[4] We used the version that is available from https://github.com/PavelTrutman/Automatic-Generator that was current at the time of the writing of this paper (git commit ID `71e530a55fd07d381bc022e1e18a3d46ef5be460`).

the closest orthogonal matrix. It is well known that this can be achieved via the SVD [5, Theorem 1], [33, Theorem 2.2]. Let the SVD of $\mathbf{R}$ be given by $\mathbf{R} = \mathbf{U}\mathbf{S}\mathbf{V}^\top$. Then, the closest orthogonal matrix to $\mathbf{R}$ is given by $\mathbf{R}_\mathrm{o} = \mathbf{U}\mathbf{V}^\top$. We use $\mathbf{R}_\mathrm{o}$ and $\boldsymbol{\lambda} = \boldsymbol{0}$ as the initial value $\boldsymbol{p}_0$.

The above algorithm will converge to any solution that fulfills the nine equations. Since the first-order conditions are merely necessary and not sufficient, this means that the algorithm might converge to a local maximum or a saddle point. To detect whether this happens, we use the second-order conditions for minimization problems with equality constraints, as described in [50, Chapter 11.5]. In our problem, we must check whether the Hessian matrix $\mathbf{L}(\mathbf{R}) = \mathbf{F}(\mathbf{R}) + \boldsymbol{\lambda}^\top \mathbf{H}(\mathbf{R})$ is positive on the tangent subspace $M = \{\boldsymbol{q} : \nabla \boldsymbol{h}(\mathbf{R})\boldsymbol{q} = \boldsymbol{0}\}$. Here, $\mathbf{F}(\mathbf{R})$ denotes the Hessian matrix of $f(\mathbf{R})$, $\boldsymbol{\lambda}^\top \mathbf{H}(\mathbf{R})$ is given by $\sum_{i=1}^{3} \lambda_i \mathbf{H}_i(\mathbf{R})$, $\mathbf{H}_i(\mathbf{R})$ denotes the Hessian matrix of $h_i(\mathbf{R})$, and $h_i(\mathbf{R})$ is the ith component of $\boldsymbol{h}(\mathbf{R})$. A basis for M can be calculated via the full SVD of $\nabla \boldsymbol{h}(\mathbf{R})$: It is given by the last three columns of $\mathbf{V}$. Let this matrix be called $\mathbf{E}(\mathbf{R})$. Therefore, we must check whether the matrix $\mathbf{E}(\mathbf{R})^\top \mathbf{L}(\mathbf{R})\mathbf{E}(\mathbf{R})$ is positive definite. To do so, we must examine the eigenvalues of this matrix and check whether they are all positive. If this is not the case, the algorithm has not converged to a local minimum. In this case, we could try to use one of the algorithms discussed in the previous sections or we could simply return an error. We will examine in Sect. 5.1 how often this case occurs.

Once $\mathbf{R}$ has been determined, $\boldsymbol{t}$ is determined by (10) and t_z is set to 0 (cf. Sect. 2.2).

3.5 The Iterative Polynomial System Solver Based on a Quaternion Representation of the Rotation

As mentioned in Remark 2, instead of the direct representation of the rotation, we can also parameterize the rotation by a unit quaternion to obtain the rotation matrix (20). We also have the constraint

$$h(\boldsymbol{q}) = q_0^2 + q_1^2 + q_2^2 + q_3^2 - 1 = 0. \tag{48}$$

Therefore, we have a constrained minimization problem

$$\begin{aligned} &\text{minimize } f(\boldsymbol{q}) \\ &\text{subject to } h(\boldsymbol{q}) = 0, \end{aligned} \tag{49}$$

where $f(\boldsymbol{q}) = f(\mathbf{R}(\boldsymbol{q}))$ is given by (33) and $\mathbf{R}(\boldsymbol{q})$ is given by (20). The first-order necessary conditions for this problem are given by (see [50, Chapter 11.3]):

$$\nabla f(\boldsymbol{q}) + \lambda \nabla h(\boldsymbol{q}) = \boldsymbol{0} \tag{50}$$

$$h(\boldsymbol{q}) = 0, \tag{51}$$

where $\lambda \in \mathbb{R}$ is a Lagrange multiplier. Computing (50) results in the following four polynomial equations:

$$\begin{aligned}
&4\Big(\big(q_0(q_0^2 + q_1^2 - q_2^2 + q_3^2) + 2q_1q_2q_3\big)a_{11} \\
&+ 2\big(q_3(q_2^2 - q_1^2) + 2q_0q_1q_2\big)a_{12} \\
&+ \big(q_2(q_0^2 - q_1^2 - q_2^2 + q_3^2) + 2q_0(q_0q_2 - q_1q_3)\big)a_{13} \\
&+ \big(q_0(q_0^2 - q_1^2 + q_2^2 + q_3^2) - 2q_1q_2q_3\big)a_{22} \\
&+ \big(q_1(q_1^2 + q_2^2 - q_3^2 - q_0^2) - 2q_0(q_0q_1 + q_2q_3)\big)a_{23} \\
&+ 2q_0(q_1^2 + q_2^2)a_{33} \\
&- q_0b_{11} + q_3b_{21} - q_2b_{31} \\
&- q_3b_{12} - q_0b_{22} + q_1b_{32}\Big) \\
&+ 2\lambda q_0 = 0
\end{aligned} \tag{52}$$

$$\begin{aligned}
&4\Big(\big(q_1(q_0^2 + q_1^2 + q_2^2 - q_3^2) + 2q_0q_2q_3\big)a_{11} \\
&+ 2\big(q_2(q_0^2 - q_3^2) - 2q_0q_1q_3\big)a_{12} \\
&+ \big(q_3(q_1^2 + q_2^2 - q_3^2 - q_0^2) + 2q_1(q_3q_1 - q_0q_2)\big)a_{13} \\
&+ \big(q_1(q_1^2 + q_2^2 + q_3^2 - q_0^2) - 2q_0q_2q_3\big)a_{22} \\
&+ \big(q_0(q_1^2 + q_2^2 - q_3^2 - q_0^2) + 2q_1(q_0q_1 + q_2q_3)\big)a_{23} \\
&+ 2q_1(q_0^2 + q_3^2)a_{33} \\
&- q_1b_{11} - q_2b_{21} - q_3b_{31} \\
&- q_2b_{12} + q_1b_{22} + q_0b_{32}\Big) \\
&+ 2\lambda q_1 = 0
\end{aligned} \tag{53}$$

$$\begin{aligned}
&4\Big(\big(q_2(q_1^2 + q_2^2 + q_3^2 - q_0^2) + 2q_0q_1q_3\big)a_{11} \\
&+ 2\big(q_1(q_0^2 - q_3^2) + 2q_0q_2q_3\big)a_{12} \\
&+ \big(q_0(q_0^2 - q_1^2 - q_2^2 + q_3^2) + 2q_2(q_1q_3 - q_0q_2)\big)a_{13} \\
&+ \big(q_2(q_0^2 + q_1^2 + q_2^2 - q_3^2) - 2q_0q_1q_3\big)a_{22} \\
&+ \big(q_3(q_1^2 + q_2^2 - q_3^2 - q_0^2) + 2q_2(q_0q_1 + q_3q_2)\big)a_{23} \\
&+ 2q_2(q_0^2 + q_3^2)a_{33} \\
&+ q_2b_{11} - q_1b_{21} - q_0b_{31} \\
&- q_1b_{12} - q_2b_{22} - q_3b_{32}\Big) \\
&+ 2\lambda q_2 = 0
\end{aligned} \tag{54}$$

$$\begin{aligned}
&4\Big(\big(q_3(q_0^2 - q_1^2 + q_2^2 + q_3^2) + 2q_0q_1q_2\big)a_{11} \\
&+ 2\big(q_0(q_2^2 - q_1^2) - 2q_1q_2q_3\big)a_{12} \\
&+ \big(q_1(q_1^2 + q_2^2 - q_3^2 - q_0^2) + 2q_3(q_0q_2 - q_1q_3)\big)a_{13} \\
&+ \big(q_3(q_0^2 + q_1^2 - q_2^2 + q_3^2) - 2q_0q_1q_2\big)a_{22} \\
&+ \big(q_2(q_1^2 + q_2^2 - q_3^2 - q_0^2) - 2q_3(q_0q_1 + q_2q_3)\big)a_{23} \\
&+ 2q_3(q_1^2 + q_2^2)a_{33} \\
&+ q_3b_{11} + q_0b_{21} - q_1b_{31} \\
&- q_0b_{12} + q_3b_{22} - q_2b_{32}\Big) \\
&+ 2\lambda q_3 = 0
\end{aligned} \tag{55}$$

Together with (48), we have five polynomial equations of degree two and three in the five unknowns. By Bézout's theorem [14, Chapter 3.3], this polynomial system has at most $2 \cdot 3^4 = 162$ distinct solutions. Since $\boldsymbol{q}$ and $-\boldsymbol{q}$ describe the same rotation, we can expect that this polynomial system has more local minima that that of Sect. 3.4.

Like for the polynomial system in Sect. 3.4, we used the solver generator described in [42] to generate a solver for this problem. In addition, we wrote a wrapper code around the solver that extracted the solution that had the minimum error of (33). When we integrated the solver into the experimental framework in Sect. 5.1, a similar problem to that described in Sect. 3.4 occurred: the generated solver failed to find any solution in up to 10% of the random test cases, independent of the scenario (random noise, outliers, and random point correspondences). Furthermore, the experiments showed that the generated solver never returned a better solution than the best solution of any of the other solvers. Therefore, we did not consider the generated solver any further.

Instead, we used the same approach as in Sect. 3.4: We use Newton's method to compute a solution. Since the inner loop only depends on $\mathbf{A}$ and $\mathbf{B}$, we do not reduce the number of point correspondences to three. The same initialization as in Sect. 3.4 is used. After convergence, we check whether the second-order conditions for minimization problems are fulfilled to ensure the algorithm has converged to a local minimum. Once $\boldsymbol{q}$ has been determined, we compute $\mathbf{R}$ and then $\boldsymbol{t}$ as in Sect. 3.4.[5]

3.6 The Polynomial System Solver Based on Gloptipoly

As noted in Sect. 3.4, solving (35) is a problem of minimizing a polynomial function with polynomial constraints. A specialized algorithm (called Gloptipoly) for these kinds of problems has been proposed in [30,31]. Therefore, we also implemented a solver based on Gloptipoly, version 3.8.

If the function $f(\mathbf{R})$ in (33) is expanded and the polynomial terms are collected, it can be seen that (33) is a polynomial of degree two in the entries of $\mathbf{R}$ that only depends on the entries of the matrices $\mathbf{A}$ and $\mathbf{B}$ (see (44) and (45)). Therefore, we did not reduce the number of point correspondences to three by Proposition 2. We did not use the quaternion parameterization of Sect. 3.5 because this would have resulted in a much more complicated polynomial equation of degree four. Based on the experiments reported in Sect. 5, it was determined that Gloptipoly's relaxation parameter had to be set to 2 to ensure that Gloptipoly finds the global optimum.

[5] One advantage of the quaternion parameterization is that the third row of $\mathbf{R}$ is explicitly specified by $\boldsymbol{q}$.

4 Algorithms for Solving the O*n*P Problem for Coplanar Points

None of the algorithms that are described in Sect. 3 work for coplanar points. For the algorithm by Green and Gower (Sect. 3.1), the norm of the difference in step 7 is 0 in this case. Therefore, the algorithm terminates immediately with an incorrect solution. For the algorithm of Koschat and Swayne (Sect. 3.2), the matrix $\mathbf{Q}$ in step 8 is equal to $\mathbf{Q}_\mathrm{o}$, which leads to the same problem. For the Levenberg–Marquardt algorithm (Sect. 3.3) and the iterative polynomial system solvers (Sects. 3.4 and 3.5), the initial solution cannot be computed since the matrix $\mathbf{A}$ is singular. Furthermore, for the Levenberg–Marquardt algorithm and the first polynomial solver, the equation systems that are solved in the inner loop of the algorithms are singular.

As described in Sect. 2.4, the O*n*P problem for coplanar points is equivalent to the sub-Stiefel Procrustes problem. The only algorithm for solving this problem that we could find is the algorithm of Cardoso and Ziętak [10], which is discussed in Sect. 4.1.

Analogous to the non-coplanar case, we also implemented a Levenberg–Marquardt algorithm (Sect. 4.2) and two algorithms that iteratively solve a system of polynomial equations that are based on the Karush–Kuhn–Tucker conditions of the Procrustes problem (Sect. 4.3) and on Lagrange multipliers of the Procrustes problem (Sect. 4.4).

Finally, to check whether the above algorithms converge to the global optimum, we also implemented a solver based on Gloptipoly (Sect. 4.5).

4.1 The Algorithm of Cardoso and Ziętak

The algorithm of Cardoso and Ziętak is described in [10, Section 7]. It uses a similar idea as the algorithm of Green and Gower: The sub-Stiefel Procrustes problem is expanded to a 3D orthogonal Procrustes problem. Applied to the coplanar O*n*P problem, it can be described as follows:

1. Reduce $\mathbf{X}$ and $\mathbf{Y}$ to the 2×2 matrices $\mathbf{X}'$ and $\mathbf{Y}'$ as described in Proposition 2.
2. Set $\mathbf{X} = \gamma\mathbf{X}'$ and $\mathbf{Y} = \gamma\mathbf{Y}'$. As discussed below, the parameter γ is required to ensure fast convergence of the algorithm.
3. Set $\mathbf{Q} = \mathrm{diag}(1, 0.5)$.
4. Set

$$\mathbf{X}_\mathrm{e} = \begin{pmatrix} \mathbf{X} & \boldsymbol{0} \\ \boldsymbol{0}^\top & 1 \end{pmatrix}. \tag{56}$$

5. Set $\boldsymbol{p} = \mathbf{X}(-\sqrt{0.75}, 0)^\top$ and $\boldsymbol{q} = (0, \sqrt{0.75})^\top$.

6. Set

$$\mathtt{Y}_\mathrm{e} = \begin{pmatrix} \mathtt{Y} & \boldsymbol{p} \\ \boldsymbol{q}^\top & 0.5 \end{pmatrix}. \tag{57}$$

7. Use a balanced orthogonal Procrustes algorithm to determine the 3D rotation matrix $\mathtt{Q}_\mathrm{e}$ based on $\mathtt{X}_\mathrm{e}$ and $\mathtt{Y}_\mathrm{e}$. We use the algorithm proposed in [68].
8. Partition $\mathtt{Q}_\mathrm{e}$ as

$$\mathtt{Q}_\mathrm{e} = \begin{pmatrix} \mathtt{Q}_\mathrm{n} & \boldsymbol{p} \\ \boldsymbol{q}^\top & \alpha \end{pmatrix}. \tag{58}$$

9. If $\|\mathtt{Q}_\mathrm{n} - \mathtt{Q}\|_\mathrm{F}^2$ is smaller than a threshold, set $\mathtt{Q} = \mathtt{Q}_\mathrm{n}$ and go to step 13.
10. Set $\mathtt{Q} = \mathtt{Q}_\mathrm{n}$.
11. Set

$$\mathtt{Y}_\mathrm{e} = \begin{pmatrix} \mathtt{Y} & \mathtt{X}\boldsymbol{p} \\ \mathrm{sign}(\alpha)\boldsymbol{q}^\top & |\alpha| \end{pmatrix}. \tag{59}$$

12. Go to step 7.
13. Complete $\mathtt{Q}$ to a 3×2 matrix by Proposition 4 (selecting an arbitrary solution of the two possible solutions; see Remark 7).
14. Compute $\mathtt{R} = \mathtt{Q}^\top$ and $\boldsymbol{t}$ by (10). Set $t_z = 0$ (cf. Sect. 2.2).

In our experiments, we found that the parameter γ is crucial for the convergence of the algorithm. If it is set too small, the algorithm will converge extremely slowly. We experimentally determined that $\gamma = 10{,}000$ is a suitable value for our formulation of the OnP problem. This presumably is the case since we use meters as the units for $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$. Because telecentric lenses cannot be arbitrarily large, the values of the point coordinates are typically in the order of a few centimeters. A principled way to select γ is currently unknown [10, Section 8].

4.2 The Levenberg–Marquardt Algorithm

The Levenberg–Marquardt algorithm for the coplanar OnP problem is mostly identical to that for the non-coplanar OnP problem in Sect. 3.3. The differences are that Proposition 2 allows us to reduce the problem to one with two point correspondences and that we have two potential solutions (see Proposition 4). Because the two solutions are related to each other in a very simple manner (see Remark 7), we only return one of the two possible solutions.

4.3 The Iterative Polynomial System Solver Based on a Direct Representation of the Rotation

The OnP problem (12) for coplanar points can also be regarded as a constrained minimization problem. Let us denote the function to be minimized as

$$f(\mathtt{R}) = \sum_{i=1}^{n} \|\mathtt{R}\boldsymbol{x}_i - \boldsymbol{y}_i\|_2^2. \tag{60}$$

The constraint in Proposition 3 can be written explicitly as

$$\begin{aligned} h(\mathtt{R}) = {} & r_{11}^2 + r_{12}^2 + r_{21}^2 + r_{22}^2 \\ & - (r_{11}r_{22} - r_{12}r_{21})^2 - 1 = 0. \end{aligned} \tag{61}$$

In addition, $\mathtt{R}$ obviously must fulfill the following inequality constraints:

$$\boldsymbol{g}(\mathtt{R}) = \begin{pmatrix} r_{11}^2 + r_{12}^2 - 1 \\ r_{21}^2 + r_{22}^2 - 1 \\ r_{11}^2 + r_{21}^2 - 1 \\ r_{12}^2 + r_{22}^2 - 1 \end{pmatrix} \leq \boldsymbol{0}. \tag{62}$$

Thus, the constrained minimization problem is given by

$$\begin{aligned} &\text{minimize} && f(\mathtt{R}) \\ &\text{subject to} && h(\mathtt{R}) = 0 \\ & && \boldsymbol{g}(\mathtt{R}) \leq \boldsymbol{0}. \end{aligned} \tag{63}$$

The first-order necessary conditions (the Karush–Kuhn–Tucker (KKT) conditions) for this problem are given by (see [50, Chapter 11.8]):

$$\nabla f(\mathtt{R}) + \lambda \nabla h(\mathtt{R}) + \boldsymbol{\mu}^\top \nabla \boldsymbol{g}(\mathtt{R}) = \boldsymbol{0} \tag{64}$$

$$\boldsymbol{\mu}^\top \boldsymbol{g}(\mathtt{R}) = \boldsymbol{0} \tag{65}$$

$$h(\mathtt{R}) = 0 \tag{66}$$

$$\boldsymbol{g}(\mathtt{R}) \leq \boldsymbol{0} \tag{67}$$

$$\boldsymbol{\mu} \geq \boldsymbol{0}, \tag{68}$$

where $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3, \mu_4)^\top$.

Computing (64)–(66) explicitly results in the following nine equations:

$$\begin{aligned} &2(a_{11}r_{11} + a_{12}r_{12} + \lambda r_{22}(r_{12}r_{21} - r_{11}r_{22}) \\ &\quad + (\lambda + \mu_1 + \mu_3)r_{11} - b_{11}) = 0 \end{aligned} \tag{69}$$

$$\begin{aligned} &2(a_{12}r_{11} + a_{22}r_{12} + \lambda r_{21}(r_{11}r_{22} - r_{12}r_{21}) \\ &\quad + (\lambda + \mu_1 + \mu_4)r_{12} - b_{21}) = 0 \end{aligned} \tag{70}$$

$$\begin{aligned} &2(a_{11}r_{21} + a_{12}r_{22} + \lambda r_{12}(r_{11}r_{22} - r_{12}r_{21}) \\ &\quad + (\lambda + \mu_2 + \mu_3)r_{21} - b_{12}) = 0 \end{aligned} \tag{71}$$

$$\begin{aligned} &2(a_{12}r_{21} + a_{22}r_{22} + \lambda r_{11}(r_{12}r_{21} - r_{11}r_{22}) \\ &\quad + (\lambda + \mu_2 + \mu_4)r_{22} - b_{22}) = 0 \end{aligned} \tag{72}$$

$$\mu_1(r_{11}^2 + r_{12}^2 - 1) = 0 \tag{73}$$

$$\mu_2(r_{21}^2 + r_{22}^2 - 1) = 0 \tag{74}$$

$$\mu_3(r_{11}^2 + r_{21}^2 - 1) = 0 \tag{75}$$
$$\mu_4(r_{12}^2 + r_{22}^2 - 1) = 0 \tag{76}$$
$$r_{11}^2 + r_{12}^2 + r_{21}^2 + r_{22}^2 - (r_{11}r_{22} - r_{12}r_{21})^2 - 1 = 0. \tag{77}$$

where $\mathbf{A}$ and $\mathbf{B}$ are given by (44) and (45), respectively.

As can be seen, the KKT conditions are nine polynomial equations of degree three or four in the nine unknowns $\mathbf{R}$, λ, and $\boldsymbol{\mu}$. By Bézout's theorem [14, Chapter 3.3], the polynomial system has at most $3^4 \cdot 4^5 = 82944$ distinct solutions. Using the software Bertini [4], we examined a few test instances of the problem.[6] It turned out that the polynomial system can have several tens of real finite solutions (many of which do not fulfill the condition $\boldsymbol{\mu} \geq \boldsymbol{0}$). Because of the large number of possible solutions, we did not attempt to use the solver generator described in [42].

Like for the non-coplanar OnP solvers in Sects. 3.4 and 3.5, we solve the nine polynomial equations by Newton's method. If we denote the nine equations as a function $\boldsymbol{z}(\boldsymbol{p})$, where $\boldsymbol{p}$ denotes the nine parameters in $\mathbf{R}$, λ, and $\boldsymbol{\mu}$, we start with an initial value $\boldsymbol{p}_0$ and iterate $\boldsymbol{p}_{i+1} = \boldsymbol{p}_i - (\nabla \boldsymbol{z}(\boldsymbol{p}_i))^{-1}\boldsymbol{z}(\boldsymbol{p}_i)$ until convergence.

Note that $\boldsymbol{z}(\boldsymbol{p})$ and $\nabla \boldsymbol{z}(\boldsymbol{p})$ only depend on $\mathbf{A}$ and $\mathbf{B}$. Therefore, we do not reduce the number of point correspondences to two.

To obtain the initial value $\boldsymbol{p}_0$, we use the same heuristic as in Sect. 3.4: suppose the points $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are related perfectly by an orthogonal matrix $\mathbf{R}$. Then, we would have $\lambda = 0$ and $\boldsymbol{\mu} = \boldsymbol{0}$. Therefore, (69)–(77) reduce to $\mathbf{A}\mathbf{R}^\top = \mathbf{B}$, i.e., $\mathbf{R}^\top = \mathbf{A}^{-1}\mathbf{B}$.

In reality, the points $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are not related perfectly by an orthogonal matrix. Consequently, the matrix $\mathbf{R}$ computed above will not be a sub-Stiefel matrix. We therefore project $\mathbf{R}$ onto the closest sub-Stiefel matrix. By [10, Theorem 5.5], this can be achieved via the SVD. Let the SVD of $\mathbf{R}$ be given by $\mathbf{R} = \mathbf{U}\mathbf{S}\mathbf{V}^\top$. Then, the closest sub-Stiefel matrix to $\mathbf{R}$ is given by $\mathbf{R}_\mathrm{s} = \mathbf{U}\mathbf{S}_*\mathbf{V}^\top$, where $\mathbf{S}_* = \mathrm{diag}(1, \ldots, 1, s_*)$, $s_* = \min(\sigma_n(\mathbf{R}), 1)$, and $\sigma_n(\mathbf{R})$ denotes the smallest singular value or $\mathbf{R}$. We use $\mathbf{R}_\mathrm{s}$, $\lambda = 0$, and $\boldsymbol{\mu} = \boldsymbol{0}$ as the initial value $\boldsymbol{p}_0$.

The above algorithm will converge to any solution that fulfills the nine equations. Since the first-order conditions are merely necessary and not sufficient, the algorithm might converge to a local maximum, a saddle point, or even to a point outside the feasible region (62). We use the second-order conditions for minimization problems with equality and inequality constraints [50, Chapter 11.8] to detect whether this happens. In our problem, we must check whether the Hessian matrix $\mathbf{L}(\mathbf{R}) = \mathbf{F}(\mathbf{R}) + \lambda\mathbf{H}(\mathbf{R}) + \boldsymbol{\mu}^\top\mathbf{G}(\mathbf{R})$ is positive on the tangent subspace $M = \{\boldsymbol{q} : \nabla h(\mathbf{R})\boldsymbol{q} = \boldsymbol{0} \wedge \nabla g_j(\mathbf{R})\boldsymbol{q} = \boldsymbol{0}\ \forall j : g_j(\mathbf{R}) = 0 \wedge \mu_j > 0\}$, i.e., on the tangent subspace corresponding to the equality constraint and all active inequality constraints. Here, $\mathbf{F}(\mathbf{R})$ denotes the Hessian matrix of $f(\mathbf{R})$, $\mathbf{H}(\mathbf{R})$ the Hessian matrix of $h(\mathbf{R})$, $\boldsymbol{\mu}^\top\mathbf{G}(\mathbf{R})$ is given by $\sum_{i=1}^4 \mu_i\mathbf{G}_i(\mathbf{R})$, $\mathbf{G}_i(\mathbf{R})$ denotes the Hessian matrix of $g_i(\mathbf{R})$, and $g_i(\mathbf{R})$ is the ith component of $\boldsymbol{g}(\mathbf{R})$. A basis for M can be calculated via the full SVD of the matrix composed of the gradients of the equality constraint and the active inequality constraints: It is given by the last $m + 1$ columns of $\mathbf{V}$, where m denotes the number of active inequality constraints. Let this matrix be called $\mathbf{E}(\mathbf{R})$. Therefore, we must check whether the matrix $\mathbf{E}(\mathbf{R})^\top\mathbf{L}(\mathbf{R})\mathbf{E}(\mathbf{R})$ is positive definite. To do so, we must examine the eigenvalues of this matrix and check whether they are all positive. Furthermore, we check whether the conditions $\boldsymbol{\mu} \geq \boldsymbol{0}$ are fulfilled. If any of the preceding conditions are not fulfilled, the algorithm has not converged to a feasible local minimum. In this case, we could try to use one of the algorithms discussed in the previous sections or we could simply return an error. We will examine in Sect. 5.2 how often this case occurs.

Once $\mathbf{R}$ has been determined, it must be completed to a 2×3 matrix in a manner analogous to Proposition 4 (an arbitrary solution of the two possible solutions can be selected; see Remark 7). Finally, $\boldsymbol{t}$ is determined by (10) and t_z is set to 0 (cf. Sect. 2.2).

4.4 The Iterative Polynomial System Solver Based on a Quaternion Representation of the Rotation

As mentioned in Remark 8, we can also parameterize the rotation in (60) using unit quaternions. The matrix $\mathbf{R}$ in (60) is given by (32). With this, conceptually (48)–(51) remain unchanged. The resulting polynomial system is given by (52)–(55), where all the terms involving a_{13}, a_{33}, a_{33}, b_{31}, and b_{32} are deleted.

By Bézout's theorem [14, Chapter 3.3], this polynomial system has at most $2 \cdot 3^4 = 162$ distinct solutions. This is a significantly smaller number of potential solutions than that of the approach in Sect. 4.3.

Like for the polynomial systems in Sects. 3.4 and 3.5, we used the solver generator described in [42] to generate a solver for this problem. It turned out that the generated solver returned solutions that did not even fulfill the polynomial equations that were specified to the solver generator. Therefore, we did not use the generated solver.

Instead, like for the other polynomial solvers, we use Newton's method to compute a solution. Since the inner loop only depends on $\mathbf{A}$ and $\mathbf{B}$, we do not reduce the number of point correspondences to two. The same initialization as in Sect. 4.3 is used. After convergence, we check whether the second-order conditions for minimization problems are fulfilled to ensure the algorithm has converged to a local minimum. Once $\boldsymbol{q}$ has been determined, we compute $\mathbf{R}$ and then $\boldsymbol{t}$ as in Sect. 4.3.

[6] Bertini required several hours of computation time to compute all 82944 solutions of a particular instance to the problem.

4.5 The Polynomial System Solver Based on Gloptipoly

Like for the non-coplanar case (cf. Sect. 3.6), we also implemented a solver based on Gloptipoly [30,31]. In contrast to the non-coplanar case, we used the quaternion representation of Sect. 4.4 to parameterize the rotation, resulting in a polynomial function of degree four that must be minimized under the constraint (48) and the two constraints given in Corollary 2. As before, the function to be optimized depends on the point correspondences only via the matrices $\mathbf{A}$ and $\mathbf{B}$ (see (44) and (45)). Therefore, we did not reduce the number of point correspondences to two. Based on the experiments reported in Sect. 5, it was determined that Gloptipoly's relaxation parameter had to be set to 2 to ensure that Gloptipoly finds the global optimum.

5 Evaluation

In this section, we will evaluate the algorithms that were described in Sects. 3 and 4. Our purpose is to answer the question which of the proposed algorithms best fulfills the following two criteria:

- *Robustness* Ideally, the algorithm should find the global minimum of the OnP problem.
- *Speed* The algorithm should be as fast as possible.

To evaluate the algorithms with respect to these criteria, we performed the following experiments: n random non-coplanar or coplanar 3D points and a random pose were generated. The random non-coplanar 3D points were distributed uniformly $\in [-0.01, 0.01]^3$ [m]. The coplanar points were distributed uniformly $\in [-0.01, 0.01]^2$ [m] ($z = 0$). The random 3D points were projected into a virtual image of size 2560×1920 using a telecentric camera with the following data: $m = 0.08$, no distortion,[7] $s_x = s_y = 2$ µm, $(c_x, c_y)^\top = (1180, 1010)^\top$. The number of point correspondences n was varied between the minimum number, i.e., 4 or 3, and 50,000 in roughly logarithmic increments.[8] The random 3D points were projected into the image using the random pose to obtain 2D points. For each n, 10,000 random experiments were performed. To test the robustness, three different scenarios were evaluated:

- *Random noise* The 3D points and their 2D correspondences were disturbed by random, uniformly distributed noise. The maximum value of the noise corresponded to 1% of the diameter of the point cloud. Hence, the 3D points were disturbed by noise uniformly distributed $\in [-0.0001, 0.0001]^d$ [m] ($d = 3$ for the non-coplanar algorithms and $d = 2$ for the coplanar algorithms) and the 2D points were disturbed by noise uniformly distributed $\in [-4, 4]^2$ [Pixel]. Note that this is a very large amount of noise that is not realistic for real applications and is solely designed to test the robustness of the algorithms.
- *Outliers* 80% of the 3D points and their 2D correspondences were disturbed by random, uniformly distributed noise. To increase the difficulty of the problem, the maximum value of the noise corresponded to 2% of the diameter of the point cloud. Hence, the 3D points were disturbed by noise uniformly distributed $\in [-0.0002, 0.0002]^d$ [m] ($d = 3$ for the non-coplanar algorithms and $d = 2$ for the coplanar algorithms) and the 2D points were disturbed by noise uniformly distributed $\in [-8, 8]^2$ [Pixel]. Furthermore, 20% of the point correspondences were created as outliers. This was done by adding noise to the point correspondences that amounted to 100% of the diameter of the point cloud, i.e., noise $\in [-0.01, 0.01]^d$ [m] ($d = 3$ for the non-coplanar algorithms and $d = 2$ for the coplanar algorithms) for the 3D points and $\in [-400, 400]$ [Pixel] for the 2D points. If there were fewer than five point correspondences, at least one outlier was generated to ensure that the results were always contaminated by outliers. This scenario is intended to test the algorithms with a moderate number of outliers and an extremely large noise level.
- *Random point correspondences* To make the problem as difficult as possible, the n 3D points were replaced by random points $\in [-0.01, 0.01]^d$ [m] ($d = 3$ for the non-coplanar algorithms and $d = 2$ for the coplanar algorithms). The 2D points were unchanged. Consequently, the 3D points had no functional relation to the 2D points whatsoever, i.e., the outlier ratio was 100%. The motivation for this experiment is mainly to evaluate the breaking point of the algorithms and to identify significant performance differences between the algorithms. We do not consider this experiment relevant for real applications.

We implemented the Gloptipoly solvers in Sects. 3.6 and 4.5 to determine the global optimum of the above problems. The runtime of both algorithms in MATLAB is approximately 2 s. All the other algorithms were implemented in C as HALCON[9] operators in a HALCON extension package. Our evaluation framework was also implemented within HALCON. To interface the Gloptipoly solvers to the evaluation

[7] The distortions are immaterial. As described in Sects. 2.1 and 2.2, the image points are transformed to metric points $\boldsymbol{p}_\mathrm{p}$ using the interior orientation of the camera, thereby removing any lens distortions.

[8] Conceptually, the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ and multiples of powers of 10 of this set were used. For the non-coplanar algorithms, the minimum value for n was 4, for the coplanar algorithms it was 3.

[9] See http://www.halcon.com/.

framework, we initially called the MATLAB executable for each problem instance. This created an overhead of around 14 s per call, raising the execution time to 16 s. To reduce the overhead, we compiled the Gloptipoly solvers into executables using the MATLAB command `mcc`. This decreased the overhead to around 4 s. Hence, one call of the Gloptipoly solvers took approximately 6 s. With this execution time, the experiments reported in the following would have required approximately three months to run for each scenario. Unfortunately, this meant that it was infeasible to integrate the Gloptipoly solvers into the experiments in Sects. 5.1 and 5.2 directly. Therefore, we will first evaluate the robustness of all algorithms except the Gloptipoly solvers and will then address the question whether the proposed solvers converge to the global optimum based on experiments with fewer different numbers of point correspondences and with fewer trials.

Therefore, in the first evaluation, we use the solution with the minimum root-mean-square (RMS) error returned by all evaluated algorithms as the global minimum, i.e., as the correct solution. We will then use the second evaluation to determine an estimate for the probability that the Gloptipoly solvers find a better solution than the best solution of all the other algorithms. We will see that this probability is extremely low for all scenarios, i.e., it is almost certain that one of the iterative algorithms converges to the global optimum. This justifies our procedure for the first evaluation.

In the first evaluation, we count a solution as being correct if it achieves an RMS error that lies within 0.1% of the smallest RMS error. This takes into account that the algorithms have slightly different stopping criteria. In the evaluation, we report in what percentage of the 10,000 trials the respective algorithm has converged to the solution with the minimum RMS error. Therefore, the ideal algorithm would achieve an evaluation of 100% on this metric. To test the speed of the algorithms, we report their average runtime in milliseconds on the 10,000 trials, as measured on a 3.2 GHz Intel Core i5-4570 CPU under Linux. All algorithms were implemented in C. The linear algebra used in the algorithms was implemented using LAPACK [1].

5.1 Algorithms for Non-coplanar 3D Points

The results of the evaluation of the algorithm of Green and Gower (Sect. 3.1), the algorithm of Koschat and Swayne (Sect. 3.2), our Levenberg–Marquardt algorithm (Sect. 3.3), and our proposed iterative polynomial system solvers (Sects. 3.4 and 3.5) are displayed in Fig. 1. For the iterative polynomial system solver, two results are displayed. One graph displays the robustness if the internal self diagnosis taken into account and an error is returned if the internal self diagnosis indicates that the algorithm has not converged to a local minimum, i.e., the result is only counted as correct if the internal self diagnosis indicates that the algorithm has converged to a local minimum and if the RMS error of the local minimum corresponds to the smallest RMS error of all algorithms, as described above. A second graph displays the results if the internal self diagnosis is taken into account and the algorithm of Green and Gower is used as a fallback if the internal self diagnosis indicates that the algorithm has not converged to a local minimum. In the interest of brevity, we will refer to these two cases as "without fallback" and "with fallback" below.

The experiments with random noise in Fig. 1a show that in this experiment all algorithms almost always find the correct solution. The only exceptions are the cases $n = 4$ and $n = 5$, where the algorithms in rare cases do not find the correct solution. In these cases, the two polynomial system solvers without fallback, the polynomial system solver with fallback that is based on unit quaternions, and the algorithm by Koschat and Swayne perform slightly worse than the other three algorithms. Interestingly, the performance of the polynomial system solver with fallback that is based on a direct rotation representation is the best of the seven algorithms. The runtimes in Fig. 1b show that the polynomial system solvers are asymptotically the fastest. It is only for $n \leq 200$ that the Levenberg–Marquardt algorithm is slightly faster than some of the polynomial system solvers.[10] For $n = 100$, the polynomial system solvers and the Levenberg–Marquardt algorithm are faster by a factor of more than 10 than the algorithm of Green and Gower and by a factor of more than 100 than the algorithm of Koschat and Swayne (note the logarithmic scale of the runtime graphs). Asymptotically, the polynomial system solvers are faster by a factor of more than 4 than all other algorithms. The algorithm of Koschat and Swayne is the slowest by a large margin. The reason for this is that, especially for small n, it requires tens of thousands of iterations to converge. Thus, we can conclude that for the random noise experiment, the polynomial system solvers, in particular the polynomial system solver with fallback that is based on a direct rotation representation, provide the best tradeoff between speed and robustness.

In the experiment with outliers (see Fig. 1c, d), the polynomial system solvers without fallback, the polynomial solver with fallback that is based on unit quaternions, and the

[10] It might be surprising to see that the polynomial system solvers with fallback are a little faster than those without fallback. Theoretically, they should be equally fast since the fallback is practically never called in this experiment. This seems to be an artifact of the compilation process. The algorithms are so fast that slight changes in the layout of the code by the compiler can have a significant performance impact. In our software development, we sometimes have seen drastic changes in performance of identical assembler code simply by the fact that the code was aligned differently. We assume that the same effect has happened here: it is likely that the code of the solvers with fallback is faster because it was aligned in a more favorable manner than the code of the solvers without fallback.

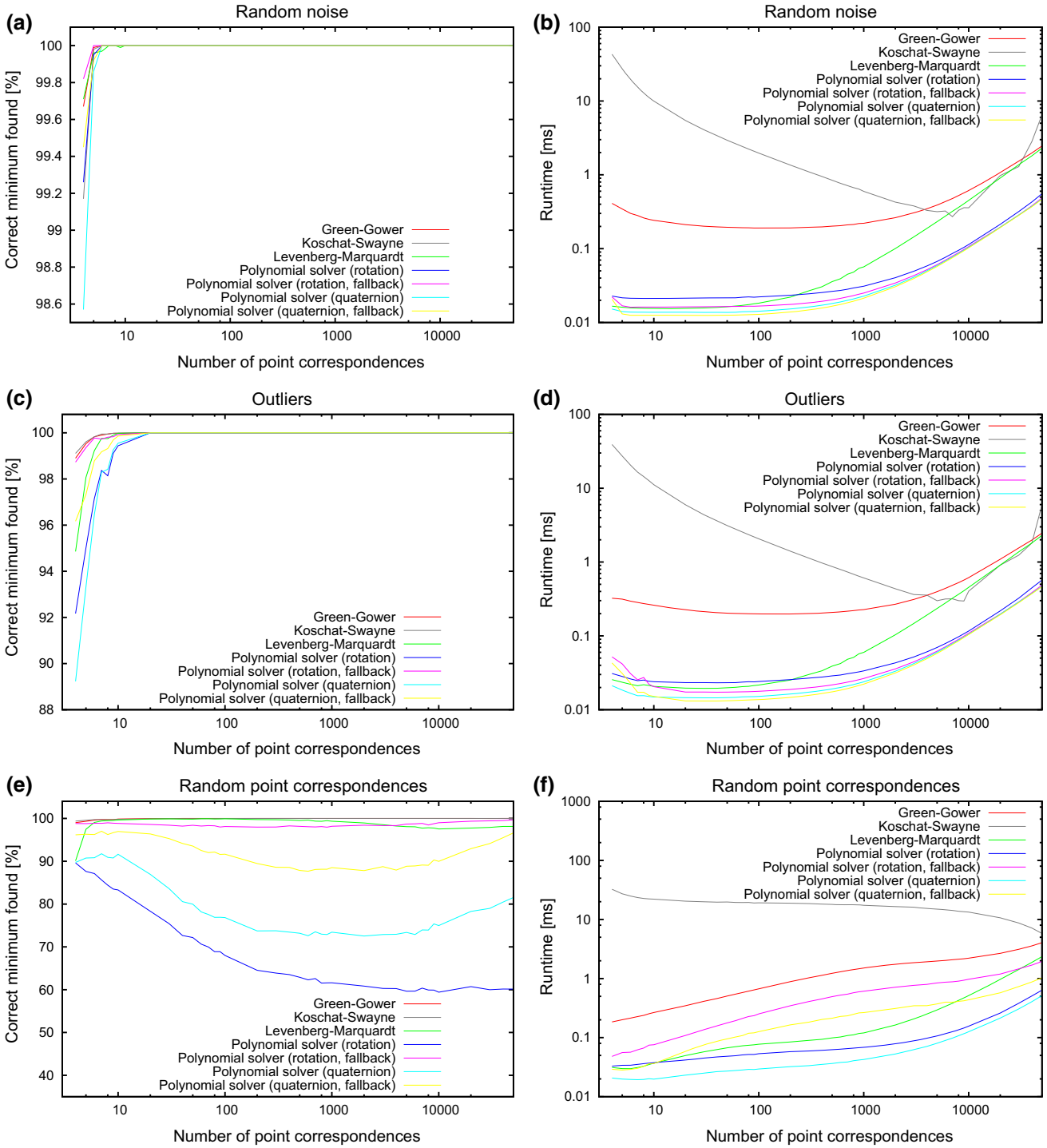

Fig. 1 Results of the evaluation of the O*n*P algorithms for non-coplanar points for three test cases. The results in the left column display the percentage of the tests in which an algorithm found the best solution as a function of the number of point correspondences. The best solution was determined as the solution that had the lowest RMS error of all algorithms for a particular random data set. A total of 10,000 experiments with random poses, random points, and random noise were executed for each number of point correspondences. The right column displays the average runtime in milliseconds as a function of the number of point correspondences. Note the logarithmic scale on the x axis in both columns and on the y axis in the right column. See the text for details about the three experiments displayed in the graphs

Levenberg–Marquardt algorithm perform the worst for small n, while the polynomial system solver with fallback that is based on a direct rotation representation, the algorithm of Green and Gower and that of Koschat and Swayne perform best. For larger n, all four algorithms find the correct solution in all cases. The runtime behavior of the algorithms is very similar to the random noise case. Therefore, in this case, the best tradeoff between robustness and speed is achieved by the polynomial system solver with fallback that is based on a direct rotation representation.

For the experiment with completely random point correspondences (100% outliers), Fig. 1e, f show that the algorithm by Green and Gower and that by Koschat and Swayne almost always achieve the correct solution. The Levenberg–Marquardt algorithm sometimes does not find the correct solution for small or large n. The polynomial system solvers without fallback provide the worst robustness. Their heuristic to obtain the initial solution fails because the underlying assumptions are no longer fulfilled. The polynomial system solver that is based on a direct rotation representation converges to saddle points or local maxima in up to 40% of the cases. With the fallback, the correct solution can be obtained in more than 98% of the cases. However, in the remaining less than 2% of the cases, the algorithm converges to a different local minimum. Interestingly, the polynomial system solver that is based on unit quaternions converges to saddle points or local maxima in only up to 28% of the cases. With the fallback, however, it performs worse than the polynomial system solver with fallback that is based on a direct rotation representation. It sometimes fails to find the correct solution in more than 12% of the cases. Obviously, the more complex formulation of Sect. 3.5 causes more local minima than the simpler formulation of Sect. 3.4. Note that the polynomial system solvers without fallback are by far the fastest algorithms. With fallback, they are beaten by the Levenberg–Marquardt algorithm, which is less robust, however. In this experiment, the algorithm of Green and Gower clearly results in the best tradeoff between robustness and speed. However, this case is not really representative for the problems we are trying to solve in practice. Here, we would assume that the image points are related to the 3D points, at least partially. Consequently, the first two experiments are much more relevant for real applications. Therefore, the polynomial system solver with fallback that is based on a direct rotation representation seems to be the best choice overall. The algorithm of Green and Gower can be used for exceedingly difficult cases, such as those in the third experiment.

We now examine the question how often the above algorithms do not converge to the global optimum. As described previously, we test how often the Gloptipoly solver of Sect. 3.6 finds a better solution than the best solution returned by the other algorithms. We counted a solution as better if the Gloptipoly solver achieved an RMS error that was at least 0.001% smaller than the smallest RMS error of the remaining algorithms. We used the same scenarios, but reduced the number of trials to 1000 (instead of 10,000, as in the above experiments) and used only 20 different numbers of point correspondences (instead of 38), resulting in 20,000 trials for each scenario. Despite this large reduction, the evaluation of each scenario still required more than two days of computation time.

From this experiment, we can derive an estimate for the probability p_b that the Gloptipoly solver returns a better solution than the best solution of the remaining algorithms. For the random noise scenario, the Gloptipoly solver never returned a better solution ($p_b = 0\%$), for the outlier scenario, it found two instances with a smaller RMS error ($p_b = 0.01\%$), and for the random point correspondence scenario, it found twelve instances ($p_b = 0.06\%$). Therefore, even for the scenario that was designed to produce the maximum likelihood for the iterative algorithms to converge to the wrong local minimum, it is extremely likely (>99.9%) that at least one of the iterative algorithms converges to the global minimum.

5.2 Algorithms for Coplanar 3D Points

Figure 2 displays the results of the evaluation of the algorithm of Cardoso and Ziętak (Sect. 4.1), our Levenberg–Marquardt algorithm (Sect. 4.2), and our proposed iterative polynomial system solvers (Sects. 4.3 and 4.4). Like in Sect. 5.1, two results are displayed for the iterative polynomial system solvers: the results without and with fallback. In this case, the algorithm of Cardoso and Ziętak was used as the fallback.

Figure 2a shows that the algorithm of Cardoso and Ziętak performs slightly less robustly for small n than the Levenberg–Marquardt algorithm and the polynomial system solvers without fallback. The polynomial system solvers with fallback show the best overall robustness. Furthermore, as shown by Fig. 2b, the polynomial system solvers are asymptotically the fastest algorithms.[11] For $n = 100$, the polynomial system solvers and the Levenberg–Marquardt algorithm are faster than the algorithm of Cardoso and Ziętak by a factor of more than 30. Asymptotically, the polynomial system solvers are faster than the Levenberg–Marquardt algorithm by a factor of more than 3.

For the experiment with outliers (Fig. 2c, d), the Levenberg–Marquardt algorithm performs the worst for small n, followed by the polynomial system solver without fallback that is based on a direct rotation representation. The algorithm of Cardoso and Ziętak and the polynomial

[11] As for the non-coplanar case, there are instances for which the algorithms with fallback are faster than those without fallback. As explained in Sect. 5.1, it is likely that this is an artifact that is caused by different code alignment.

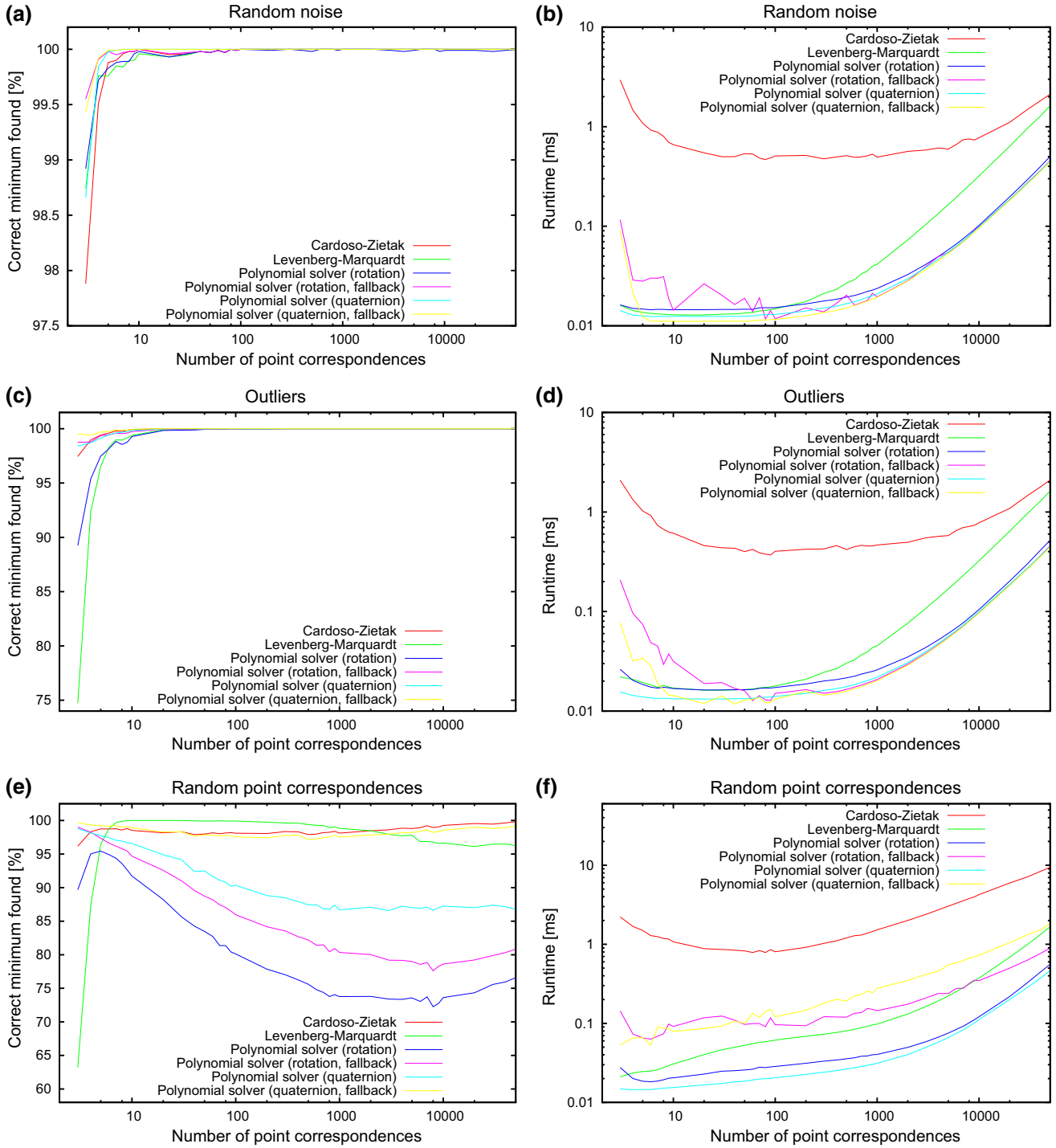

Fig. 2 Results of the evaluation of the O*n*P algorithms for coplanar points for three test cases. The results in the left column display the percentage of the tests in which an algorithm found the best solution as a function of the number of point correspondences. The best solution was determined as the solution that had the lowest RMS error of all algorithms for a particular random data set. 10,000 experiments with random poses, random points, and random noise were executed for each number of point correspondences. The right column displays the average runtime in milliseconds as a function of the number of point correspondences. Note the logarithmic scale on the x axis in both columns and on the y axis in the right column. See the text for details about the three experiments displayed in the graphs

system solver with fallback that is based on a direct rotation representation and the polynomial system solver without fallback that is based on unit quaternions perform almost equally well. The polynomial system solver with fallback that is based on unit quaternions performs best overall. All algorithms return the correct result for larger n. The runtime behavior of the algorithms is very similar to the random noise case. Therefore, in this case, the best tradeoff between robustness and speed is achieved by the polynomial system solver with fallback that is based on unit quaternions.

Finally, the results of the experiment with completely random point correspondences (100% outliers; see Fig. 2e, f) show that there is no clear winner in terms of robustness. For small n, the algorithm of Cardoso and Ziętak and the polynomial system solvers (without or with fallback) and for large n the algorithm of Cardoso and Ziętak and the polynomial system solver based on unit quaternions outperform the Levenberg–Marquardt algorithm. On the other hand, the Levenberg–Marquardt algorithm performs best for medium n. Overall, the polynomial system solvers without fallback and the polynomial system solver with fallback that is based on a direct rotation representation exhibit the least robustness. It is interesting to note the differences between the different kinds of polynomial system solvers. The solvers that are based on a direct rotation representation without or with fallback perform worse than the solver without fallback that is based on unit quaternions. The fallback for the solver that is based on the direct rotation representation does not improve the performance substantially. It is likely that the reason for this is that the more complex formulation in Sect. 4.3 causes many more local minima than the comparatively simpler formulation in Sect. 4.4. Note that, like in the non-coplanar case, the polynomial system solvers are by far the fastest algorithms. In this experiment, the algorithm of Cardoso and Ziętak and the polynomial system solver with fallback that is based on unit quaternions seem to be the best compromise. As discussed in Sect. 5.1, we do not regard the completely random case as representative for the problems we are trying to solve in practice. Consequently, the first two experiments are much more relevant for real applications. Therefore, the polynomial system solver with fallback based on unit quaternions seems to be the best choice overall. Because it shows a slight robustness advantage, the algorithm of Cardoso and Ziętak can be used for exceedingly difficult cases, such as those in the third experiment.

We now turn to the question how often the above algorithms do not converge to the global optimum. As in Sect. 5.1, we used the same scenarios, but reduced the number of trials to 1000 and used only 21 different numbers of point correspondences (instead of 39), resulting in 21000 trials for each scenario.

For the random noise scenario, the Gloptipoly solver never returned a better solution ($p_\mathrm{b} = 0\%$), for the outlier scenario, it found one instance with a smaller RMS error ($p_\mathrm{b} = 0.0048\%$), and for the random point correspondence scenario, it found 60 instances ($p_\mathrm{b} = 0.286\%$). Therefore, even for the scenario that was designed to produce the maximum likelihood for the iterative algorithms to converge to the wrong local minimum, it is extremely likely (>99.6%) that at least one of the iterative algorithms converges to the global minimum.

An application of the algorithms in this section is to use them as a minimal solver ($n = 3$) in a RANSAC scheme [19] to automatically determine the pose of an object from point correspondences with outliers. The evaluation shows that the polynomial system solver without fallback based on unit quaternions is the most suitable algorithm for this purpose. Its robustness for $n = 3$ is better than that of the algorithm of Cardoso and Ziętak, the Levenberg–Marquardt algorithm, and the polynomial system solver without fallback that is based on a direct rotation representation. Furthermore, although the robustness increases slightly if the fallback is used, this does not seem to justify the fourfold increase in the runtime. The RANSAC scheme will benefit more from the increased speed of the minimal solver than from the slight increase in robustness.

5.3 Accuracy of the Results

In addition to the robustness and speed, we evaluated the accuracy of the proposed algorithms. This was done by creating random poses and random 3D points in an identical manner as in the robustness experiments. The 3D points were then projected into a virtual image and were disturbed by uniform noise of a maximum amplitude a. Hence, the standard deviation of the noise was $\sigma_\mathrm{n} = a/\sqrt{3}$. The number of point correspondences n was varied as in the robustness experiments, while the noise amplitude was varied between 0 and 10 in steps of 0.5. For each combination of n and a, 10,000 trials were performed. All the proposed algorithms except the Gloptipoly solvers were tested.

We evaluated four error measures. The first measure is the norm of the difference between the true translation $\boldsymbol{t}_\mathrm{t}$ and the translation estimated by the algorithms $\boldsymbol{t}_\mathrm{e}$: $\varepsilon_{\boldsymbol{t}} = \|\boldsymbol{t}_\mathrm{t} - \boldsymbol{t}_\mathrm{e}\|_2$. Since all algorithms determine the rotation matrix $\mathbf{R}$, the second error measure we evaluated is the Frobenius norm of error of the rotation matrix: $\varepsilon_{\mathbf{R}} = \|\mathbf{R}_\mathrm{t} - \mathbf{R}_\mathrm{e}\|_\mathrm{F}$. For the non-coplanar O$n$P algorithms, $\mathbf{R}$ is a 2×3 matrix, while for the coplanar OnP algorithms, it is a 2×2 matrix. Since $\varepsilon_{\mathbf{R}}$ is hard to interpret geometrically, we derived two additional error measures from the rotation matrices. We converted the rotations into an axis–angle representation $(\boldsymbol{n}, \theta)$ $(\|\boldsymbol{n}\|_2 = 1)$. For the coplanar algorithms, we ensured that the solution that

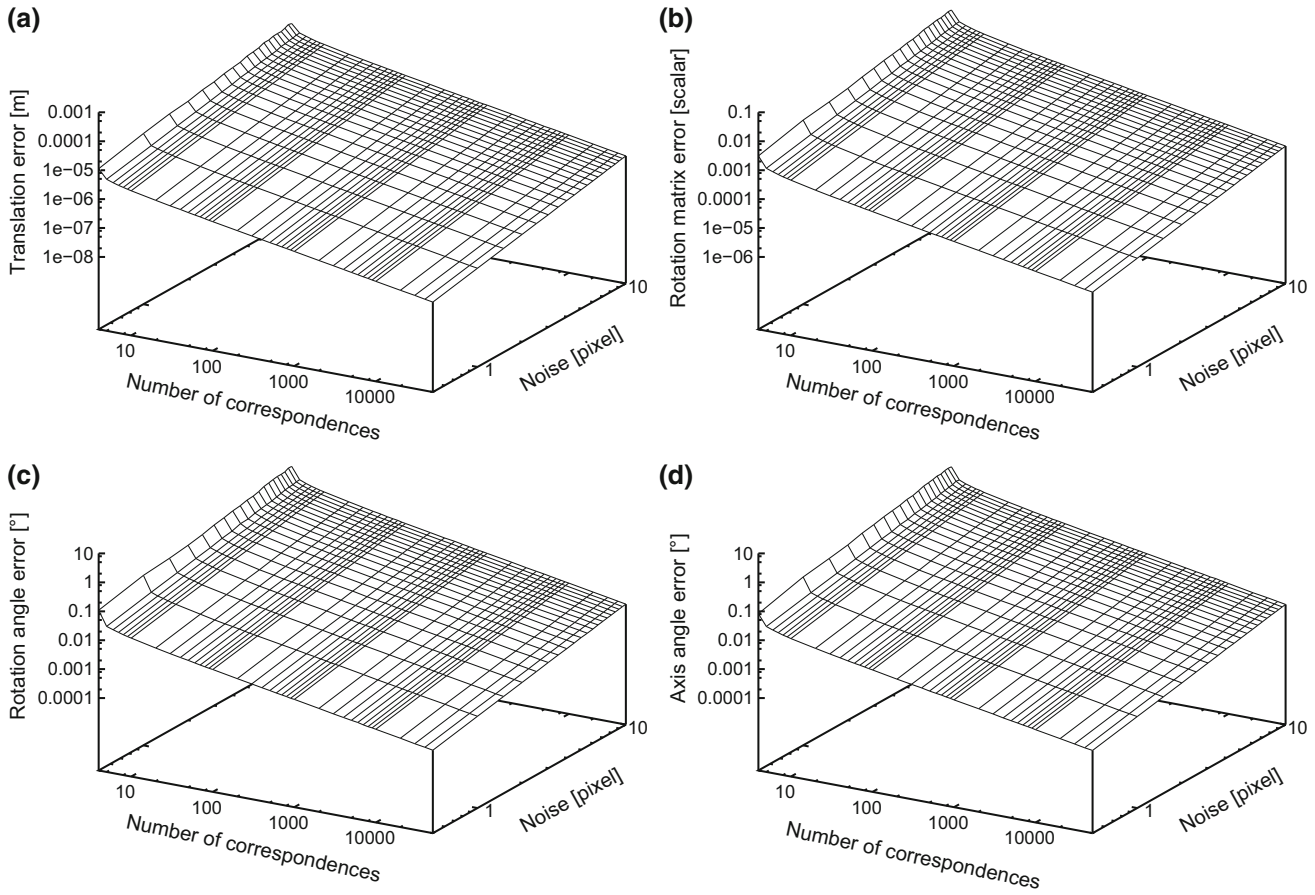

Fig. 3 Average errors for the pose parameters for the OnP algorithms for non-coplanar points as functions of the number of point correspondences and the noise level. Note the logarithmic scale on all three axes. **a** Average translation error. **b** Average norm of the error of the elements of the rotation matrix. **c** Average error of the rotation angle of the pose. **d** Average error of the rotation axis of the pose

corresponds best to the true pose was selected.[12] With this, the third measure is the rotation angle error $\varepsilon_\theta = |\theta_\mathrm{t} - \theta_\mathrm{e}|$ and the fourth measure is the angle error between the rotation axes $\varepsilon_{\boldsymbol{n}} = \arccos(\boldsymbol{n}_\mathrm{t} \cdot \boldsymbol{n}_\mathrm{e})$.

The results of the evaluation are displayed in Fig. 3 for the non-coplanar OnP algorithms and in Fig. 4 for the coplanar OnP algorithms. All algorithms return almost exactly the same accuracies. This is not surprising since all algorithms solve the same minimization problem (9). The only difference is the parameterization of the rotation. However, at the end of each algorithm, the result is converted to a rotation matrix. Since the algorithms converge to the same minimum of (9), it does not matter which path they took to reach the minimum. Consequently, we only display the results of a single algorithm for each case (the algorithm of Sect. 3.4 for the non-coplanar case and the algorithm of Sect. 4.4 for the coplanar case).

From [29, Result 5.2], we expect that the error measures are proportional to σ_n (and hence to a) and to $1/\sqrt{n}$. To check whether this is the case, we plot the results in Figs. 3 and 4 logarithmically for all axes.[13] We obviously had to omit the values for $a = 0$ to be able to use logarithmic axes. The errors for $a = 0$ are all in the range of the floating point precision of the machine on which we ran the tests (e.g., around 10^{-14} for ε_t). As can be seen from Figs. 3 and 4, the errors behave as expected: they are proportional to $a/\sqrt{n}$, except for very small values of n, where they are slightly larger. For reasonable amounts of noise ($a \leq 1$), the translation errors are always smaller than 25 µm for the non-coplanar case (for $n = 4$), while for the coplanar case they are always smaller than 60 µm (for $n = 3$). Similarly, the angle errors are always smaller than 0.25° for the non-coplanar case and 1° for the coplanar case. The errors decrease quickly for increasing n.

[12] As discussed in Sect. 2.4, there are two possible solutions with different rotations, but identical translations.

[13] To adapt a quote by Thomas Koenig: The joy of engineering is to find a plane on a triple logarithmic diagram.

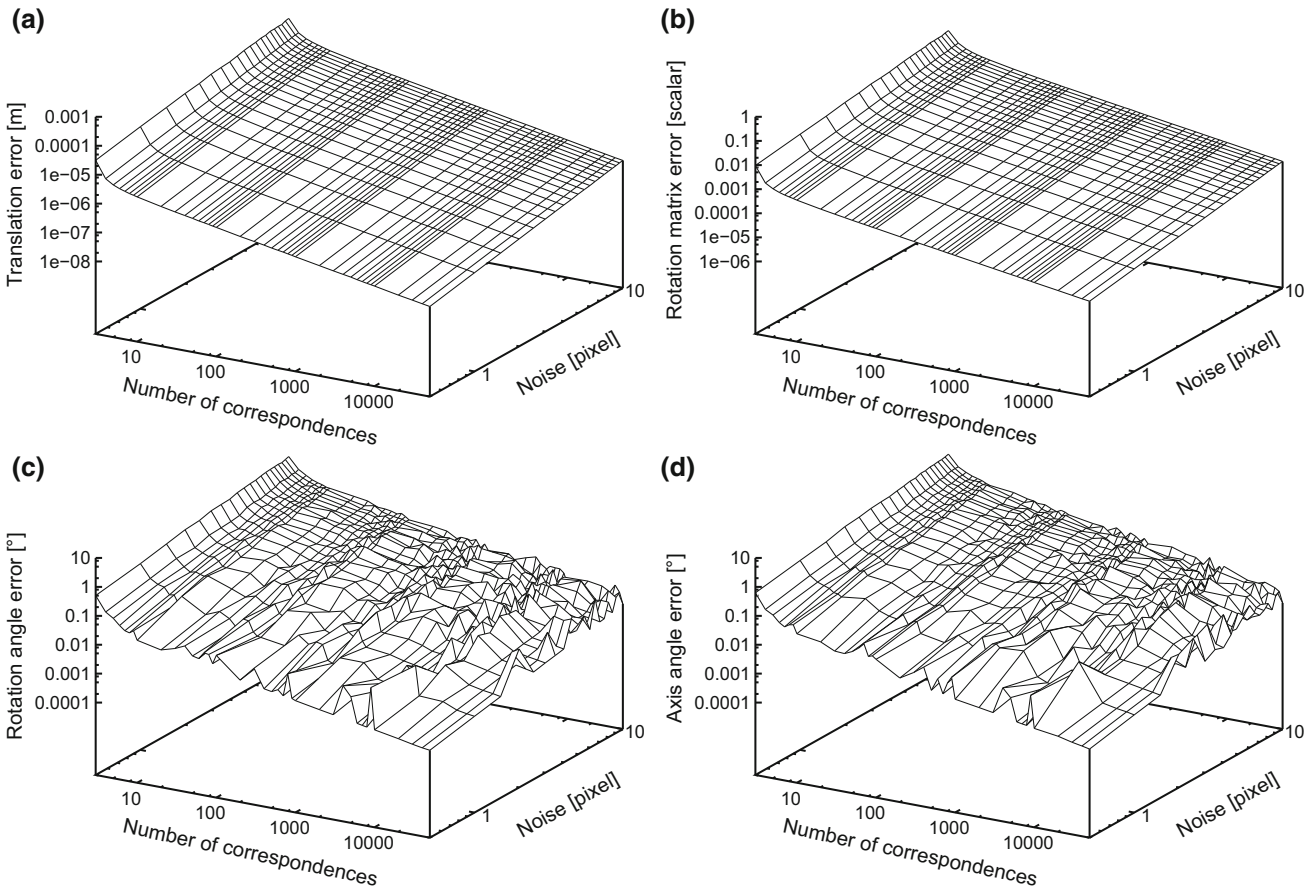

Fig. 4 Average errors for the pose parameters for the O*n*P algorithms for coplanar points as functions of the number of point correspondences and the noise level. Note the logarithmic scale on all three axes. **a** Average translation error. **b** Average norm of the error of the elements of the rotation matrix. **c** Average error of the rotation angle of the pose. **d** Average error of the rotation axis of the pose

6 Conclusions

We have examined the O*n*P problem (the P*n*P problem for cameras with telecentric lenses) and have shown that it is equivalent to the unbalanced orthogonal Procrustes problem for non-coplanar 3D points and to the sub-Stiefel Procrustes problem for coplanar 3D points.

For non-coplanar 3D points, we have applied two existing algorithms (Green and Gower; Koschat and Swayne) to the O*n*P problem. Furthermore, we have proposed three novel algorithms (one based on the Levenberg–Marquardt algorithm and two based on iterative polynomial system solvers) to solve the O*n*P problem. The evaluation of the algorithms has shown that the polynomial system solvers provide the best tradeoff between robustness and speed for reasonably posed problems (with noise and outliers in the data). For exceedingly difficult problems (100% outliers), the algorithm of Green and Gower provides the optimum robustness. However, it is substantially slower than the polynomial system solvers. The Levenberg–Marquardt algorithm does not provide the optimum robustness for a small number of point correspondences and it does not provide the fastest runtime for a large number of point correspondences. The algorithm of Koschat and Swayne is quite robust, but extremely slow compared to the other algorithms. Overall, the polynomial system solver with fallback that is based on a direct rotation representation is the best choice, while the algorithm of Green and Gower is a useful choice if robustness on very difficult problems is essential.

For coplanar 3D points, we have applied one existing algorithm (Cardoso and Ziętak) to the O*n*P problem. Furthermore, we have proposed three novel algorithms (one based on the Levenberg–Marquardt algorithm and two based on iterative polynomial system solvers) to solve the O*n*P problem. Based on the evaluation of the four algorithms, it was determined that the polynomial system solvers with fallback provide the best tradeoff between robustness and speed for reasonably posed problems (with noise and outliers in the data). For exceedingly difficult problems (100% outliers), there is no clear winner. The algorithm by Cardoso and Ziętak and the

polynomial system solver based on unit quaternions seem slightly preferable to the Levenberg–Marquardt algorithm in terms of robustness. Overall, the polynomial system solver with fallback that is based on unit quaternions is the best choice. If robustness on very difficult problems is essential, the algorithm by Cardoso and Ziętak and the polynomial system solver based on unit quaternions seem to be the best choice. The evaluation also has shown that the polynomial system solver without fallback that is based on unit quaternions is the best choice for a minimal solver in a RANSAC scheme.

References

1. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenny, A., Sorensen, D.: LAPACK User's Guide, 3rd edn. SIAM (1999)
2. Ansar, A., Daniilidis, K.: Linear pose estimation from points or lines. IEEE Trans. Pattern Anal. Mach. Intell. **25**(5), 578–589 (2003)
3. Arun, K.S., Huang, T.S., Blostein, S.D.: Least-squares fitting of two 3-d point sets. IEEE Trans. Pattern Anal. Mach. Intell. **9**(5), 698–700 (1987)
4. Bates, D.J., Hauenstein, J.D., Sommese, A.J., Wampler, C.W.: Bertini: Software for numerical algebraic geometry. http://bertini.nd.edu/ with permanent DOI:10.7274/R0H41PB5. Online Accessed 28 Dec 2016
5. Björck, Å., Bowie, C.: An iterative algorithm for computing the best estimate of an orthogonal matrix. SIAM J. Numer. Anal. **8**(2), 358–364 (1971)
6. Blahusch, G., Eckstein, W., Steger, C., Lanser, S.: Algorithms and evaluation of a high precision tool measurement system. In: 5th International Conference on Quality Control by Artificial Vision, pp. 31–36 (1999)
7. Bojanczyk, A.W., Lutoborski, A.: The Procrustes problem for orthogonal Stiefel matrices. SIAM J. Sci. Comput. **21**(4), 1291–1304 (1999)
8. Brown, D.C.: Decentering distortion of lenses. Photogramm. Eng. **32**(3), 444–462 (1966)
9. Brown, D.C.: Close-range camera calibration. Photogramm. Eng. **37**(8), 855–866 (1971)
10. Cardoso, J.R., Ziętak, K.: On a sub-Stiefel Procrustes problem arising in computer vision. Numer. Linear Algebra Appl. **22**(3), 523–547 (2015)
11. Chen, C.S., Chang, W.Y.: On pose recovery for generalized visual sensors. IEEE Trans. Pattern Anal. Mach. Intell. **26**(7), 848–861 (2004)
12. Chu, M.T., Trendafilov, N.T.: The orthogonally constrained regression revisited. J. Comput. Graph. Stat. **10**(4), 746–771 (2001)
13. Cliff, N.: Orthogonal rotation to congruence. Psychometrika **31**(1), 33–42 (1966)
14. Cox, D.A., Little, J., O'Shea, D.: Using Algebraic Geometry, 2nd edn. Springer, New York (2005)
15. DeMenthon, D., Davis, L.S.: Exact and approximate solutions of the perspective-three-point problem. IEEE Trans. Pattern Anal. Mach. Intell. **14**(11), 1100–1105 (1992)
16. DeMenthon, D.F., Davis, L.S.: Model-based object pose in 25 lines of code. Int. J. Comput. Vis. **15**(1), 123–141 (1995)
17. Ferraz, L., Binefa, X., Moreno-Noguer, F.: Very fast solution to the PnP problem with algebraic outlier rejection. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 501–508 (2014)
18. Fiore, P.D.: Efficient linear solution of exterior orientation. IEEE Trans. Pattern Anal. Mach. Intell. **23**(2), 140–148 (2001)
19. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981)
20. Fitzgibbon, A.W.: Simultaneous linear estimation of multiple view geometry and lens distortion. In: IEEE Conference on Computer Vision and Pattern Recognition, vol. I, pp. 125–132 (2001)
21. Förstner, W., Wrobel, B.P.: Photogrammetric Computer Vision: Statistics, Geometry, Orientation and Reconstruction. Springer, Cham (2016)
22. Gao, X.S., Hou, X.R., Tang, J., Cheng, H.F.: Complete solution classification for the perspective-three-point problem. IEEE Trans. Pattern Anal. Mach. Intell. **25**(8), 930–943 (2003)
23. Garro, V., Crosilla, F., Fusiello, A.: Solving the PnP problem with anisotropic orthogonal Procrustes analysis. In: Second Joint 3DIM/3DPVT Conference: 3D Imaging, Modeling, Processing, Visualization & Transmission, pp. 262–269 (2012)
24. Golub, G.H., Van Loan, C.F.: Matrix Computations, 4th edn. Johns Hopkins University Press, Baltimore (2013)
25. Gower, J.C., Dijksterhuis, G.B.: Procrustes Problems. Oxford University Press, Oxford (2004)
26. Green, B.F.: The orthogonal approximation of an oblique structure in factor analysis. Psychometrika **17**(4), 429–440 (1952)
27. Haralick, R.M., Joo, H., Lee, C.N., Zhuang, X., Vaidya, V.G., Kim, M.B.: Pose estimation from corresponding point data. IEEE Trans. Syst. Man Cybern. **19**(6), 1426–1446 (1989)
28. Haralick, R.M., Lee, C.N., Ottenberg, K., Nölle, M.: Review and analysis of solutions of the three point perspective pose estimation problem. Int. J. Comput. Vis. **13**(3), 331–356 (1994)
29. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, Cambridge (2003)
30. Henrion, D., Lasserre, J.B.: GloptiPoly: global optimization over polynomials with Matlab and SeDuMi. ACM Trans. Math. Softw. **29**(2), 165–194 (2003)
31. Henrion, D., Lasserre, J.B., Löfberg, J.: GloptiPoly 3: moments, optimization and semidefinite programming. Optim. Methods Softw. **24**(4–5), 761–779 (2009)
32. Hesch, J.A., Roumeliotis, S.I.: A direct least-squares (DLS) method for P*n*P. In: International Conference on Computer Vision, pp. 383–390 (2011)
33. Higham, N.J.: Computing the polar decomposition—with applications. SIAM J. Sci. Stat. Comput. **7**(4), 1160–1174 (1986)
34. Horaud, R., Conio, B., Leboulleux, O., Lacolle, B.: An analytic solution for the perspective 4-point problem. Comput. Vis. Graph. Image Process. **47**(1), 33–44 (1989)
35. Horaud, R., Dornaika, F., Lamiroy, B., Christy, S.: Object pose: the link between weak perspective, paraperspective, and full perspective. Int. J. Comput. Vis. **22**(2), 173–189 (1997)
36. Horn, B.K.P.: Closed-form solution of absolute orientation using unit quaternions. J. Opt. Soc. Am. A **4**(4), 629–642 (1987)
37. Horn, B.K.P., Hilden, H.M., Negahdaripour, S.: Closed-form solution of absolute orientation using orthonormal matrices. J. Opt. Soc. Am. A **5**(7), 1127–1135 (1988)
38. Huttenlocher, D.P., Ullman, S.: Recognizing solid objects by alignment with an image. Int. J. Comput. Vis. **5**(2), 195–212 (1990)
39. Kneip, L., Li, H., Seo, Y.: UPnP: an optimal $O(n)$ solution to the absolute pose problem with universal applicability. In: D. Fleet,

T. Pajdla, B. Schiele, T. Tuytelaars (eds.) 13th European Conference on Computer Vision, Lecture Notes in Computer Science, vol. 8689, pp. 127–142, Springer (2014)
40. Kneip, L., Scaramuzza, D., Siegwart, R.: A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 2969–2976 (2011)
41. Koschat, M.A., Swayne, D.F.: A weighted Procrustes criterion. Psychometrika **56**(2), 229–239 (1991)
42. Kúkelová, Z., Bujnak, M., Pajdla, T.: Automatic generator of minimal problem solvers. In: D. Forsyth, P. Torr, A. Zisserman (eds.) Tenth European Conference on Computer Vision, Lecture Notes in Computer Science, vol. 5304, pp. 302–315, Springer (2008)
43. Lanser, S.: Modellbasierte Lokalisation gestützt auf monokulare Videobilder. Dissertation, Forschungs- und Lehreinheit Informatik IX, Technische Universität München (1997)
44. Lanser, S., Zierl, C., Beutlhauser, R.: Multibildkalibrierung einer CCD-Kamera. In: Sagerer, G., Posch, S., Kummert, F. (eds.) Mustererkennung, Informatik aktuell, pp. 481–491. Springer, Berlin (1995)
45. Lenz, R.: Linsenfehlerkorrigierte Eichung von Halbleiterkameras mit Standardobjektiven für hochgenaue 3D-Messungen in Echtzeit. In: Paulus, E. (ed.) Mustererkennung, Informatik-Fachberichte, vol. 149, pp. 212–216. Springer, Berlin (1987)
46. Lenz, R.: Viedeometrie mit CCD-Sensoren und ihre Anwendung in der Robotik. Lehrstuhl für Nachrichtentechnik der Technischen Universität München, Habilitationsschrift (1988)
47. Lenz, R., Fritsch, D.: Accuracy of videometry with CCD sensors. ISPRS Journal of Photogrammetry and Remote Sensing **45**(2), 90–110 (1990)
48. Lepetit, V., Moreno-Noguer, F., Fua, P.: EPnP: an accurate $O(n)$ solution to the PnP problem. Int. J. Comput. Vis. **81**(2), 155–166 (2009)
49. Li, S., Xu, C., Xie, M.: A robust $O(n)$ solution to the perspective-n-point problem. IEEE Trans. Pattern Anal. Mach. Intell. **34**(7), 1444–1450 (2012)
50. Luenberger, D.G., Ye, Y.: Linear and Nonlinear Programming, 3rd edn. Springer, Berlin (2008)
51. Luhmann, T., Robson, S., Kyle, S., Boehm, J.: Close-Range Photogrammetry and 3D Imaging, 2nd edn. Walter de Gruyter GmbH, Berlin (2014)
52. McGlone, J.C. (ed.): Manual of Photogrammetry, 5th edn. American Society for Photogrammetry and Remote Sensing, Bethesda (2004)
53. Mooijaart, A., Commandeur, J.J.F.: A general solution of the weighted orthonormal Procrustes problem. Psychometrika **44**(4), 657–663 (1990)
54. Nistér, D., Stewénius, H.: A minimal solution to the generalised 3-point pose problem. J. Math. Imaging Vis. **27**(1), 67–79 (2007)
55. Oberkampf, D., DeMenthon, D.F., Davis, L.S.: Iterative pose estimation using coplanar feature points. Comput. Vis. Image Underst. **63**(3), 495–511 (1996)
56. Olsson, C., Kahl, F., Oskarsson, M.: Branch-and-bound methods for euclidean registration problems. IEEE Trans. Pattern Anal. Mach. Intell. **31**(5), 793–794 (2009)
57. Park, H.: A parallel algorithm for the unbalanced orthogonal Procrustes problem. Parallel Comput. **17**(8), 913–923 (1991)
58. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes: The Art of Scientific Computing, 3rd edn. Cambridge University Press, Cambridge (2007)
59. Quan, L., Lan, Z.: Linear n-point camera pose determination. IEEE Trans. Pattern Anal. Mach. Intell. **21**(8), 774–780 (1999)
60. Schönemann, P.H.: A generalized solution of the orthogonal Procrustes problem. Psychometrika **31**(1), 1–10 (1966)
61. Schweighofer, G., Pinz, A.: Robust pose estimation from a planar target. IEEE Trans. Pattern Anal. Mach. Intell. **28**(12), 2024–2030 (2006)
62. Schweighofer, G., Pinz, A.: Globally optimal $O(n)$ solution to the PnP problem for general camera models. In: British Machine Vision Conference (2008)
63. Steger, C.: A comprehensive and versatile camera model for cameras with tilt lenses. Int. J. Comput. Vis. **123**(2), 121–159 (2017)
64. Steger, C., Ulrich, M., Wiedemann, C.: Machine Vision Algorithms and Applications. Wiley-VCH, Weinheim (2008)
65. Stiefel, E.: Richtungsfelder und Fernparallelismus in n-dimensionalen Mannigfaltigkeiten. Commentarii Mathematici Helvetici **8**, 305–353 (1935)
66. ten Berge, J.M.F., Knol, D.L.: Orthogonal rotations to maximal agreement for two or more matrices of different column orders. Psychometrika **49**(1), 49–55 (1984)
67. Triggs, B.: Camera pose and calibration from 4 or 5 known 3D points. In: 7th International Conference on Computer Vision, vol. 1, pp. 278–284 (1999)
68. Umeyama, S.: Least-squares estimation of transformation parameters between two point patterns. IEEE Trans. Pattern Anal. Mach. Intell. **13**(4), 376–380 (1991)
69. Walker, M.W., Shao, L., Volz, R.A.: Estimating 3-D location parameters using dual number quaternions. Comput. Vis. Graph. Image Process. Image Underst. **54**(3), 358–367 (1991)
70. Wu, Y., Hu, Z.: PnP problem revisited. J. Math. Imaging Vis. **24**(1), 131–141 (2006)
71. Zhang, Z., Du, K.: Successive projection method for solving the unbalanced Procrustes problem. Sci. China Ser. A Math. **49**(7), 971–986 (2006)
72. Zheng, Y., Kuang, Y., Sugimoto, S., Åström, K., Okutomi, M.: Revisiting the PnP problem: a fast, general and optimal solution. In: International Conference on Computer Vision, pp. 2344–2351 (2013)

Carsten Steger studied computer science at the Technische Universität München (TUM) and received the PhD degree from TUM in 1998. In 1996, he cofounded the company MVTec Software GmbH, where he heads the Research Department. He has authored and coauthored more than 80 scientific publications in the fields of computer and machine vision, including several textbooks on machine vision. In 2011, he was appointed a TUM adjunct professor for the field of computer vision. He has been a member of the Technical Committee of the German Association for Pattern Recognition (DAGM) since 2013.