Supplementary Material

1 Derivation of $\mathcal{L}, \mathcal{L}_{s1}$ and $\mathcal{L}_{s2}$

1.1 Derivation of $\mathcal{L}(\boldsymbol{q}, \boldsymbol{t})$

According to [1], the Quadratic Pose Estimation Problem addresses the estimation of 6DOF poses under the following constraints:

$$\begin{cases} f(\boldsymbol{q}^4, \lambda) = 0 \\ \boldsymbol{q}^\top \boldsymbol{q} = 1 \\ \boldsymbol{t} = \mathcal{T}(q^2) \end{cases} \tag{1}$$

where $\boldsymbol{t}$ must be expressible as a linear combination of $\boldsymbol{q}^2$. This necessitates that the partial derivatives of $\boldsymbol{t}$ only contain terms up to $\boldsymbol{t}^2$ to avoid introducing terms of $\boldsymbol{q}^5$. Similarly, $\boldsymbol{t}$ can only be directly multiplied by $\boldsymbol{q}^2$. After addressing the constraints on $\boldsymbol{q}$, we consider that the presence of $\boldsymbol{q}^4$ does not alter the definition provided in (1). Since $\boldsymbol{q}$ represents a rotation matrix, there are no odd powers of $\boldsymbol{q}$. Combining these terms, the variables of our function include $\boldsymbol{q}^4$, $\boldsymbol{q}^2\boldsymbol{t}$, $\boldsymbol{t}$, and $\boldsymbol{t}^2$, potentially with varying coefficients and additional constants. Thus, the expression for $\mathcal{L}(\boldsymbol{q}, \boldsymbol{t})$ is given by:

$$\mathcal{L}(\boldsymbol{q}, \boldsymbol{t}) = \boldsymbol{V}_{35}^\top \boldsymbol{q}^4 + \boldsymbol{V}_{40}^\top \mathrm{vec}(\boldsymbol{q}^2 \otimes \tilde{\boldsymbol{t}}) + \boldsymbol{V}_6^\top \boldsymbol{t}^2 + \boldsymbol{V}_3^\top \boldsymbol{t} + C$$

1.2 Derivation of $\mathcal{L}_{s1}$

Regarding the addition of a scale factor to the pose's translation vector, if we substitute $\boldsymbol{t}$ with $\boldsymbol{t}^* = \boldsymbol{t} \cdot s$, the problem formulation remains unchanged, indicating that scale is not an independent variable to be solved in this scenario.

For scenarios involving sensors with unknown scale (e.g., monocular cameras), the world coordinates lack scale and are represented in a homogeneous 4×1 vector format, where the fourth dimension represents the scale factor s. Homogeneous coordinates maintain their linear characteristics after transformations by any 4×4 matrix, hence the scale information retains similar properties to other translation components: $s = \mathcal{T}_s(\boldsymbol{q}^2)$. Incorporating scale into (1.1) while maintaining the highest power not exceeding four, we ensure the inheritance of the QPEP properties, leading to:

$$\mathcal{L}_{s1}(\boldsymbol{q}, \boldsymbol{t}, s) = \boldsymbol{V}_{35}^\top \boldsymbol{q}^4 + \boldsymbol{V}_{40}^\top \mathrm{vec}(\boldsymbol{q}^2 \otimes \tilde{\boldsymbol{t}}) + s \cdot \boldsymbol{V}_{10}^\top \boldsymbol{q}^2 + \boldsymbol{V}_6^\top \boldsymbol{t}^2 + s \cdot \boldsymbol{V}_4^\top \boldsymbol{t_s} + \boldsymbol{V}_4^\top \boldsymbol{t_s} + C$$

1.3 Derivation of $\mathcal{L}_{s2}$

If the scale factor is incorporated into the rotation matrix, transforming it into a scaled rotation matrix, this change has substantial physical implications. As previously discussed, since $\boldsymbol{q}$ is derived from a rotation matrix, incorporating a scale factor s affects every occurrence of $\boldsymbol{q}^2$. The derivation, based on (1), involves multiplying each term involving $\boldsymbol{q}^2$ by s, resulting in:

$$\mathcal{L}_{s2}(\boldsymbol{q}, \boldsymbol{t}, s) = s^2 \cdot \boldsymbol{V}_{35}^\top \boldsymbol{q}^4 + s \cdot \boldsymbol{V}_{40}^\top \mathrm{vec}(\boldsymbol{q}^2 \otimes \tilde{\boldsymbol{t}}) + \boldsymbol{V}_6^\top \boldsymbol{t}^2 + \boldsymbol{V}_3^\top \boldsymbol{t} + C.$$

2 Example : Using sQPEP for Hand-Eye Calibration

2.1 Back Ground

In the fields of robotics, computer vision, and graphics, a pose matrix is used to describe an object's position (translation) and orientation (rotation) in three-dimensional space. Since a pose encompasses 6 degrees of freedom (DOF), it requires at least six parameters for representation. Various expressions exist for rotation, including rotation matrices, quaternions, and angle-axes. If expressed using a rotation

matrix $\boldsymbol{R}_{3\times3}$, this matrix must satisfy $\boldsymbol{R}^\top\boldsymbol{R} = \boldsymbol{I}$. Alternatively, if expressed using quaternions, it must satisfy $\boldsymbol{q}^\top\boldsymbol{q} = 1$. Quaternion representation offers the most compact expression for rotations and can naturally be converted into a rotation matrix form.

Pose estimation is often required for several reasons, such as determining the current sensor's position and orientation relative to a specific coordinate frame. This determination is commonly referred to as "pose." Depending on the application, poses are stored using various minimal representations. A common minimal representation uses the parameters $q_w, q_x, q_y, q_z, t_x, t_y, t_z$. Here, $\boldsymbol{q} = [q_w, q_x, q_y, q_z]^\top$ represents the quaternion, which inherently facilitates conversion to and from rotation matrices. $\boldsymbol{t} = [t_x, t_y, t_z]$ denotes the translation components. For further details on quaternion properties, see [2]. It is important to note that although quaternions involve four variables, they only represent 3 DOF due to the inherent constraint $\boldsymbol{q}^\top\boldsymbol{q} = 1$. In most domains, this 6DOF pose information is expressed using the transformation matrix $\boldsymbol{X}$:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{R}_{3\times3} & \boldsymbol{t}_{3\times1} \\ \boldsymbol{0}_{3\times1} & 1 \end{bmatrix}_{4\times4} = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2q_wq_z + 2q_xq_y & 2q_xq_z - 2q_wq_y & t_x \\ 2q_xq_y - 2q_wq_z & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2q_wq_x + 2q_yq_z & t_y \\ 2q_wq_y + 2q_xq_z & 2q_yq_z - 2q_wq_x & q_w^2 - q_x^2 - q_y^2 + q_z^2 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\boldsymbol{R}$ denotes the rotation matrix, and $\boldsymbol{t}$ denotes the translation vector.

2.2 Hand-eye calibration

Hand-Eye Calibration originally derived from estimating the pose relationship between a camera and the robot end-effector, hand-eye calibration is now commonly applied to various sensor pair calibrations such as LiDAR-Camera, IMU-Camera, and GNSS-IMU-Camera systems. The hand-eye calibration problem is often formulated as the $\boldsymbol{AX} = \boldsymbol{XB}$ problem, where $\boldsymbol{X}$ represents the pose relationship that needs to be estimated between two sensors. Figure 1 provides further details on the notation used.

Readers may notice that in our figures, the matrix is not simply denoted as $\boldsymbol{A}$ but as $\boldsymbol{A}(s)$. This notation highlights the scale ambiguity encountered when calibration patterns do not provide scale information. In monocular camera systems, pose trajectories derived from feature-based methods, optical flow, etc., lack scale information. Consequently, all collected data are represented in the form $\boldsymbol{A}$ or $\boldsymbol{A}(s)$, reflecting this inherent uncertainty.

$$\boldsymbol{A} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{A}(s) = \begin{bmatrix} A_{11} & A_{12} & A_{13} & s \cdot A_{14} \\ A_{21} & A_{22} & A_{23} & s \cdot A_{24} \\ A_{31} & A_{32} & A_{33} & s \cdot A_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix $bmA$is also a transformation matrix, so the 9 elements in the upper left form a rotation matrix.

$$\boldsymbol{R}_A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}, \quad \text{where } \boldsymbol{R}_A^\top\boldsymbol{R}_A = \boldsymbol{1}_{3\times3}, \; \textit{and} \; \det(\boldsymbol{R}_A = 1)$$

In a similar manner, matrix $\boldsymbol{B}$ is derived from motion trajectories captured by scaled sensors. Commonly in industrial robots, this is achieved through command inputs that determine variations in the end-effector's position. In other contexts, such as wheel or GNSS sensors, changes are tracked over time. The use of transformation matrices is prevalent in the context of hand-eye calibration, represented as:

$$\boldsymbol{B} = \begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ B_{31} & B_{32} & B_{33} & B_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

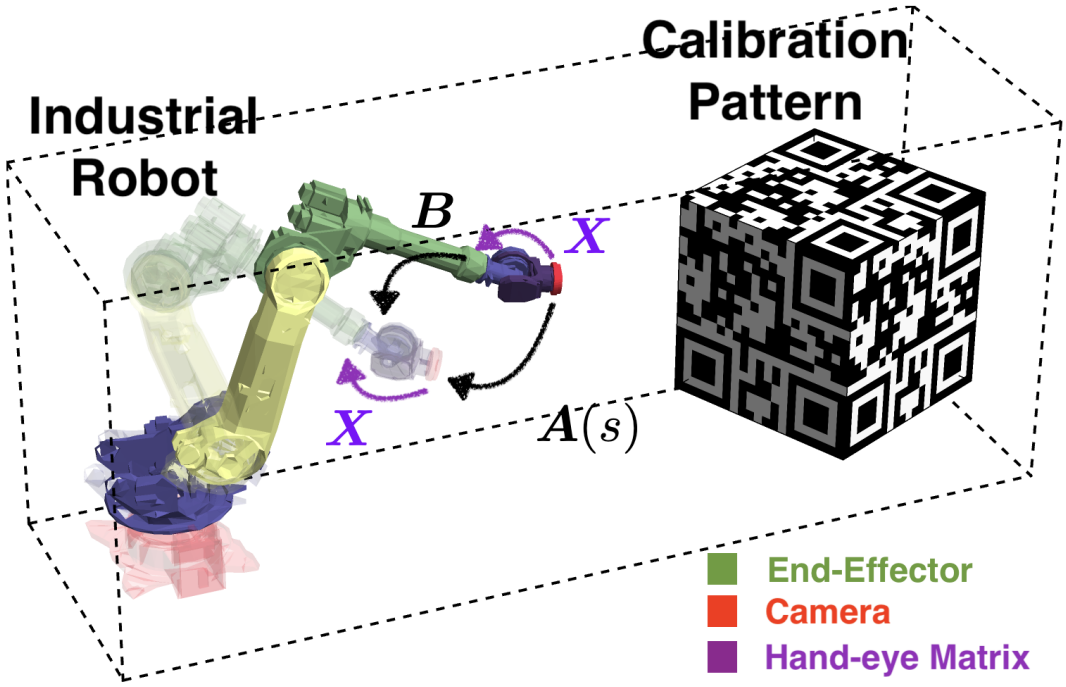

Figure 1: What is hand-eye calibration? In this context, $\boldsymbol{A}(s)$ or $\boldsymbol{A}$ represents the camera's displacement in the world, typically expressed using a transformation matrix. $\boldsymbol{B}$ denotes the displacement of the Robot End-Effector in the world, also expressed using a transformation matrix. $\boldsymbol{X}$ defines the pose relationship between the camera and the Robot End-Effector, which is a constant and the target of our estimation. $\boldsymbol{A}(s)$ represents the camera's motion trajectory, with an unknown scale factor. If the calibration pattern does not provide effective scale information (such as edge lengths), the scale s must be estimated. This implies that in such cases, $\boldsymbol{A}(s)$ is used to express the camera's motion trajectory. For a more detailed explanation, see Supplement B.

Similarly, we define:

$$\boldsymbol{R}_B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix}, \quad \text{where } \boldsymbol{R}_B^\top \boldsymbol{R}_B = \mathbf{1}_{3\times 3}, \ \textit{and} \ \det(\boldsymbol{R}_B = 1).$$

The focus of this manuscript is on scaled problems, thus the primary equation of interest is $\boldsymbol{A}(s)\boldsymbol{X} = \boldsymbol{X}\boldsymbol{B}$, rather than $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{X}\boldsymbol{B}$. It is important to note that in the development of hand-eye calibration, the definition of residuals varies, influencing problem definitions, solution methods, and accuracy across different datasets. Here, we employ the Frobenius norm, also known as the Euclidean norm, to define the problem's residuals:

$$\hat{\boldsymbol{q}}, \hat{\boldsymbol{t}}, \hat{s} = \arg\min_{\boldsymbol{q},\boldsymbol{t},s} \sum ||\boldsymbol{A}(\boldsymbol{s})_{\boldsymbol{i}}\boldsymbol{X} - \boldsymbol{X}\boldsymbol{B}_{\boldsymbol{i}}||$$

$$||\boldsymbol{A}(\boldsymbol{s})_{\boldsymbol{i}}\boldsymbol{X} - \boldsymbol{X}\boldsymbol{B}_{\boldsymbol{i}}|| = \sum \operatorname{trace}(\boldsymbol{J}_i^\top \boldsymbol{J}_i), \ \text{where } \boldsymbol{J}_i = \boldsymbol{A}(\boldsymbol{s})_{\boldsymbol{i}}\boldsymbol{X} - \boldsymbol{X}\boldsymbol{B}_{\boldsymbol{i}}.$$

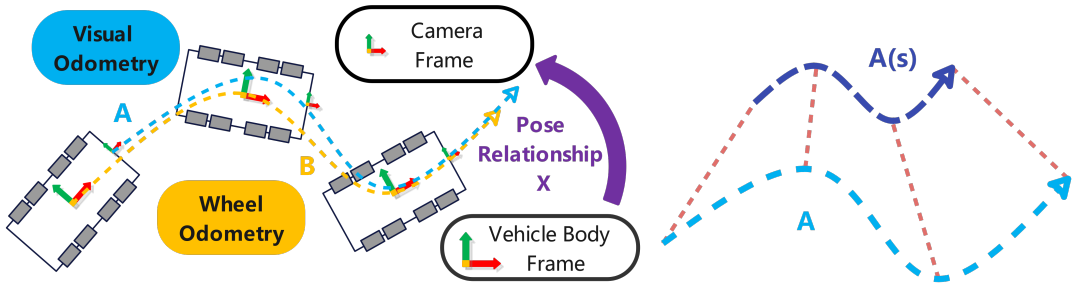

Figure 2: We address another variant of the hand-eye calibration problem, pertinent to vehicles equipped with a monocular camera. The camera's trajectory, depicted in blue and corresponding to $\boldsymbol{A}$, and the vehicle's center, derived from wheel odometry and depicted in orange corresponding to $\boldsymbol{B}$, require pose estimation between them. Unlike the scenario illustrated in Fig. 1, the vehicle may not have access to a pre-prepared calibration pattern. Operational vehicles in external environments typically rely on alternative methods to deduce and acquire scale-free information of $\boldsymbol{A}(s)$.

For clarity, $\boldsymbol{J}_i$ represents the residual term for the i-th trajectory. Expanding $\boldsymbol{J}_i$ yields:

$$\boldsymbol{J}_i = \boldsymbol{A(s)_i X} - \boldsymbol{X B_i} =$$

$$\begin{bmatrix} {}^iA_{11} & {}^iA_{12} & {}^iA_{13} & s\cdot{}^iA_{14} \\ {}^iA_{21} & {}^iA_{22} & {}^iA_{23} & s\cdot{}^iA_{24} \\ {}^iA_{31} & {}^iA_{32} & {}^iA_{33} & s\cdot{}^iA_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_w^2+q_x^2-q_y^2-q_z^2 & 2q_wq_z+2q_xq_y & 2q_xq_z-2q_wq_y & t_x \\ 2q_xq_y-2q_wq_z & q_w^2-q_x^2+q_y^2-q_z^2 & 2q_wq_x+2q_yq_z & t_y \\ 2q_wq_y+2q_xq_z & 2q_yq_z-2q_wq_x & q_w^2-q_x^2-q_y^2+q_z^2 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} -$$

$$\begin{bmatrix} q_w^2+q_x^2-q_y^2-q_z^2 & 2q_wq_z+2q_xq_y & 2q_xq_z-2q_wq_y & t_x \\ 2q_xq_y-2q_wq_z & q_w^2-q_x^2+q_y^2-q_z^2 & 2q_wq_x+2q_yq_z & t_y \\ 2q_wq_y+2q_xq_z & 2q_yq_z-2q_wq_x & q_w^2-q_x^2-q_y^2+q_z^2 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^iB_{11} & {}^iB_{12} & {}^iB_{13} & {}^iB_{14} \\ {}^iB_{21} & {}^iB_{22} & {}^iB_{23} & {}^iB_{24} \\ {}^iB_{31} & {}^iB_{32} & {}^iB_{33} & {}^iB_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Here, ${}^iA_{jk}$ and ${}^iB_{jk}$ are constants derived from various scenarios usually obtained through methods like SLAM or SFM, or directly from sensor readings. We input these values into $\boldsymbol{J}_i^\top \boldsymbol{J}_i$, focusing primarily on the trace of the diagonal elements. The calculations can be complex and are typically performed using MATLAB.

```
syms A11 A12 A13 A14 real
syms A21 A22 A23 A24 real
syms A31 A32 A33 A34 real
syms B11 B12 B13 B14 real
syms B21 B22 B23 B24 real
syms B31 B32 B33 B34 real
syms q0 q1 q2 q3 real
syms t1 t2 t3  real
syms scale real
A = [A11, A12, A13, scale*A14;
    A21, A22, A23, scale*A24;
    A31, A32, A33, scale*A34;
      0,   0,   0,   1;];
B = [B11, B12, B13, B14;
    B21, B22, B23, B24;
    B31, B32, B33, B34;
      0,   0,   0,   1;];
```

```
q = [q0; q1; q2; q3];
t = [t1; t2; t3];
R = q2R(q);
X = [R, t; zeros(1, 3), 1];
cost_i = trace((A*X-X*B).' * (A*X-X*B))
function R = q2R(q)
q0 = q(1); q1 = q(2); q2 = q(3); q3 = q(4);
R = [ q0^2 + q1^2 - q2^2 - q3^2, 2*q0*q3 + 2*q1*q2,  2*q1*q3 - 2*q0*q2;
      2*q1*q2 - 2*q0*q3, q0^2 - q1^2 + q2^2 - q3^2,  2*q0*q1 + 2*q2*q3;
      2*q0*q2 + 2*q1*q3,  2*q2*q3 - 2*q0*q1, q0^2 - q1^2 - q2^2 + q3^2];
end
```

Through the above code, we can obtain

$||\boldsymbol{A(s)_i X - X B_i}|| = (A_{11}(q_w^2+q_x^2-q_y^2-q_z^2)-B_{11}(q_w^2+q_x^2-q_y^2-q_z^2)-A_{12}(2q_wq_z-2q_xq_y)+A_{13}(2q_wq_y+2q_xq_z)-B_{21}(2q_wq_z+2q_xq_y)+B_{31}(2q_wq_y-2q_xq_z))^2+(A_{12}(q_w^2-q_x^2+q_y^2-q_z^2)-B_{12}(q_w^2+q_x^2-q_y^2-q_z^2)+A_{11}(2q_wq_z+2q_xq_y)-A_{13}(2q_wq_x-2q_yq_z)-B_{22}(2q_wq_z+2q_xq_y)+B_{32}(2q_wq_y-2q_xq_z))^2+(A_{13}(q_w^2-q_x^2-q_y^2+q_z^2)-B_{13}(q_w^2+q_x^2-q_y^2-q_z^2)-A_{11}(2q_wq_y-2q_xq_z)+A_{12}(2q_wq_x+2q_yq_z)-B_{23}(2q_wq_z+2q_xq_y)+B_{33}(2q_wq_y-2q_xq_z))^2+(A_{21}(q_w^2+q_x^2-q_y^2-q_z^2)-B_{21}(q_w^2-q_x^2+q_y^2-q_z^2)-A_{22}(2q_wq_z-2q_xq_y)+A_{23}(2q_wq_y+2q_xq_z)+B_{11}(2q_wq_z-2q_xq_y)-B_{31}(2q_wq_x+2q_yq_z))^2+(A_{22}(q_w^2-q_x^2+q_y^2-q_z^2)-B_{22}(q_w^2-q_x^2+q_y^2-q_z^2)+A_{21}(2q_wq_z+2q_xq_y)-A_{23}(2q_wq_x-2q_yq_z)+B_{12}(2q_wq_z-2q_xq_y)-B_{32}(2q_wq_x+2q_yq_z))^2+(A_{23}(q_w^2-q_x^2-q_y^2+q_z^2)-B_{23}(q_w^2-q_x^2+q_y^2-q_z^2)-A_{21}(2q_wq_y-2q_xq_z)+A_{22}(2q_wq_x+2q_yq_z)+B_{13}(2q_wq_z-2q_xq_y)-B_{33}(2q_wq_x+2q_yq_z))^2+(A_{31}(q_w^2+q_x^2-q_y^2-q_z^2)-B_{31}(q_w^2-q_x^2-q_y^2+q_z^2)-A_{32}(2q_wq_z-2q_xq_y)+A_{33}(2q_wq_y+2q_xq_z)-B_{11}(2q_wq_y+2q_xq_z)+B_{21}(2q_wq_x-2q_yq_z))^2+(A_{32}(q_w^2-q_x^2+q_y^2-q_z^2)-B_{32}(q_w^2-q_x^2-q_y^2+q_z^2)+A_{31}(2q_wq_z+2q_xq_y)-A_{33}(2q_wq_x-2q_yq_z)-B_{12}(2q_wq_y+2q_xq_z)+B_{22}(2q_wq_x-2q_yq_z))^2+(A_{33}(q_w^2-q_x^2-q_y^2+q_z^2)-B_{33}(q_w^2-q_x^2-q_y^2+q_z^2)-A_{31}(2q_wq_y-2q_xq_z)+A_{32}(2q_wq_x+2q_yq_z)-B_{13}(2q_wq_y+2q_xq_z)+B_{23}(2q_wq_x-2q_yq_z))^2+(A_{14}s-B_{14}(q_w^2+q_x^2-q_y^2-q_z^2)-t_x+A_{11}t_x+A_{12}t_y+A_{13}t_z-B_{24}(2q_wq_z+2q_xq_y)+B_{34}(2q_wq_y-2q_xq_z))^2+(A_{24}s-B_{24}(q_w^2-q_x^2+q_y^2-q_z^2)-t_y+A_{21}t_x+A_{22}t_y+A_{23}t_z+B_{14}(2q_wq_z-2q_xq_y)-B_{34}(2q_wq_x+2q_yq_z))^2+(A_{34}s-B_{34}(q_w^2-q_x^2-q_y^2+q_z^2)-t_z+A_{31}t_x+A_{32}t_y+A_{33}t_z-B_{14}(2q_wq_y+2q_xq_z)+B_{24}(2q_wq_x-2q_yq_z))^2$

In the provided script, the left superscripts i on A and B are omitted for simplicity. The equation in question forms a multivariate polynomial in terms of $\boldsymbol{q}, \boldsymbol{t}$, and s. Utilizing MATLAB, we extract the terms of this polynomial as follows:

```
[C, T]= coeffs(cost_i ,[q;t;scale]);
display(T.')
```

We can obtain the set $\mathcal{T}$:

$\mathcal{T} = \{q_w^4, q_w^3q_x, q_w^3q_y, q_w^3q_z, q_w^2q_x^2, q_w^2q_xq_y, q_w^2q_xq_z, q_w^2q_y^2, q_w^2q_yq_z, q_w^2q_z^2, q_w^2t_x,$
$q_w^2t_y, q_w^2t_z, q_w^2s, q_wq_x^3, q_wq_x^2q_y, q_wq_x^2q_z, q_wq_xq_y^2, q_wq_xq_yq_z, q_wq_xq_z^2, q_wq_xt_x, q_wq_xt_y, q_wq_xt_z, q_wq_xs,$
$q_wq_y^3, q_wq_y^2q_z, q_wq_yq_z^2, q_wq_yt_x, q_wq_yt_y, q_wq_yt_z, q_wq_ys, q_wq_z^3, q_wq_zt_x, q_wq_zt_y, q_wq_zt_z, q_wq_zs, q_x^4,$
$q_x^3q_y, q_x^3q_z, q_x^2q_y^2, q_x^2q_yq_z, q_x^2q_z^2, q_x^2t_x, q_x^2t_y, q_x^2t_z, q_x^2s, q_xq_y^3, q_xq_y^2q_z, q_xq_yq_z^2,$
$q_xq_yt_x, q_xq_yt_y, q_xq_yt_z, q_xq_ys, q_xq_z^3, q_xq_zt_x, q_xq_zt_y, q_xq_zt_z, q_xq_zs, q_y^4, q_y^3q_z, q_y^2q_z^2, q_y^2t_x, q_y^2t_y,$
$q_y^2t_z, q_y^2s, q_yq_z^3, q_yq_zt_x, q_yq_zt_y, q_yq_zt_z, q_yq_zs, q_z^4, q_z^2t_x, q_z^2t_y, q_z^2t_z, q_z^2s, t_x^2, t_xt_y, t_xt_z, st_x, t_y^2, t_yt_z, st_y, t_z^2, st_z, s^2\}$

This means we can obtain another form of expression

$$||\boldsymbol{A(s)_i X - X B_i}|| = \mathcal{P}(\boldsymbol{A}(s)_i, \boldsymbol{B}_i, \mathcal{T})$$
$$\hookrightarrow L = \sum \mathcal{P}(\boldsymbol{A}(s)_i, \boldsymbol{B}_i, \mathcal{T})$$

Here L denotes the residual of the entire problem. $\mathcal{P}$ represents the polynomial function that transforms the input of the i-th constant into the i-th residual. It is evident that the addition of polynomials

with identical terms does not alter the terms of the polynomial. This implies that the terms of L are all contained within $\mathcal{T}$.

By conducting a one-by-one check or utilizing any available tool, we can ascertain that

$$\mathcal{T} \subseteq \boldsymbol{q}^4 \cup \boldsymbol{q}^2 \otimes \tilde{\boldsymbol{t}} \cup s\boldsymbol{q}^2 \cup \boldsymbol{t}^2 \cup s\boldsymbol{t}_{\boldsymbol{s}} \cup \boldsymbol{t}_{\boldsymbol{s}},$$

where all terms of the polynomial are subsets of the terms of polynomial in $\mathcal{L}_{s1}$. This means that for terms not present in $\mathcal{L}_{s1}$, we can assign a value of zero, thus transforming the residual L into the form of $\mathcal{L}_{s1}$. Consequently, the problem can be solved using the method provided in the manuscript.

3 Example : using sQPEP for point-to-plane problem

3.1 Background

The objective of the point-to-plane problem is to measure the perpendicular distances from each point to a plane and potentially use these distances to adjust the plane's position or orientation, thereby optimizing the collection of distances (e.g., the average or sum) in some manner. For example, during surface inspection by a robotic arm, adjustments can be made to the robot's position by comparing the distances between the measured point set and a theoretical plane model, enabling more precise alignment or manipulation of the object. Additionally, in point cloud registration tasks, normal vectors of one point cloud are calculated to transform the problem into a point-to-plane issue, which is particularly effective when the point cloud possesses significant planar properties.

Specifically, the task involves estimating a pose state—rotation and translation (6DOF)—to achieve alignment between the point cloud and the plane. However, sometimes the algorithms/sensors used to acquire point clouds may not provide accurate scale, or may even be scale-free. This necessitates the estimation of a scale factor to scale the original point cloud for better fitting with the target. Figure **??** serves as an example to help understand the objectives that need to be addressed in this issue.

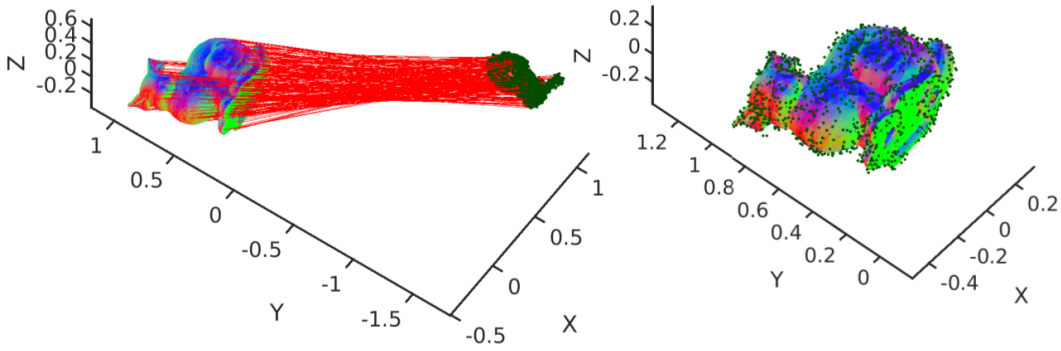

Figure 3: What is Point-to-Plane? Left: Original state of point cloud and associated surfaces. The **red lines** indicate the correspondences between matched points, while the **multi-colored surfaces** denote the directions of different surface normals. The **dark green** points represent the points that are to be matched. Right: Resultant configuration after computation with our solver, showcasing the alignment of point correspondences and the orientation of surface normals. We aim to estimate the pose relationship between the dark green and the multicolored rabbits, characterized by rotation $\boldsymbol{R}$, translation $\boldsymbol{t}$, and scale factor s. By enlarging the point cloud of the original dark green rabbit, followed by rotation and translation, alignment with the multicolored rabbit is achieved. The image on the right is an example where the points of the dark green rabbit closely match the multicolored rabbit.

The objective function for the given problem is expressed as:

$$\arg\min_{\boldsymbol{R},\boldsymbol{t},s} = \sum_{i=1}^{\mathcal{N}} (\boldsymbol{n}_i^\top (s\boldsymbol{R}\boldsymbol{x}_i + \boldsymbol{t} - \boldsymbol{y}_i))^2 \tag{2}$$

where $\boldsymbol{R}$, $\boldsymbol{t}$, and s are the variables to be estimated. Here, the i-th point $\boldsymbol{x}_i$ corresponds to the i-th plane, and the residual formula represents the sum of squared distances from each point to each plane. It is important to note in our manuscript that all $\boldsymbol{R}$ can be converted between $\boldsymbol{q}$.

It is particularly noteworthy, similar to the hand-eye problem, that the definition of residuals for the point-to-plane problem is not unique; different residual models are suitable for tasks or data with varying characteristics. We provide one common residual model and apply it within our proposed sQPEP framework for solution.

3.2 Problem Resolution

Simply put, the problem conversion can be achieved using the same method as in Supplementary Material 2, except that the residual equation is modified to $\sum_{i=1}^{\mathcal{N}} (\boldsymbol{n}_i (s\boldsymbol{R}\boldsymbol{x}_i + \boldsymbol{t} - \boldsymbol{y}i))^2$. This allows the transformation of the problem based on this residual into the $\mathcal{L}s2$ problem, thus facilitating solution using our method. In this problem, $\boldsymbol{n}_i$ represents the normal vector of the i-th plane, and $\boldsymbol{y}_i$ is a point on the i-th plane. $\boldsymbol{x}$ refers to scale-free point cloud information obtained from a sensor. With given matching relations, minimizing this residual equation essentially estimates the pose information of the sensor.

Our approach continues to involve expressing the residual equation using the estimated parameters $\boldsymbol{q}$, $\boldsymbol{t}$, and s in polynomial form, observing whether the terms of the polynomial are subsets of $\mathcal{L}_{s2}$, thereby transforming the problem accordingly. The code for this is nearly identical to that used in the hand-eye problem.

We provide code here to obtain the $\mathcal{T}$corresponding to the problem:

```
syms s real
syms q_w q_x q_y q_z real
syms t1 t2 t3 lambda real
syms pa_1 pa_2 pa_3 real
syms pb_1 pb_2 pb_3 real
syms nv_1 nv_2 nv_3 real
r=[pa_1; pa_2; pa_3];
b=[pb_1; pb_2; pb_3];
NV=[nv_1; nv_2; nv_3];
q = [ q_w; q_x; q_y; q_z];
t = [t_x; t_y; t_z];
tts=[t_x; t_y; t_z;s];
R = q2R(q);
cost_i = (NV.' * (s * R * r + t - b))^2;
[C, T]= coeffs(cost_i,[q;t;scale]);
display(T.')
```

Through the above code, we can obtain

$$\begin{aligned}
\mathcal{T} = \{&q_w^4 s^2, q_w^3 q_x s^2, q_w^3 q_y s^2, q_w^3 q_z s^2, q_w^2 q_x^2 s^2, q_w^2 q_x q_y s^2, q_w^2 q_x q_z s^2, q_w^2 q_y^2 s^2, q_w^2 q_y q_z s^2, \\
&q_w^2 q_z^2 s^2, q_w^2 s t_x, q_w^2 s t_y, q_w^2 s t_z, q_w^2 s, q_w q_x^3 s^2, q_w q_x^2 q_y s^2, q_w q_x^2 q_z s^2, \\
&q_w q_x q_y^2 s^2, q_w q_x q_y q_z s^2, q_w q_x q_z^2 s^2, q_w q_x s t_x, q_w q_x s t_y, q_w q_x s t_z, q_w q_x s, q_w q_y^3 s^2, \\
&q_w q_y^2 q_z s^2, q_w q_y q_z^2 s^2, q_w q_y s t_x, q_w q_y s t_y, q_w q_y s t_z, q_w q_y s, q_w q_z^3 s^2, q_w q_z s t_x, \\
&q_w q_z s t_y, q_w q_z s t_z, q_w q_z s, q_x^4 s^2, q_x^3 q_y s^2, q_x^3 q_z s^2, q_x^2 q_y^2 s^2, q_x^2 q_y q_z s^2, q_x^2 q_z^2 s^2, \\
&q_x^2 s t_x, q_x^2 s t_y, q_x^2 s t_z, q_x^2 s, q_x q_y^3 s^2, q_x q_y^2 q_z s^2, q_x q_y q_z^2 s^2, q_x q_y s t_x, q_x q_y s t_y, \\
&q_x q_y s t_z, q_x q_y s, q_x q_z^3 s^2, q_x q_z s t_x, q_x q_z s t_y, q_x q_z s t_z, q_x q_z s, q_y^4 s^2, q_y^3 q_z s^2, q_y^2 q_z^2 s^2, \\
&q_y^2 s t_x, q_y^2 s t_y, q_y^2 s t_z, q_y^2 s, q_y q_z^3 s^2, q_y q_z s t_x, q_y q_z s t_y, q_y q_z s t_z, q_y q_z s, \\
&q_z^4 s^2, q_z^2 s t_x, q_z^2 s t_y, q_z^2 s t_z, q_z^2 s, t_x^2, t_x t_y, t_x t_z, t_x, t_y^2, t_y t_z, t_y, t_z^2, t_z, 1\}
\end{aligned}$$

it is easy to get

$$\mathcal{T} \subseteq s^2 \boldsymbol{q}^4 \cup s \cdot \mathrm{vec}(\boldsymbol{q}^2 \otimes \tilde{\boldsymbol{t}}) \cup \boldsymbol{t}^2 \cup \boldsymbol{t}$$

Thus, the problem is transformed into $\mathcal{L}_{s2}$.

4 How to use Elimination Template

This summary aims to provide an overview for readers unfamiliar with the field, illustrating the basic approach to generating elimination templates using Gröbner bases. We present a straightforward example, employing pre-computed elimination templates derived from [3], enabling engineers who may not be deeply versed in the underlying principles to quickly grasp the methodology. We strive to minimize the discussion of complex theories. The solver code is now open-sourced and available at https://github.com/byronsit/sQPEP_Solver for use.

Briefly, utilizing Gröbner bases to solve systems of multivariate polynomials is a classical approach, whose details can be referred to in [4] and [3]. In essence, we first compute the Gröbner basis of the system, which consists of a set of polynomials that possess special properties, generating the same ideal as the original system of equations. Subsequently, we select a set of "favorable" monomials as a basis and compute the representation of each basis element under the action of each polynomial in the Gröbner basis. This process yields a "multiplication matrix," delineating the multiplicative relations among the basis elements. We then simplify each polynomial in the original system, expressing the simplified results as linear combinations of the basis elements. If the multiplication matrix is "perfect," this linear combination should exactly correspond to the original polynomial; however, there are typically some "error" terms. These error terms are expressed as linear combinations of the original equations, with the coefficients forming an "elimination matrix." This matrix provides a set of new equations that consistently hold within the ideal generated by the original system. Integrating these new equations into the original system results in an "elimination template," which can be used to streamline and expedite subsequent computations.

The key to this process lies in exploiting the properties of the Gröbner basis to transform the original problem into a more tractable "linearized" problem algebraically. By introducing the multiplication matrix and elimination matrix, we establish a linkage between the original and new equations, enabling the use of the new equations to "eliminate" certain terms in the original equations, thus simplifying the solving process.

The fundamental concept here is the use of Gröbner bases to convert nonlinear problems into linear ones, thereby facilitating computation. While the actual computational process involves many technical details and optimization strategies, the central idea remains the conversion of nonlinear issues into linear ones through the use of Gröbner bases.

4.1 An easy example

Here is a simple example to facilitate understanding: For the system of equations:

$$c_{11}x^2 + c_{12}xy + c_{13} = 0$$
$$c_{21}y^2 + c_{22}xy + c_{23} = 0$$

We can use the method of [3] to obtain its elimination template:

$$\boldsymbol{C_0} = \begin{pmatrix} c_{11} & c_{12} & 0 & 0 \\ 0 & 0 & c_{22} & 0 \\ 0 & c_{22} & 0 & c_{21} \\ c_{22} & c_{21} & 0 & 0 \end{pmatrix} \quad \boldsymbol{C_1} = \begin{pmatrix} 0 & 0 & c_{13} & 0 \\ c_{23} & 0 & 0 & c_{21} \\ 0 & 0 & c_{23} & 0 \\ 0 & c_{23} & 0 & 0 \end{pmatrix}$$

we need to solve$\boldsymbol{C_0^\top}\ \boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x}$,where

$$\boldsymbol{b} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}$$

then get

$$\boldsymbol{x} = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \\ p_{31} & p_{32} \\ p_{41} & p_{42} \end{pmatrix}$$

then get

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{x}^\top \boldsymbol{C_1} \\ \mathbf{1}_{4\times 4} \end{pmatrix}_{5-1-6-2}$$

The process involves reordering specific rows of a matrix to form a new matrix $\boldsymbol{A}$. Specifically, rows 5, 1, 6, and 2 are sequentially extracted and arranged to construct $\boldsymbol{A}$. For each eigenpair $(\lambda_i, \mathbf{v}_i)$ of $\boldsymbol{A}$, Normalize $\boldsymbol{v}_i$ to $\tilde{\boldsymbol{v}}_i$ such that $v_{i1} = 1$, $\boldsymbol{r}_i = \begin{bmatrix} \tilde{v}_{i2} \\ \lambda_i \end{bmatrix}$, where $\boldsymbol{r}_i$ represents the solution corresponding to the i^{th} eigenvalue.

We then use$\boldsymbol{c}\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ into the elimination template, four solutions are derived, which include two complex solutions $x = 1.1714i$, $y = 0.6947i$ and $x = -1.1714i$, $y = -0.6947i$, along with two real solutions $x = -2.0910$, $y = -2.0910$, and $x = 2.0910$, $y = 2.0910$.

4.2 Example : point-to-plane problem

Similarly, by employing the elimination template methodology from [3], the process remains consistent though matrix dimensions differ. In this context, matrix $\boldsymbol{C_0}$ is of size 521×521, and $\boldsymbol{C_1}$ is 521×81. The matrix $\boldsymbol{b}$ to be solved is 521×19, where 19 elements are set to -1 at coordinates $(502 + i, i)$ for $i = 1$ to 19, with all other elements being zero. The matrix $\boldsymbol{A}$ has dimensions 81×81, resulting in 81 solutions.

The implementation will be further elucidated through code, which will detail the construction of the residual matrix.

```
load example.mat
addpath('pTop')
r0 = r;
b0 = b;
len = size(nvr, 1);
```

```
coef = double(0);
nvr = double(nvr);
for i = 1 : len
   rr = double(r0(i, :).');
   bb = double(b0(i, :).');
   coef = coef + double(1) / len * coef_J_pure_pTop_sQPEP_merge_s(rr, bb, nvr(i, :
end
```

Utilizing the predefined function, the residuals are transformed into the form of $\mathcal{L}s2$, the underlying principles of which have been previously outlined in the Supplementary Material. This yields the matrices W and Q as described in $\mathcal{L}s2$ in the original manuscript.

$$\tilde{\boldsymbol{W}}_{4\times 64}\cdot \mathrm{vec}(\tilde{\boldsymbol{q}}^{\otimes 3})_{64\times 1} = \boldsymbol{Q}_{4\times 4}\tilde{\boldsymbol{q}}$$

```
coef_J = coef;
coef_tq1 = coeftq1_pTop_sQPEP_merge_s_sQPEP(coef);
coef_tq2 = coeftq2_pTop_sQPEP_merge_s_sQPEP(coef);
coef_tq3 = coeftq3_pTop_sQPEP_merge_s_sQPEP(coef);
coefs_tq=[coef_tq1;
    coef_tq2;
    coef_tq3];
G = G_pTop_sQPEP_merge_s_sQPEP(coef);
pinvG = pinv(G);

coef_Jacob_qt1 = coef_Jacob_qt1_pTop_sQPEP_merge_s_sQPEP(coefs_tq, pinvG, coef_J);
coef_Jacob_qt2 = coef_Jacob_qt2_pTop_sQPEP_merge_s_sQPEP(coefs_tq, pinvG, coef_J);
coef_Jacob_qt3 = coef_Jacob_qt3_pTop_sQPEP_merge_s_sQPEP(coefs_tq, pinvG, coef_J);
coef_Jacob_qt4 = coef_Jacob_qt4_pTop_sQPEP_merge_s_sQPEP(coefs_tq, pinvG, coef_J);
coef_Jacob_qt = [coef_Jacob_qt1;
    coef_Jacob_qt2;
    coef_Jacob_qt3;
    coef_Jacob_qt4];

QQ = Q_sym_pTop_sQPEP_merge_s_sQPEP(coefs_tq, pinvG, coef);
WW = W_pTop_sQPEP_merge_s_sQPEP(coefs_tq, pinvG, coef);
```

Then throw the extracted W and Q into the solver and extract all real number solutions.

```
[Q]=sQPEP_L2_Solver(WW,QQ, 123456);
cnt=0;
real_res_q=zeros(0,4);
for i = 1 : length(Q)
    tmp = Q{i};
    if (~isreal(tmp(1)))
        continue;
    end
    if (isnan(tmp(1)) || sum(Q{i})==0 )
        continue;
    end
    cnt = cnt + 1;
    real_s(cnt) = Q{i}.'*Q{i};
    real_res_q(cnt,:) = tmp/sqrt(real_s(cnt));
end
```

Exhausting the residuals of all solutions to obtain the solution with the smallest residual, remember the answer of that solution.

```
ret_q = real_res_q;
ret_s = real_s;
best_idx = 1;
best_cost = inf;
for i = 1 : length(ret_s)
    cur_s = ret_s(i);
    cur_q = ret_q(i,:);
    t1 = t1_pTop_sQPEP_merge_s_sQPEP(pinvG, coefs_tq, cur_q.'*sqrt(cur_s));
    t2 = t2_pTop_sQPEP_merge_s_sQPEP(pinvG, coefs_tq, cur_q.'*sqrt(cur_s));
    t3 = t3_pTop_sQPEP_merge_s_sQPEP(pinvG, coefs_tq, cur_q.'*sqrt(cur_s));
    cur_cost = J_func_pTop((cur_q).'*(sqrt(cur_s)), ([t1;t2;t3]), (r0), (b0), (nvr
    if (best_cost > cur_cost)
        best_cost = cur_cost;
        best_idx = i;
    end
end
best_q = ret_q(best_idx,:);
best_s = ret_s(best_idx);
best_t1 = t1_pTop_sQPEP_merge_s_sQPEP(pinvG, coefs_tq, best_q.'*sqrt(best_s));
best_t2 = t2_pTop_sQPEP_merge_s_sQPEP(pinvG, coefs_tq, best_q.'*sqrt(best_s));
best_t3 = t3_pTop_sQPEP_merge_s_sQPEP(pinvG, coefs_tq, best_q.'*sqrt(best_s));
best_t = [best_t1; best_t2; best_t3];
```

Table 1: A Data for the $\boldsymbol{AX = XB}$

A_{q_w}	A_{q_x}	A_{q_y}	A_{q_z}	A_{t_x}	A_{t_y}	A_{t_z}	B_{q_w}	B_{q_x}	B_{q_y}	B_{q_z}	B_{t_x}	B_{t_y}	B_{t_z}
0.154095	0.653855	0.207552	-0.711091	-1.064502	-4.186815	1.734748	0.148860	0.443449	-0.644170	0.605177	-1.509405	0.875874	-0.242790
0.916138	0.260064	-0.145164	-0.268300	-0.754779	1.278846	1.925689	0.915513	-0.047553	-0.325107	0.232118	0.788409	0.928736	-0.490790
0.586309	0.145360	-0.171361	-0.778298	-1.128529	-2.735935	-2.191314	0.585320	0.102649	-0.800370	0.079193	0.425864	-1.270043	-0.485219
0.276378	0.782089	0.551459	0.088570	-1.354937	1.004935	1.845635	0.278651	0.493465	0.207118	0.797464	0.446945	0.645825	-0.623677
0.007267	-0.751348	-0.565873	0.339428	4.099847	-0.406478	-2.458245	0.007923	-0.651843	0.185246	-0.735338	0.365490	-1.097061	1.930213
0.702386	-0.636611	-0.166976	-0.271107	0.978720	-3.418843	0.049831	0.701857	-0.069033	-0.262404	-0.658616	-0.801224	-0.325654	0.284676

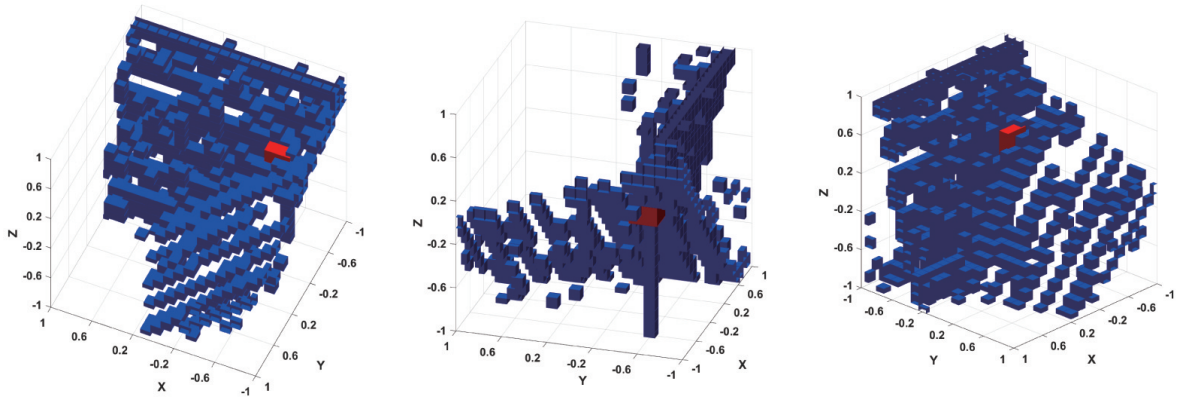

Figure 4: The convergence domains for the Gauss-Newton methods. The centers of the red blocks correspond to the ground truth translation values. The blue blocks denote regions where initializing the translation component at these centers, combined with the true values for rotation and scale, leads to iterations that converge to solutions closely approximating the true values, achieving angular errors within 1.0 degree. The scenarios depicted assume XYZ values ranging between -1 and 1.

5 Convergence Domains in Scaled Form Hand-Eye Problem

We incorporated experiments using the Gauss-Newton and Quasi-Newton methods for comparative analysis. These methods were applied using ground truth rotation and scale values as initial conditions, with the translation vector set to zero. The convergence precision was established at 10^{-12}, and the process was allowed a maximum of 1000 iterations. To further explore this issue, a supplementary section has been added that visualizes the convergence domains for a specific case. The data for this case are shown in Table 1, where the GT quaternion is $0.5968, -0.3832, -0.5606, -0.4272$, the GT translation is $-0.5727, -0.5587, 0.1784$, and the GT scale is 0.5. The convergence domains for this case are illustrated in Fig. 4. It is observed that even with the true values for rotation and scale provided, the convergence domains are quite small, indicating that the obtained solutions may not always represent the optimal solution to this problem. Therefore, employing global optimization methods is meaningful for the scaled form $AX = XB$.

6 Experimental Analysis

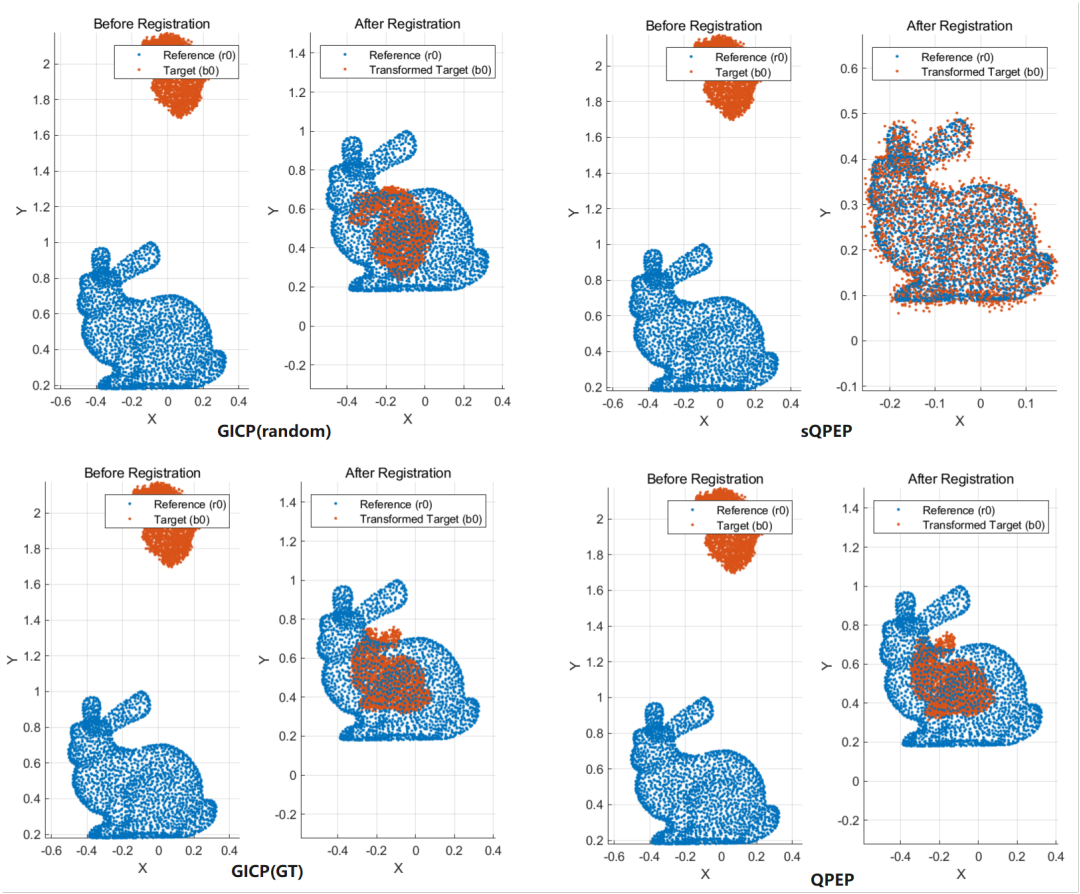

Figure 5: In the manuscript's Tab. II, the scale factor differs from 1; thus, the 6DOF method is evidently inapplicable. Additionally, it can be intuitively observed from the figure that algorithms based on numerical methods rely heavily on initial values. The absence of reliable initial values can lead to convergence to incorrect local optima. The increased variance can be attributed to these erroneous local optima, which rapidly amplify the variance. Therefore, the variance presented here should only be considered as a reference and is not suitable for characterizing the parameters of a normal distribution.

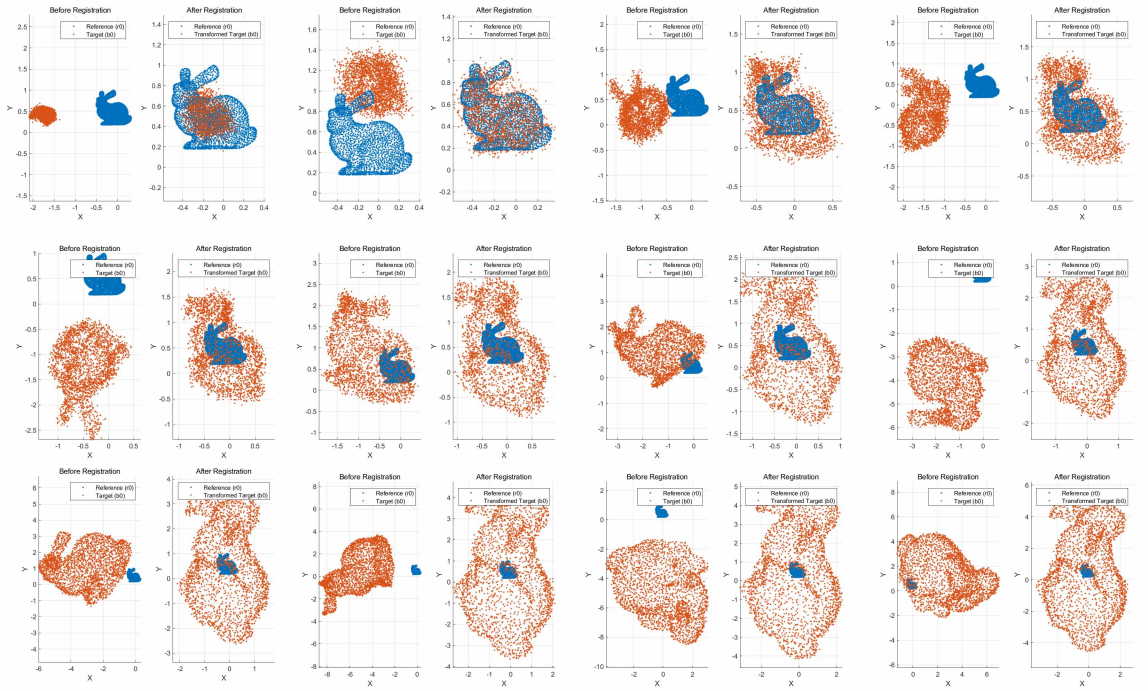

Figure 6: The figure is a vector graphics, which can be seen clearly when enlarged. As the scale increases, QPEP consistently identifies its ideal optimal position, illustrating the advantage of obtaining global optima. Nevertheless, deviations still occur. The graphical analysis reveals that as the scale enlarges, the angular errors of QPEP gradually converge. Furthermore, various optimization algorithms tend to converge towards the centroid position.

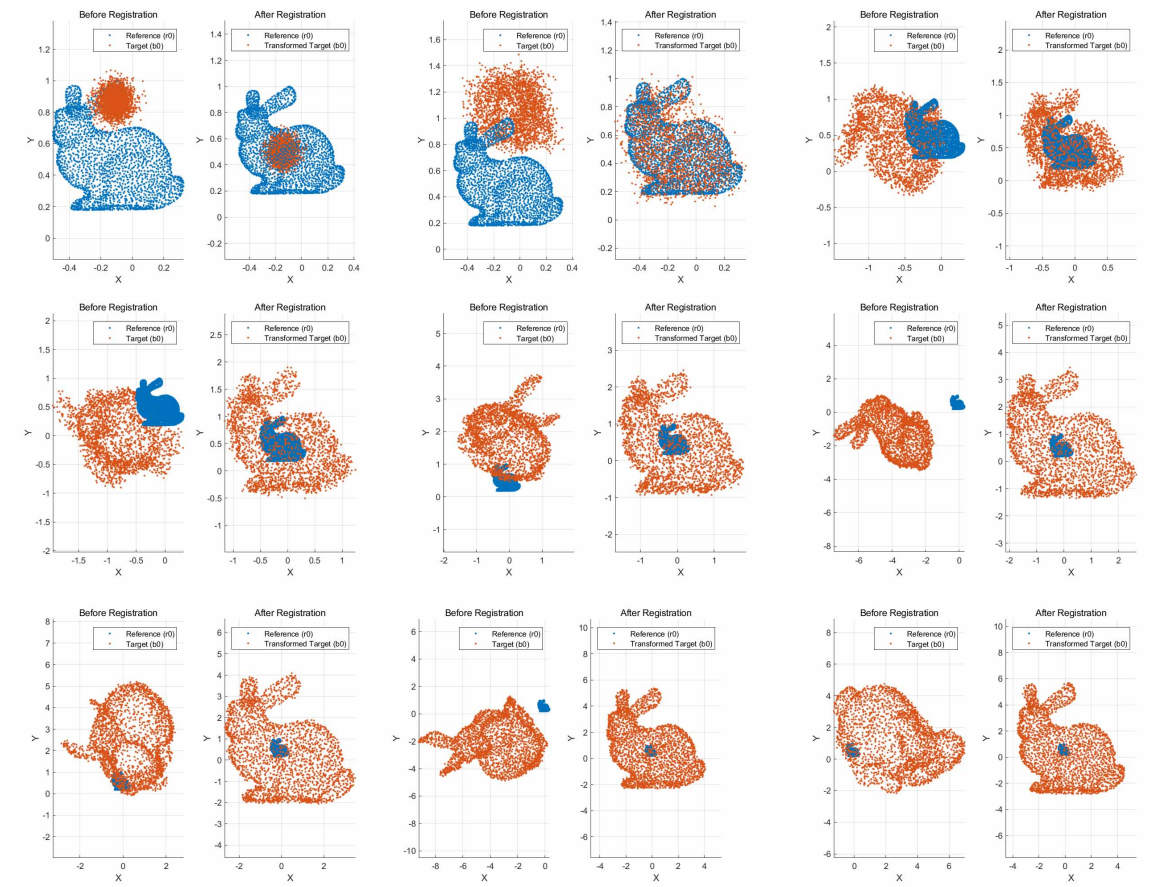

Figure 7: For the GICP and its variant, FAST-GICP, the selection of the GT as the initial value led them to converge to an alternative local optimum. The convergence behavior shows minor variations with increases in scale, which also explains the relatively stable angular errors observed. The specific differences in angular discrepancies are attributed to the distinct objective functions used by different algorithms, resulting in slight variations in the outcomes.

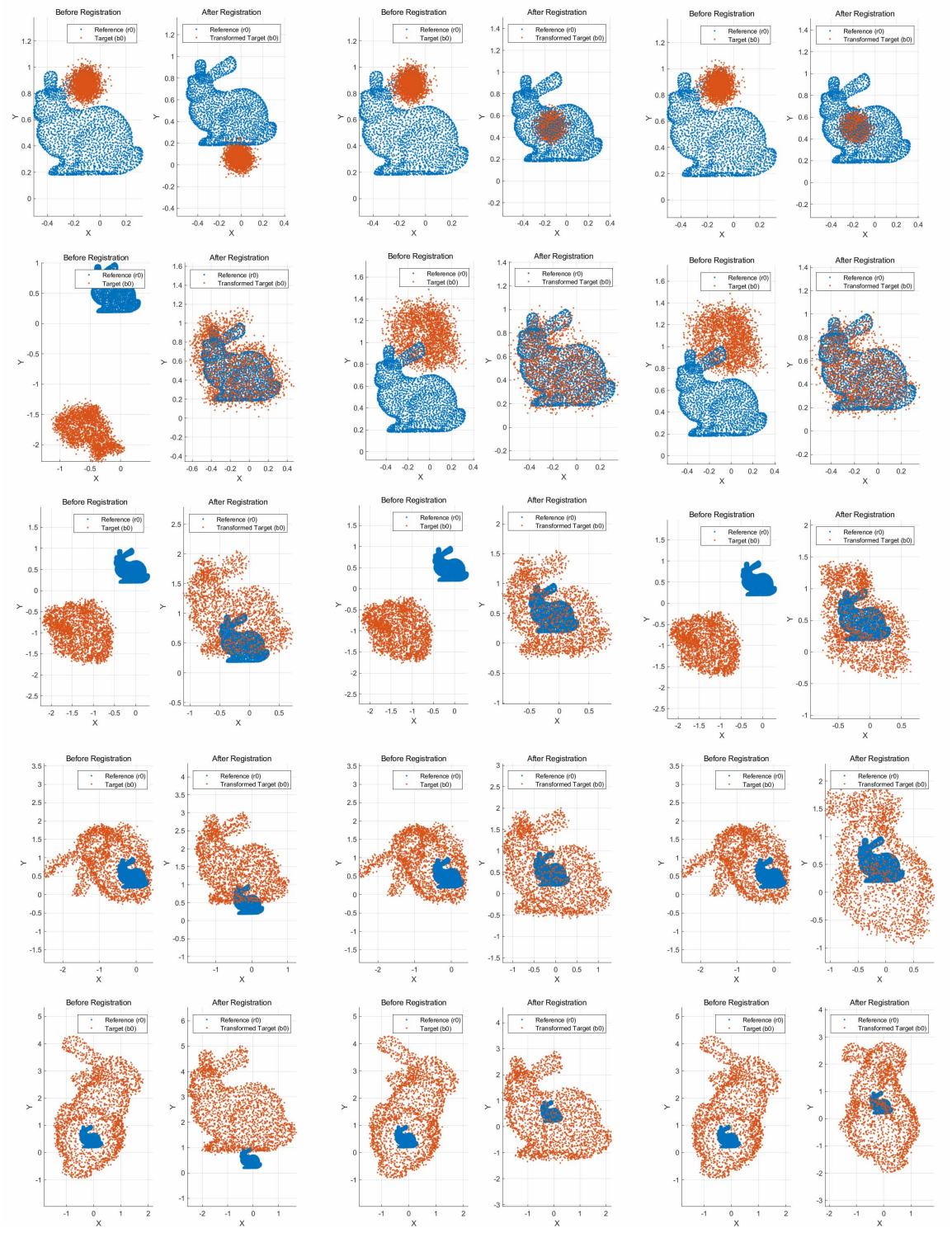

Figure 8: The first column contains data from the GT without scaling applied to the rabbit model. The second column presents data from the GICP, and the third column shows data from the QPEP. From top to bottom, the scales are set at 0.1, 1, 2, and 4, respectively. It is evident that the factor causing linear growth in translation errors is inherent to the problem itself. Since the coordinates prior to scaling are not aligned to the center of the point cloud, this misalignment leads to a linear increase in translation errors as the scale increases.

References

[1] J. Wu, Y. Zheng, Z. Gao, Y. Jiang, X. Hu, Y. Zhu, J. Jiao, and M. Liu, "Quadratic pose estimation problems: Globally optimal solutions, solvability/observability analysis, and uncertainty description," *IEEE Trans. Robot.*, vol. 38, no. 5, pp. 3314–3335, 2022.

[2] J. Sola, "Quaternion kinematics for the error-state kalman filter," *arXiv preprint arXiv:1711.02508*, 2017.

[3] V. Larsson, K. Astrom, and M. Oskarsson, "Efficient solvers for minimal problems by syzygy-based reduction," in *IEEE ICCV*, 2017, pp. 820–829.

[4] Z. Kukelova, M. Bujnak, and T. Pajdla, "Polynomial eigenvalue solutions to minimal problems in computer vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1381–1393, 2011.