Supplementary Material:

Quadratic Pose Estimation Problems: Globally Optimal Solutions, Solvability/Observability Analysis and Uncertainty Description

Jin Wu, Yu Zheng, Zhi Gao, Yi Jiang,

Xiangcheng Hu, Yilong Zhu, Jianhao Jiang and Ming Liu

This supplementary cookbook contains two parts: 1) theoretical proofs; 2) engineering applications and examples. The notations are inherited from the following paper:

WU, J., ET AL. (2021) Quadratic Pose Estimation Problems: Globally Optimal Solutions, Solvability/Observability Analysis and Uncertainty Description.

USAGE OF CODES:

The C++/MATLAB codes are available on `https://github.com/zarathustr/LibQPEP`. Please follow the instructions on the webpage to compile and install the library.

The MATLAB of version from R2007b to R2021a is capable of proper evaluation. The C++ codes are built using CMake toolkit under the C++11 programming standard. The MATLAB demo kit mainly consists of examples showing how QPEPs are constructed and solved. Three files `syms_hand_eye.m`, `syms_pnp.m`, `syms_pTop.m` contain symbolic generators for expression functions of hand-eye calibration, PnP and point-to-plane registration problems. Two test files `test_rel_att.m` and `test_stewart.m` illustrate how range-based relative pose problem and the forward kinematics problem of Stewart platform can be transformed into QPEPs. The final three files `test_cov_hand_eye.m`, `test_cov_pnp.m` and `test_cov_pTop.m` consist of globally optimal solutions and covariance estimation. The comparison with method of Nguyen et al. [1] is shown in `nguyen_covariance.m`. In covariance estimation codes of MATLAB, we use the `SeDuMi` [2] as a general optimizer. In the C++ code, the file `main.cpp` contains demos of pose and covariance estimation. The function `QPEP_grobner` solves the QPEPs via Gröbner-basis elimination. Using `QPEP_lm_single` function, the solved pose will be refined by the LM iteration. Finally, the function `csdp_cov` estimates the covariance information. For open-source verification of symbolic toolbox, we also implement the symbolic derivations of QPEP in the Octave language released by GNU. Please come to `https://github.com/zarathustr/OctQPEP` for copies of codes.

I. Theoretical Proofs

A. Derivation of QPEP and Jacobians

To replicate the results in Equation (15) $\sim$ (24) of original paper, please come to the MATLAB demo kit directory of the repository `https://github.com/zarathustr/LibQPEP/tree/main/MATLAB`. Here, for the example of hand-eye calibration, please open the file `syms_hand_eye.m` in the MATLAB editor. The following lines define the $\mathrm{SE}(3)$ matrices of $\boldsymbol{A}$ and $\boldsymbol{B}$ in the problem $\boldsymbol{AX} = \boldsymbol{XB}$ where $\boldsymbol{X}$ is the unknown hand-eye transformation.

```
syms A11 A12 A13 A14 real
syms A21 A22 A23 A24 real
syms A31 A32 A33 A34 real

A = [
    A11, A12, A13, A14;
    A21, A22, A23, A24;
    A31, A32, A33, A34;
      0,   0,   0,   1;
      ];

syms B11 B12 B13 B14 real
syms B21 B22 B23 B24 real
syms B31 B32 B33 B34 real

B = [
    B11, B12, B13, B14;
    B21, B22, B23, B24;
    B31, B32, B33, B34;
      0,   0,   0,   1;
      ];
```

The following codes construct the hand-eye calibration problem via the objective function `J_func_hand_eye` and then get the jacobian with respect to quaternion $\boldsymbol{q}$ and translation $\boldsymbol{t}$, together with Lagrange multiplier λ.

```
syms q0 q1 q2 q3 real
syms t1 t2 t3 lambda real
q = [q0; q1; q2; q3];
tt = [t1; t2; t3];
RR = q2R(q);
XX = [
    RR, tt;
    zeros(1, 3), 1];
J_pure = J_func_hand_eye(XX, A, B);
[coef_J_pure, mon_J_pure] = coeffs(J_pure, [q; tt]);
generateFuncFile(coef_J_pure, fullfile('func_files', 'coef_J_pure_hand_eye_func_new.m'), ...
    {A(1 : 3, :), B(1 : 3, :)});
generateFuncFile(mon_J_pure, fullfile('func_files', 'mon_J_pure_hand_eye_func_new.m'), ...
    {q, tt});
J = J_pure - 0.5 * lambda * (q.' * q - 1);
x = [q; tt; lambda];
Jacob = jacobian(J, x);
Jacob = expand(Jacob.');
```

where `Jacob` is exactly the system (15) in the original paper. After that, the following codes extract mandatory coefficients to be used for symbolic manipulation and store them as optimized function files for fast numerical computation:

```
[coeft1, mont1] = coeffs(Jacob(5), tt);
```

```
g1 = coeft1(1 : 3).';
[coeftq1, montq1] = coeffs(expand(Jacob(5) - g1.' * tt ), q);
generateFuncFile(coeftq1, fullfile('func_files', 'coeftq1_hand_eye_func_new.m'), {coef_J});
[coeft2, mont2] = coeffs(Jacob(6), tt);
g2 = coeft2(1 : 3).';
[coeftq2, montq2] = coeffs(expand(Jacob(6) - g2.' * tt ), q);
generateFuncFile(coeftq2, fullfile('func_files', 'coeftq2_hand_eye_func_new.m'), {coef_J});
[coeft3, mont3] = coeffs(Jacob(7), tt);
g3 = coeft3(1 : 3).';
[coeftq3, montq3] = coeffs(expand(Jacob(7) - g3.' * tt ), q);
generateFuncFile(coeftq3, fullfile('func_files', 'coeftq3_hand_eye_func_new.m'), {coef_J});
G = - [g1.'; g2.'; g3.'];
generateFuncFile(G, fullfile('func_files', 'G_hand_eye_func_new.m'), {coef_J});

pinvG = sym('pinvG', [3, 3]);
showSyms(symvar(pinvG));
coefs_tq = sym('coefs_tq', [3, 11]);
showSyms(symvar(coefs_tq));
ts = pinvG * coefs_tq * montq1.';
t1 = ts(1);
t2 = ts(2);
t3 = ts(3);
generateFuncFile(t1, fullfile('func_files', 't1_hand_eye_func_new.m'), {pinvG, coefs_tq, ...
    q});
generateFuncFile(t2, fullfile('func_files', 't2_hand_eye_func_new.m'), {pinvG, coefs_tq, ...
    q});
generateFuncFile(t3, fullfile('func_files', 't3_hand_eye_func_new.m'), {pinvG, coefs_tq, ...
    q});

[coef_Jacob1_qt, mon_Jacob1_qt] = coeffs(Jacob(1) + lambda * q0, [q; tt]);
generateFuncFile(coef_Jacob1_qt, fullfile('func_files', ...
    'coef_Jacob1_qt_hand_eye_func_new.m'), {coef_J});
[coef_Jacob2_qt, mon_Jacob2_qt] = coeffs(Jacob(2) + lambda * q1, [q; tt]);
generateFuncFile(coef_Jacob2_qt, fullfile('func_files', ...
    'coef_Jacob2_qt_hand_eye_func_new.m'), {coef_J});
[coef_Jacob3_qt, mon_Jacob3_qt] = coeffs(Jacob(3) + lambda * q2, [q; tt]);
generateFuncFile(coef_Jacob3_qt, fullfile('func_files', ...
    'coef_Jacob3_qt_hand_eye_func_new.m'), {coef_J});
[coef_Jacob4_qt, mon_Jacob4_qt] = coeffs(Jacob(4) + lambda * q3, [q; tt]);
generateFuncFile(coef_Jacob4_qt, fullfile('func_files', ...
    'coef_Jacob4_qt_hand_eye_func_new.m'), {coef_J});

coef_Jacob1_qt_syms = sym('coef_Jacob1_qt_syms', [1, 36]);
coef_Jacob2_qt_syms = sym('coef_Jacob2_qt_syms', [1, 36]);
coef_Jacob3_qt_syms = sym('coef_Jacob3_qt_syms', [1, 36]);
coef_Jacob4_qt_syms = sym('coef_Jacob4_qt_syms', [1, 36]);
showSyms(symvar(coef_Jacob1_qt_syms));
showSyms(symvar(coef_Jacob2_qt_syms));
showSyms(symvar(coef_Jacob3_qt_syms));
showSyms(symvar(coef_Jacob4_qt_syms));
coef_Jacob_qt_syms = [
    coef_Jacob1_qt_syms;
    coef_Jacob2_qt_syms;
    coef_Jacob3_qt_syms;
    coef_Jacob4_qt_syms;
    ];
Jacob_ = coef_Jacob_qt_syms * mon_Jacob1_qt.';

eqs = expand(eval([Jacob_; Jacob(8)]));

f0 = eqs(1);
f1 = eqs(2);
f2 = eqs(3);
f3 = eqs(4);
[coef_f0_q, mon_f0_q] = coeffs(f0, q);
```

```
[coef_f1_q, mon_f1_q] = coeffs(f1, q);
[coef_f2_q, mon_f2_q] = coeffs(f2, q);
[coef_f3_q, mon_f3_q] = coeffs(f3, q);
coef_f0_q_sym = sym('coef_f0_q_sym', [1, 24]);
coef_f1_q_sym = sym('coef_f1_q_sym', [1, 24]);
coef_f2_q_sym = sym('coef_f2_q_sym', [1, 24]);
coef_f3_q_sym = sym('coef_f3_q_sym', [1, 24]);
showSyms(symvar(coef_f0_q_sym));
showSyms(symvar(coef_f1_q_sym));
showSyms(symvar(coef_f2_q_sym));
showSyms(symvar(coef_f3_q_sym));

coef_f_q_sym = [
    coef_f0_q;
    coef_f1_q;
    coef_f2_q;
    coef_f3_q;
    ];
generateFuncFile(coef_f_q_sym, fullfile('func_files', ...
    'coef_f_q_sym_hand_eye_func_new.m'), {pinvG, coefs_tq, coef_Jacob_qt_syms});

f0 = coef_f0_q_sym * mon_f0_q.';
f1 = coef_f1_q_sym * mon_f1_q.';
f2 = coef_f2_q_sym * mon_f2_q.';
f3 = coef_f3_q_sym * mon_f3_q.';
eq = expand([
     f0 * q1 - f1 * q0;
     f0 * q2 - f2 * q0;
     f0 * q3 - f3 * q0;
     q0 * q0 + q1 * q1 + q2 * q2 + q3 * q3 - 1;
     ]);
generateFuncFile(eq, fullfile('func_files', 'eq_hand_eye_func_new.m'), {coef_f0_q_sym, ...
    coef_f1_q_sym, coef_f2_q_sym, coef_f3_q_sym, q});
```

The following codes applies the quaternion nature to simplify the system:

```
eq_ = eq;
eq_ = expand(eval(subs(eq_, q0^2, (1 - q1^2 - q2^2 - q3^2))));
[coefeq1, moneq1] = coeffs(eq_(1), q);
mon2th1 = [ moneq1(10), moneq1(14), moneq1(16), moneq1(23), moneq1(27), moneq1(29), ...
    moneq1(33), moneq1(35), moneq1(37)];
mon4th1 = [ moneq1(1 : 9), moneq1(11 : 13), moneq1(15), moneq1(17 : 22), moneq1(24 : ...
    26), moneq1(28), moneq1(30 : 32), moneq1(34), moneq1(36), moneq1(38)];

[coefeq2, moneq2] = coeffs(eq_(2), q);
mon2th2 = [ moneq2(10), moneq2(14), moneq2(16), moneq2(23), moneq2(27), moneq2(29), ...
    moneq2(33), moneq2(35), moneq2(37)];
mon4th2 = [ moneq2(1 : 9), moneq2(11 : 13), moneq2(15), moneq2(17 : 22), moneq2(24 : ...
    26), moneq2(28), moneq2(30 : 32), moneq2(34), moneq2(36), moneq2(38)];

[coefeq3, moneq3] = coeffs(eq_(3), q);
mon2th3 = [ moneq3(10), moneq3(14), moneq3(16), moneq3(23), moneq3(27), moneq3(29), ...
    moneq3(33), moneq3(35), moneq3(37)];
mon4th3 = [ moneq3(1 : 9), moneq3(11 : 13), moneq3(15), moneq3(17 : 22), moneq3(24 : ...
    26), moneq3(28), moneq3(30 : 32), moneq3(34), moneq3(36), moneq3(38)];

coef2th1 = [ coefeq1(10), coefeq1(14), coefeq1(16), coefeq1(23), coefeq1(27), ...
    coefeq1(29), coefeq1(33), coefeq1(35), coefeq1(37)];
coef4th1 = [ coefeq1(1 : 9), coefeq1(11 : 13), coefeq1(15), coefeq1(17 : 22), coefeq1(24 ...
    : 26), coefeq1(28), coefeq1(30 : 32), coefeq1(34), coefeq1(36), coefeq1(38)];

coef2th2 = [ coefeq2(10), coefeq2(14), coefeq2(16), coefeq2(23), coefeq2(27), ...
    coefeq2(29), coefeq2(33), coefeq2(35), coefeq2(37)];
coef4th2 = [ coefeq2(1 : 9), coefeq2(11 : 13), coefeq2(15), coefeq2(17 : 22), coefeq2(24 ...
```

```
    : 26), coefeq2(28), coefeq2(30 : 32), coefeq2(34), coefeq2(36), coefeq2(38)];

coef2th3 = [ coefeq3(10), coefeq3(14), coefeq3(16), coefeq3(23), coefeq3(27), ...
    coefeq3(29), coefeq3(33), coefeq3(35), coefeq3(37)];
coef4th3 = [ coefeq3(1 : 9), coefeq3(11 : 13), coefeq3(15), coefeq3(17 : 22), coefeq3(24 ...
    : 26), coefeq3(28), coefeq3(30 : 32), coefeq3(34), coefeq3(36), coefeq3(38)];

y = mon2th1.';
generateFuncFile(y, fullfile('func_files', 'y_func_hand_eye_new.m'), {q});
v = mon4th1(1 : end - 1).';
generateFuncFile(v, fullfile('func_files', 'v_func_hand_eye_new.m'), {q});
```

Finally, we obtain the $\boldsymbol{D}, \boldsymbol{G}, \boldsymbol{c}$ terms in (23) and $\boldsymbol{W}, \boldsymbol{Q}$ terms in (17) by

```
D = [
    coef4th1(1 : end - 1);
    coef4th2(1 : end - 1);
    coef4th3(1 : end - 1);
    ];

G = [
    coef2th1;
    coef2th2;
    coef2th3;
    ];

c = [
    coef4th1(end);
    coef4th2(end);
    coef4th3(end);
    ];

generateFuncFile(D, fullfile('func_files', 'D_func_hand_eye_new.m'), {coef_f0_q_sym, ...
    coef_f1_q_sym, coef_f2_q_sym, coef_f3_q_sym});
generateFuncFile(G, fullfile('func_files', 'G_func_hand_eye_new.m'), {coef_f0_q_sym, ...
    coef_f1_q_sym, coef_f2_q_sym, coef_f3_q_sym});
generateFuncFile(c, fullfile('func_files', 'c_func_hand_eye_new.m'), {coef_f0_q_sym, ...
    coef_f1_q_sym, coef_f2_q_sym, coef_f3_q_sym});

HH = jacobian(eq, q);
generateFuncFile(HH, fullfile('func_files', 'Jacob_hand_eye_func_new.m'), ...
    {coef_f0_q_sym, coef_f1_q_sym, coef_f2_q_sym, coef_f3_q_sym, q});
gradient = expand(HH.' * eq);

[coef1, mon1] = coeffs(eqs(1), x);
[coef2, mon2] = coeffs(eqs(2), x);
[coef3, mon3] = coeffs(eqs(3), x);
[coef4, mon4] = coeffs(eqs(4), x);

Q_sym = [
    coef1(11), coef1(18), coef1(22), coef1(24);
    coef2(11), coef2(18), coef2(22), coef2(24);
    coef3(11), coef3(18), coef3(22), coef3(24);
    coef4(11), coef4(18), coef4(22), coef4(24);
    ];
generateFuncFile(Q_sym, fullfile('func_files', 'Q_hand_eye_func_new.m'), {pinvG, ...
    coefs_tq, coef_Jacob_qt_syms});

res = expand(eqs(1 : 4) - Q_sym * q);
H = jacobian(res, q);
h1 = H(:, 1);
h2 = H(:, 2);
h3 = H(:, 3);
h4 = H(:, 4);
```

```
P1 = jacobian(h1, q);
P2 = jacobian(h2, q);
P3 = jacobian(h3, q);
P4 = jacobian(h4, q);

for i = 1 : 4
    for j = 1 : 4
        str = sprintf('W%d%d = [jacobian(P%d(1, :), q%d).''; jacobian(P%d(2, :), ...
            q%d).''; jacobian(P%d(3, :), q%d).''; jacobian(P%d(4, :), q%d).''];', ...
                      i, j, i, j - 1, i, j - 1, i, j - 1, i, j - 1);
        eval(str);
    end
end


W1 = [W11, W12, W13, W14];
W2 = [W21, W22, W23, W24];
W3 = [W31, W32, W33, W34];
W4 = [W41, W42, W43, W44];
v = kron(q, q);
u = kron(v, q);
W = - [W1, W2, W3, W4] / 6;
generateFuncFile(W, fullfile('func_files', 'W_hand_eye_func_new.m'), {pinvG, coefs_tq, ...
    coef_Jacob_qt_syms});
```

For other algorithms like PnP and Point-to-Plane registration, please turn to `syms_pnp.m` and `syms_pTop.m`, respectively.

B. Relationship between $\boldsymbol{\Sigma_q}$ and $\boldsymbol{\Sigma_R}$

Since $\boldsymbol{\Sigma_q} \in \mathbb{S}^4$ and $\boldsymbol{\Sigma_R} \in \mathbb{S}^3$, the conversion from $\boldsymbol{\Sigma_q}$ to $\boldsymbol{\Sigma_R}$ loses information. Thus the restoration from $\boldsymbol{\Sigma_R}$ to $\boldsymbol{\Sigma_q}$ is challenging. Given a rotation matrix $\boldsymbol{R} \in \mathrm{SO}(3)$, it is able for us to write it into affined-quaternion form in columns $\boldsymbol{R} = (\boldsymbol{P}_1\boldsymbol{q}, \boldsymbol{P}_2\boldsymbol{q}, \boldsymbol{P}_3\boldsymbol{q})$ [3], where

$$\begin{aligned}
\boldsymbol{P}_1 &= \begin{pmatrix} q_0 & q_1 & -q_2 & -q_3 \\ -q_3 & q_2 & q_1 & -q_0 \\ q_2 & q_3 & q_0 & q_1 \end{pmatrix} \\
\boldsymbol{P}_2 &= \begin{pmatrix} q_3 & q_2 & q_1 & q_0 \\ q_0 & -q_1 & q_2 & -q_3 \\ -q_1 & -q_0 & q_3 & q_2 \end{pmatrix} \\
\boldsymbol{P}_3 &= \begin{pmatrix} -q_2 & q_3 & -q_0 & q_1 \\ q_1 & q_0 & q_3 & q_2 \\ q_0 & -q_1 & -q_2 & q_3 \end{pmatrix}.
\end{aligned} \tag{1}$$

A small difference of $\boldsymbol{R}$ reads

$$\Delta\boldsymbol{R} = \left(\Delta\boldsymbol{P}_1\boldsymbol{q} + \boldsymbol{P}_1\Delta\boldsymbol{q},\ \Delta\boldsymbol{P}_2\boldsymbol{q} + \boldsymbol{P}_2\Delta\boldsymbol{q},\ \Delta\boldsymbol{P}_3\boldsymbol{q} + \boldsymbol{P}_3\Delta\boldsymbol{q}\right). \tag{2}$$

An interesting fact is that $\Delta\boldsymbol{P}_i\boldsymbol{q} = \boldsymbol{P}_i\Delta\boldsymbol{q}$, for $i = 1, 2, 3$. Thus, one obtains $\Delta\boldsymbol{R} = 2\left(\boldsymbol{P}_1\Delta\boldsymbol{q}, \boldsymbol{P}_2\Delta\boldsymbol{q}, \boldsymbol{P}_3\Delta\boldsymbol{q}\right)$. The covariance of $\boldsymbol{R}$ is then evaluated via

$$\boldsymbol{\Sigma_R} = \left\langle \Delta\boldsymbol{R}\Delta\boldsymbol{R}^\top \right\rangle = 4\sum\nolimits_{i=1}^{3} \boldsymbol{P}_i\boldsymbol{\Sigma_q}\boldsymbol{P}_i^\top. \tag{3}$$

The meaning of $\boldsymbol{\Sigma_R}$ actually can be approximated by the covariance of the Lie algebra $\boldsymbol{\xi}$ of $\boldsymbol{R}$ since a rotation with small angle approximation reads $\boldsymbol{R} \approx \boldsymbol{I} + \boldsymbol{\xi}_\times$, which leads to $\Delta\boldsymbol{R} \approx \Delta\boldsymbol{\xi}_\times$ and $\boldsymbol{\Sigma_R} \approx \langle\Delta\boldsymbol{\xi}_\times\Delta\boldsymbol{\xi}_\times\rangle$.

To restore $\boldsymbol{\Sigma_q}$ from a covariance $\boldsymbol{\Sigma_R}$ of a rotation matrix, we can construct the following optimization

$$\begin{aligned}
\underset{\boldsymbol{\Sigma_q}\in\mathbb{S}^4}{\arg\min}\, \mathrm{tr}\left[\left(4\sum \boldsymbol{P}_i\boldsymbol{\Sigma_q}\boldsymbol{P}_i^\top - \boldsymbol{\Sigma_R}\right)^\top\left(4\sum \boldsymbol{P}_i\boldsymbol{\Sigma_q}\boldsymbol{P}_i^\top - \boldsymbol{\Sigma_R}\right)\right] \\
\text{s.t.}\quad \boldsymbol{\Sigma_q} \succ \boldsymbol{0} \\
\boldsymbol{q}^\top\boldsymbol{\Sigma_q}\boldsymbol{q} \le \tilde{\alpha}.
\end{aligned} \tag{4}$$

As an example, we set a true rotation matrix as

$$\boldsymbol{R}_{\mathrm{true}} = \begin{pmatrix} 0.090760 & -0.784752 & -0.613129 \\ -0.858405 & -0.373774 & 0.351332 \\ -0.504880 & 0.494426 & -0.707558 \end{pmatrix}$$

whose quaternion is
$\boldsymbol{q}_{\mathrm{true}} = (-0.04855, 0.7369, -0.5575, -0.3793)^\top$. We conduct a random perturbation of quaternion via

$$\begin{aligned}
\tilde{\boldsymbol{q}} &= \boldsymbol{q}_{\mathrm{true}} + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}\left(\boldsymbol{0}, 1\times10^{-3}\boldsymbol{I}\right) \\
\boldsymbol{q} &= \tilde{\boldsymbol{q}}/\sqrt{\tilde{\boldsymbol{q}}^\top\tilde{\boldsymbol{q}}}.
\end{aligned} \tag{5}$$

Then we convert it to a rotation matrix $\boldsymbol{R}$. We perform such perturbations for 10000 times and gather statistical covariance of $\boldsymbol{R}$ as

$$\boldsymbol{\Sigma_R} = \begin{pmatrix} 7.9832\times10^{-6} & 3.9957\times10^{-8} & 8.1849\times10^{-9} \\ 3.9957\times10^{-8} & 7.9093\times10^{-6} & 2.4435\times10^{-8} \\ 8.1849\times10^{-9} & 2.4435\times10^{-8} & 7.8990\times10^{-6} \end{pmatrix}.$$

Using (4), we estimate the most appropriate $\boldsymbol{\Sigma_q}$ as

$$\boldsymbol{\Sigma_q} = \begin{pmatrix} 9.85240 & 0.37501 & -0.29118 & -0.10469 \\ 0.37501 & 4.56178 & 4.10021 & 2.78892 \\ -0.29118 & 4.10021 & 6.79495 & -1.98310 \\ -0.10469 & 2.78892 & -1.98310 & 8.34663 \end{pmatrix} \times 10^{-7}.$$

The statistical quaternion covariance is

$$\boldsymbol{\Sigma_q} = \begin{pmatrix} 9.91364 & 0.37726 & -0.29298 & -0.10531 \\ 0.37726 & 4.59001 & 4.12566 & 2.80619 \\ -0.29298 & 4.12566 & 6.83718 & -1.99540 \\ -0.10531 & 2.80619 & -1.99540 & 8.39851 \end{pmatrix} \times 10^{-7}.$$

Visualizing them generates Fig. 1, which shows that the restoration is almost lossless.

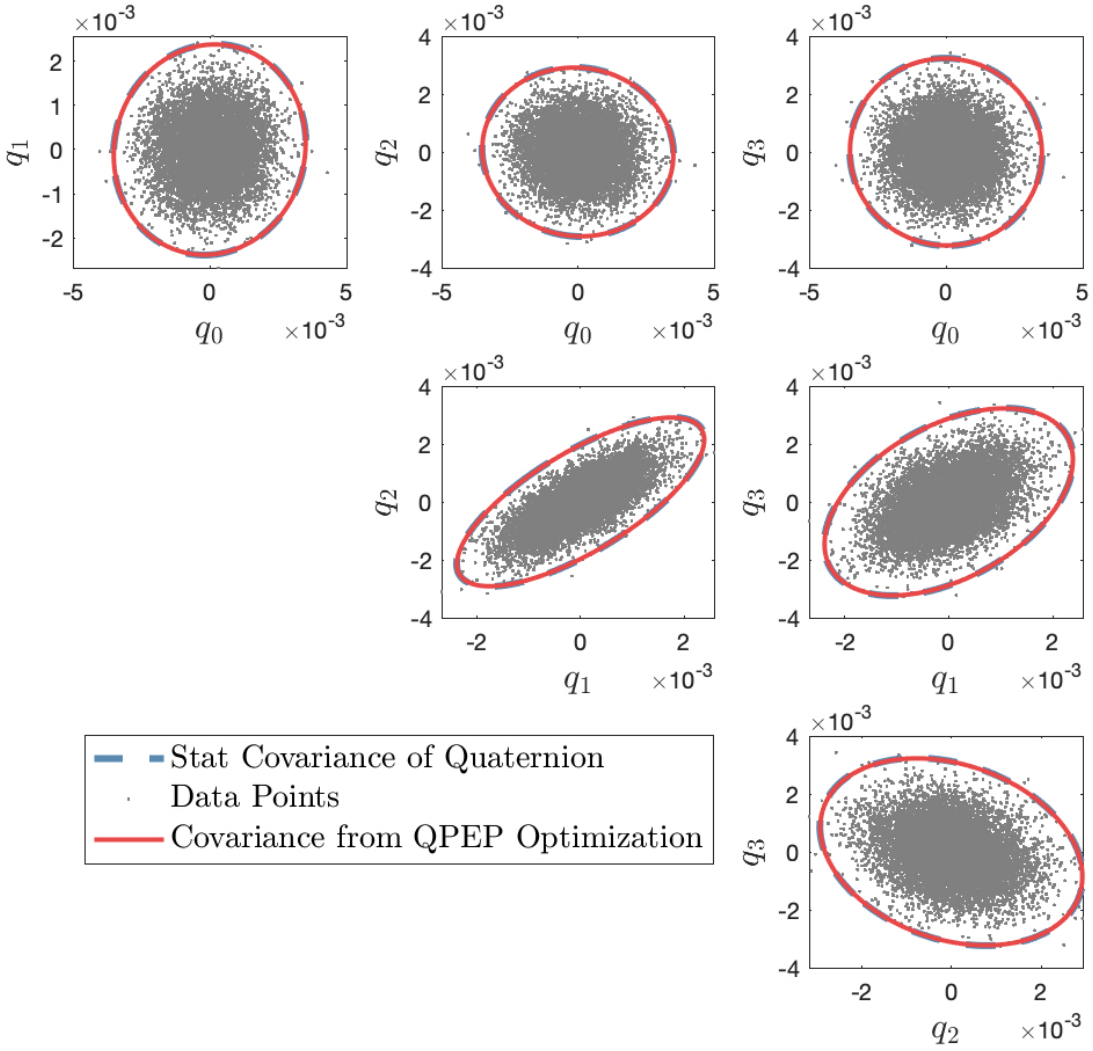


Fig. 1. The restored quaternion covariance from $\boldsymbol{\Sigma_R}$ with statistically averaged one.

C. Relationship between $\boldsymbol{\Sigma_q}$, $\boldsymbol{\Sigma_R}$ and $\boldsymbol{\Sigma_\xi}$

The Lie algebra $\boldsymbol{\xi}$ associated with a rotation matrix $\boldsymbol{R}$ has the property of $\boldsymbol{\xi}_\times = \log(\boldsymbol{R})$. If we write the Lie algebra as the $\boldsymbol{\xi} = \theta\boldsymbol{\theta}$, where θ is the norm of $\boldsymbol{\xi}$ and $\boldsymbol{\theta}$ is the normalized vector of $\boldsymbol{\xi}$, we then need to determine the covariance of $\boldsymbol{\xi}$ for a given covariance of $\boldsymbol{R}$. In Section I-B, the lossless conversion from $\boldsymbol{\Sigma_R}$ to $\boldsymbol{\Sigma_q}$ has been formulated. Therefore, the conversion from $\boldsymbol{\Sigma_R}$ to $\boldsymbol{\Sigma_\xi}$ can be bridged using $\boldsymbol{\Sigma_q}$. There is an important property of a quaternion $\boldsymbol{q}$ corresponding to a rotation $\boldsymbol{R}$, such that it can be represented as

$$\boldsymbol{q} = \left(q_0, \tilde{\boldsymbol{q}}^\top\right)^\top \tag{6}$$

where $q_0 = \cos(\theta/2)$ is the scalar part and $\tilde{\boldsymbol{q}} = \sin(\theta/2)\boldsymbol{\theta}$ is the vector part. To derive the uncertainty of $\boldsymbol{\xi}$, we first study $\Delta\boldsymbol{q}$:

$$\begin{aligned}\Delta\boldsymbol{q} &= \left(-\frac{1}{2}\sin\frac{\theta}{2}\Delta\theta, \frac{1}{2}\cos\frac{\theta}{2}\Delta\theta\boldsymbol{\theta}^\top + \sin\frac{\theta}{2}\Delta\boldsymbol{\theta}^\top\right)^\top \\ &= \Delta(q_0, \tilde{\boldsymbol{q}}^\top)^\top \end{aligned} \tag{7}$$

This gives the fact that

$$\boldsymbol{\Sigma}_{q_0} = \frac{1}{4}\sin^2\frac{\theta}{2}\boldsymbol{\Sigma}_\theta \tag{8}$$

$$\boldsymbol{\Sigma}_{q_0,\tilde{\boldsymbol{q}}} = -\frac{1}{4}\sin\frac{\theta}{2}\cos\frac{\theta}{2}\boldsymbol{\Sigma}_\theta\boldsymbol{\theta}^\top - \frac{1}{2}\sin^2\frac{\theta}{2}\boldsymbol{\Sigma}_{\theta,\boldsymbol{\theta}} \tag{9}$$

$$\begin{aligned}\boldsymbol{\Sigma}_{\tilde{\boldsymbol{q}}} =& \frac{1}{4}\cos^2\frac{\theta}{2}\boldsymbol{\theta}\boldsymbol{\Sigma}_\theta\boldsymbol{\theta}^\top + \sin^2\frac{\theta}{2}\boldsymbol{\Sigma}_{\boldsymbol{\theta}} \\ &+\frac{1}{2}\sin\frac{\theta}{2}\cos\frac{\theta}{2}\boldsymbol{\theta}\boldsymbol{\Sigma}_{\theta,\boldsymbol{\theta}} + \frac{1}{2}\sin\frac{\theta}{2}\cos\frac{\theta}{2}\boldsymbol{\Sigma}_{\boldsymbol{\theta},\theta}\boldsymbol{\theta}^\top \end{aligned} \tag{10}$$

which give the results of $\boldsymbol{\Sigma}_\theta$ and $\boldsymbol{\Sigma}_{\theta,\boldsymbol{\theta}}$ gradually. Then $\boldsymbol{\Sigma_\xi}$ can be give by

$$\Delta\boldsymbol{\xi} = \Delta\theta\boldsymbol{\theta} + \theta\Delta\boldsymbol{\theta} \tag{11}$$

which implies

$$\boldsymbol{\Sigma_\xi} = \boldsymbol{\theta}\boldsymbol{\Sigma}_\theta\boldsymbol{\theta}^\top + \theta^2\boldsymbol{\Sigma}_{\boldsymbol{\theta}} + \theta\boldsymbol{\theta}\boldsymbol{\Sigma}_{\theta,\boldsymbol{\theta}} + \theta\boldsymbol{\Sigma}_{\boldsymbol{\theta},\theta}\boldsymbol{\theta}^\top \tag{12}$$

whose required terms are given in (8) to (10). A simple test is performed to show the accuracy of covariance conversion from $\boldsymbol{\Sigma_q}$ to $\boldsymbol{\Sigma_\xi}$:

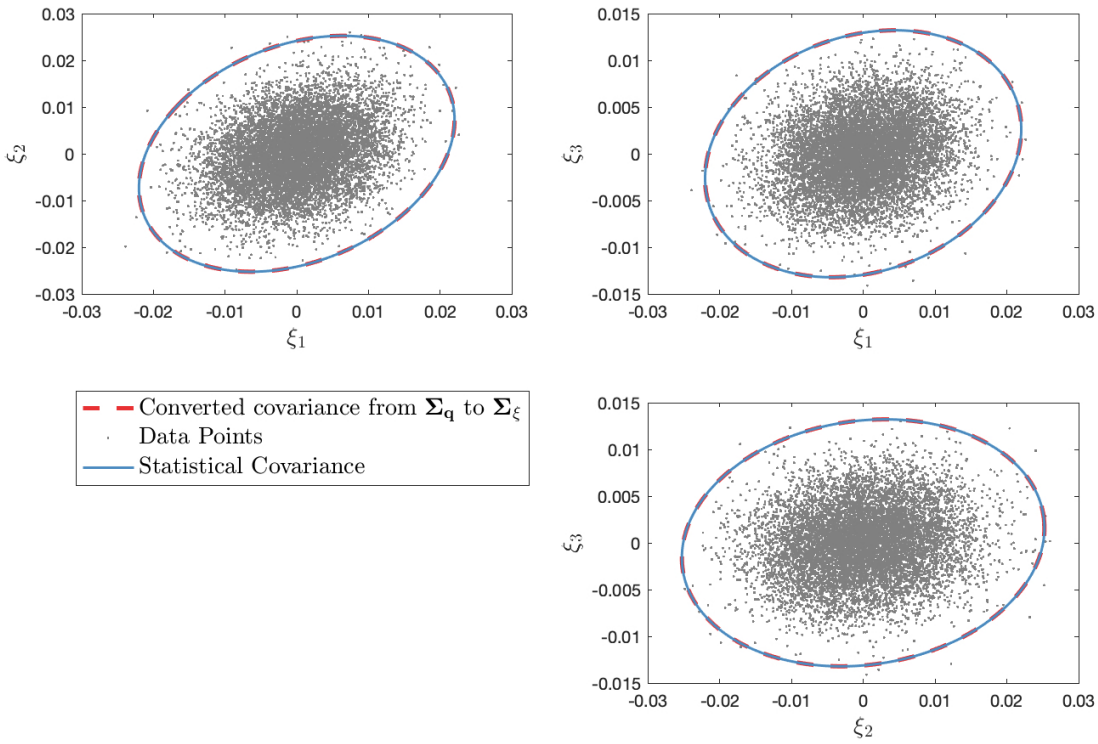


Fig. 2. The restored Lie algebra covariance from $\mathbf{\Sigma}_{\boldsymbol{q}}$ with statistically averaged one.

D. Compounding Two Poses Using Quaternions

Suppose that we have two poses to be compounded: $\boldsymbol{T}_1 = \mathrm{SE}_3(\boldsymbol{R}_1, \boldsymbol{t}_1) \in \mathrm{SE}(3)$, $\boldsymbol{T}_2 = \mathrm{SE}_3(\boldsymbol{R}_2, \boldsymbol{t}_2) \in \mathrm{SE}(3)$. The result pose is

$$\boldsymbol{T}_3 = \mathrm{SE}_3(\boldsymbol{R}_3, \boldsymbol{t}_3) = \begin{pmatrix} \boldsymbol{R}_1\boldsymbol{R}_2 & \boldsymbol{R}_1\boldsymbol{t}_2 + \boldsymbol{t}_1 \\ \boldsymbol{0} & 1 \end{pmatrix} \in \mathrm{SE}(3) \tag{13}$$

which implies

$$\Delta\boldsymbol{T}_3 = \begin{pmatrix} \Delta\boldsymbol{R}_1\boldsymbol{R}_2 + \boldsymbol{R}_1\Delta\boldsymbol{R}_2 & \Delta\boldsymbol{R}_1\boldsymbol{t}_2 + \boldsymbol{R}_1\Delta\boldsymbol{t}_2 + \Delta\boldsymbol{t}_1 \\ \boldsymbol{0} & 0 \end{pmatrix}. \tag{14}$$

Therefore we have

$$\begin{aligned} \boldsymbol{\Sigma}_{\boldsymbol{R}_3} &= \boldsymbol{\mathcal{C}}_{\boldsymbol{R}}\boldsymbol{\Sigma}_{\boldsymbol{q}_1}\boldsymbol{\mathcal{C}}_{\boldsymbol{R}}^\top + \boldsymbol{R}_1\boldsymbol{\Sigma}_{\boldsymbol{R}_2}\boldsymbol{R}_1^\top + \boldsymbol{\mathcal{C}}_{\boldsymbol{R}}\boldsymbol{\Sigma}_{\boldsymbol{q}_1,\boldsymbol{R}_2}\boldsymbol{R}_1^\top + \boldsymbol{R}_1\boldsymbol{\Sigma}_{\boldsymbol{R}_2,\boldsymbol{q}_1}\boldsymbol{\mathcal{C}}_{\boldsymbol{R}}^\top \\ \boldsymbol{\Sigma}_{\boldsymbol{t}_3} &= \boldsymbol{\mathcal{C}}_{\boldsymbol{t}}\boldsymbol{\Sigma}_{\boldsymbol{q}_1}\boldsymbol{\mathcal{C}}_{\boldsymbol{t}}^\top + \boldsymbol{R}_1\boldsymbol{\Sigma}_{\boldsymbol{t}_2}\boldsymbol{R}_1^\top + \boldsymbol{\Sigma}_{\boldsymbol{t}_1} + \\ &\quad \boldsymbol{\mathcal{C}}_{\boldsymbol{t}}\boldsymbol{\Sigma}_{\boldsymbol{q}_1,\boldsymbol{t}_2}\boldsymbol{R}_1^\top + \boldsymbol{\mathcal{C}}_{\boldsymbol{t}}\boldsymbol{\Sigma}_{\boldsymbol{q}_1,\boldsymbol{t}_1} + \boldsymbol{R}_1\boldsymbol{\Sigma}_{\boldsymbol{t}_2,\boldsymbol{q}_1}\boldsymbol{\mathcal{C}}_{\boldsymbol{t}}^\top + \\ &\quad \boldsymbol{R}_1\boldsymbol{\Sigma}_{\boldsymbol{t}_2,\boldsymbol{t}_1} + \boldsymbol{\Sigma}_{\boldsymbol{t}_1,\boldsymbol{q}_1}\boldsymbol{\mathcal{C}}_{\boldsymbol{t}}^\top + \boldsymbol{\Sigma}_{\boldsymbol{t}_1,\boldsymbol{t}_2}\boldsymbol{R}_1^\top \end{aligned} \tag{15}$$

where

$$\boldsymbol{R}_1 = (\boldsymbol{P}_{1,1}\boldsymbol{q}_1, \boldsymbol{P}_{1,2}\boldsymbol{q}_1, \boldsymbol{P}_{1,3}\boldsymbol{q}_1) \tag{16}$$

$$\boldsymbol{R}_2^\top = \left(\boldsymbol{Q}_{2,1}^\top\boldsymbol{q}_2, \boldsymbol{Q}_{2,2}^\top\boldsymbol{q}_2, \boldsymbol{Q}_{2,3}^\top\boldsymbol{q}_2\right)^\top. \tag{17}$$

and $\boldsymbol{Q}_{2,1}^\top, \boldsymbol{Q}_{2,2}^\top, \boldsymbol{Q}_{2,3}^\top$ have very similar forms with (1) [4]. Other related terms are

$$\begin{aligned} \boldsymbol{\mathcal{C}}_{\boldsymbol{R}} &= 2\sum\nolimits_{i=1}^{3}\left[\left(\boldsymbol{Q}_{2,i}^\top\boldsymbol{q}_2\right) \otimes \boldsymbol{P}_{1,i}\right] \\ \boldsymbol{\mathcal{C}}_{\boldsymbol{t}} &= 2\sum\nolimits_{i=1}^{3} t_{2,i}\boldsymbol{P}_{1,i}. \end{aligned} \tag{18}$$

E. Parameters of the Stewart Platform

The parameters of Stewart platform in Fig. 9 of original paper are described here. The six coordinates of the base and moving platform are given by (in meters)

$$\text{base}: \begin{matrix} 0.1449, 1.0, 0.03882 \\ -0.03882, 1.0, 0.1449 \\ -0.1061, 1.0, 0.1061 \\ -0.1061, 1.0, -0.1061 \\ -0.03882, 1.0, -0.1449 \\ 0.1449, 1.0, -0.03882 \end{matrix}, \ \text{plat}: \begin{matrix} 0.07071, 1.0, 0.07071 \\ 0.02588, 1.0, 0.09659 \\ -0.09659, 1.0, 0.02588 \\ -0.09659, 1.0, -0.02588 \\ 0.02588, 1.0, -0.09659 \\ 0.07071, 1.0, -0.07071 \end{matrix}.$$

The height of the platform in the identity pose is 0.15m. The true poses of the two cases in two lines of Fig. 9 are

$$\begin{aligned} \boldsymbol{q} &= (0.96449, 0.020127, -0.059542, 0.25654)^\top \\ \boldsymbol{t} &= (0.001219, 0.009613, 0.02227)^\top \ \ (\mathrm{m}) \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{q} &= (0.98266, -0.082391, 0.094327, -0.13672)^\top \\ \boldsymbol{t} &= (-0.00655, -0.005118, 0.004117)^\top \ \ (\mathrm{m}). \end{aligned}$$

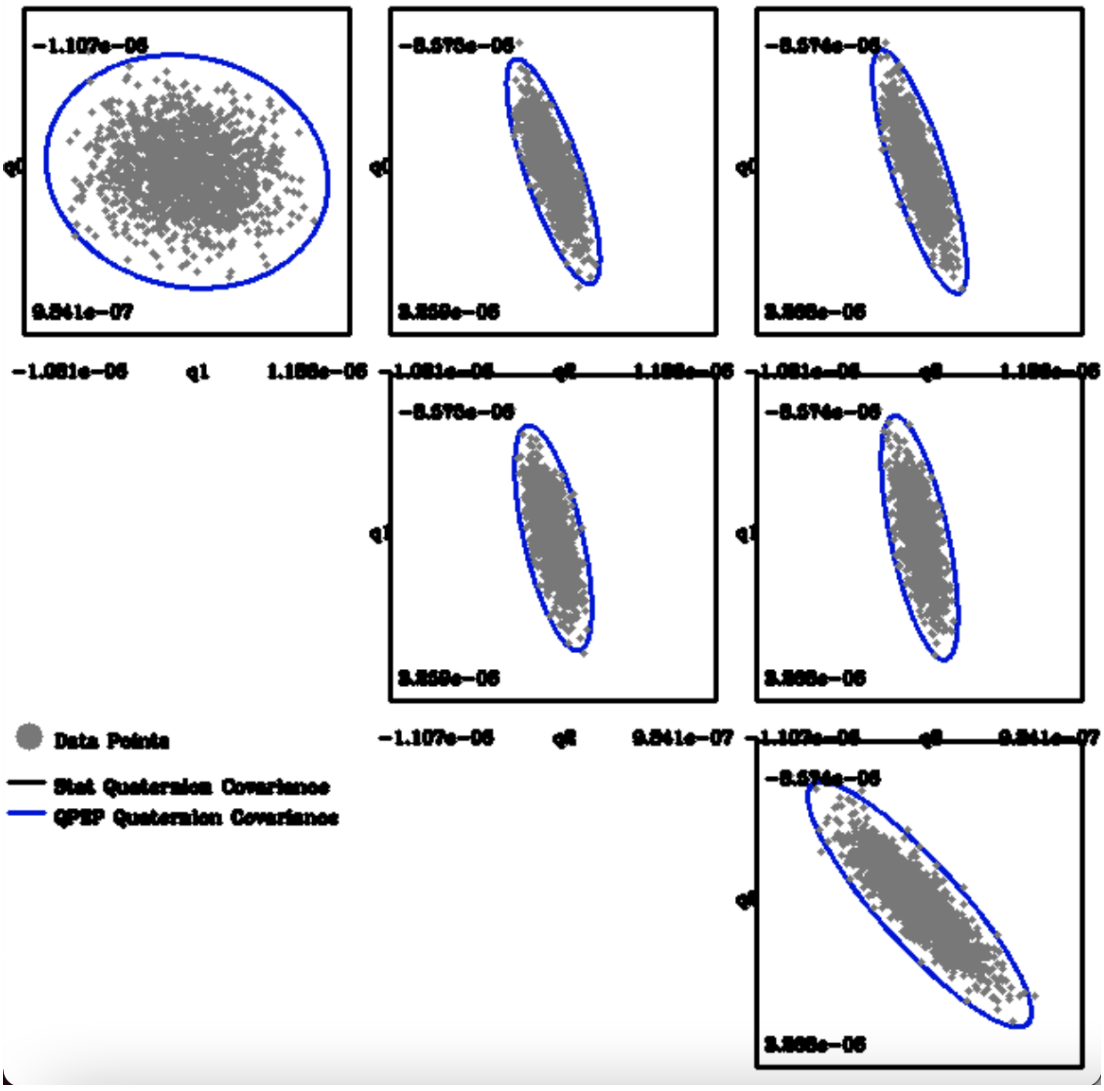
II. Engineering Applications and Examples

A. Test the Covariance of PnP or Point-to-Plane Registration

After following the instructions in `https://github.com/zarathustr/LibQPEP`, you can now evaluate the covariance of PnP or point-to-plane registration. The following codes in `main.cpp` specifies which method to be tested in the loop.

```
//TODO: Change this to METHOD_PTOP
//      if you need to test Point-to-Plane Registration
method = METHOD_PNP;
```

After building the `LibQPEP`, please feel free to run the executable, provided that you have installed `OpenCV`. The following picture occurs, if all stuffs are correct:



B. ICP Mapping

The proposed QPEP method is capable of solving point-to-plane registration in a globally optimal manner. To test the real ICP mapping performance, please first come to `https://github.com/zarathustr/libpointmatcher_QPEP` for a QPEP-based `libpointmatcher` library. Please follow the instructions to build and install `LibQPEP` and `libpointmatcher_QPEP` to your preferred install prefix. If you have robotic operating system (ROS) installed, you can visualize the results via RViz. Please clone the repository to your `catin` workspace: `https://github.com/zarathustr/icp_mapping_QPEP`. After building and installation, please come to `ethzasl_icp_mapper/launch` to choose the `roslaunch` file that you want. Currently, multiple devices and datasets are supported, including Kinect, KITTI dataset (Velodyne HDL 64-E Lidar), I.C.Sens dataset (Velodyne HDL 128 Lidar). Typical mapping results are shown as follows:

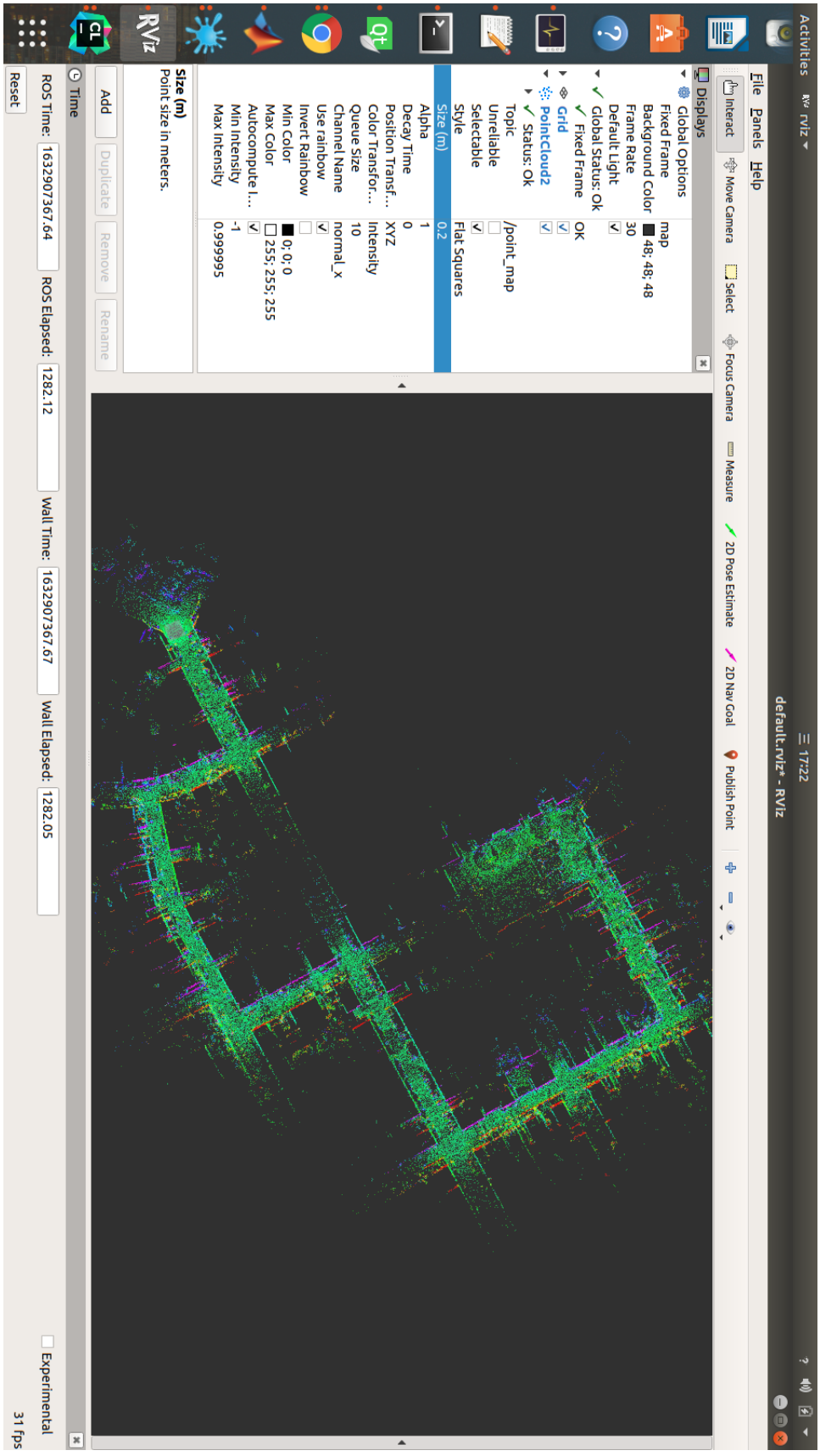


Fig. 3. The KITTI mapping results.

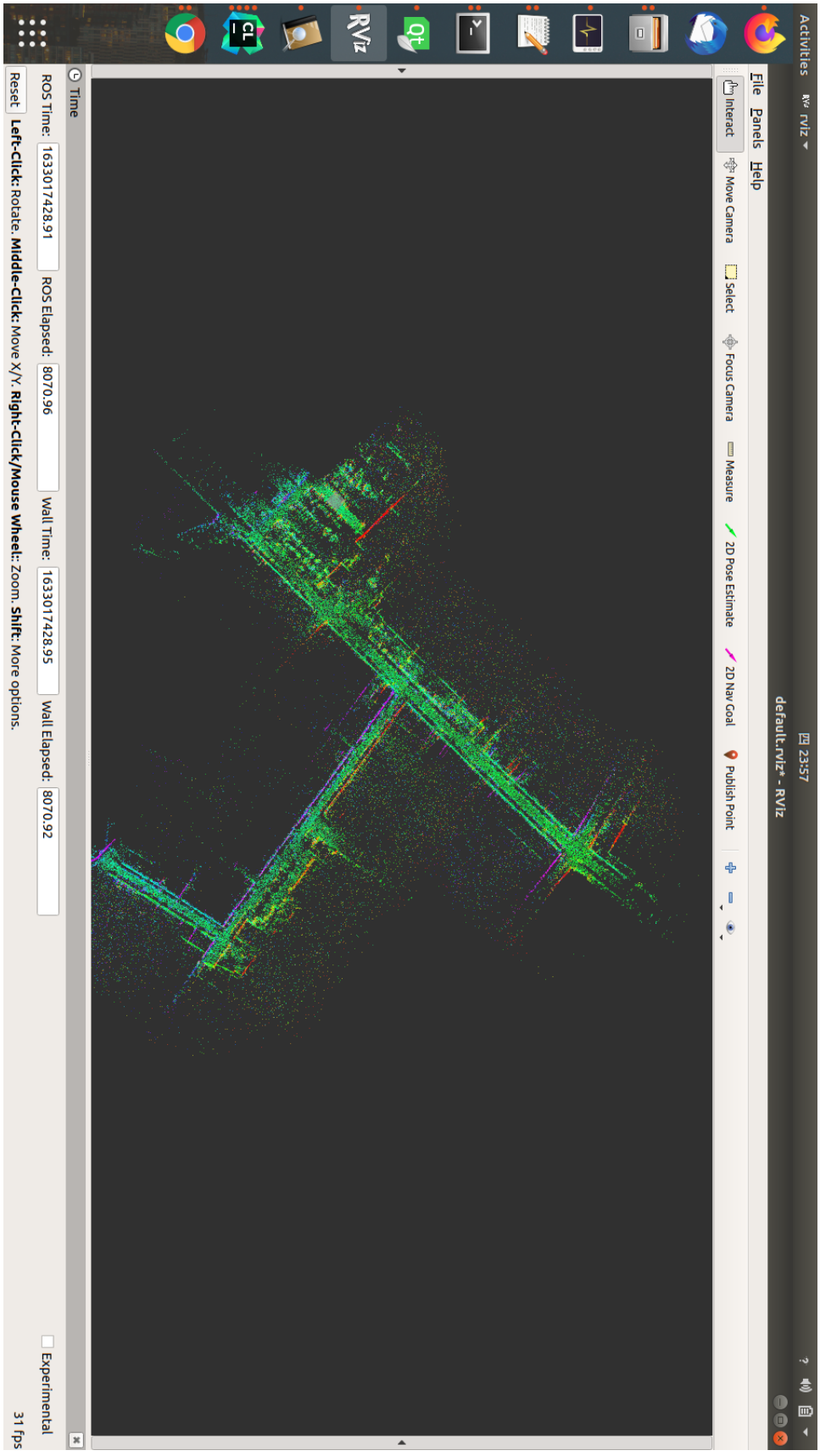
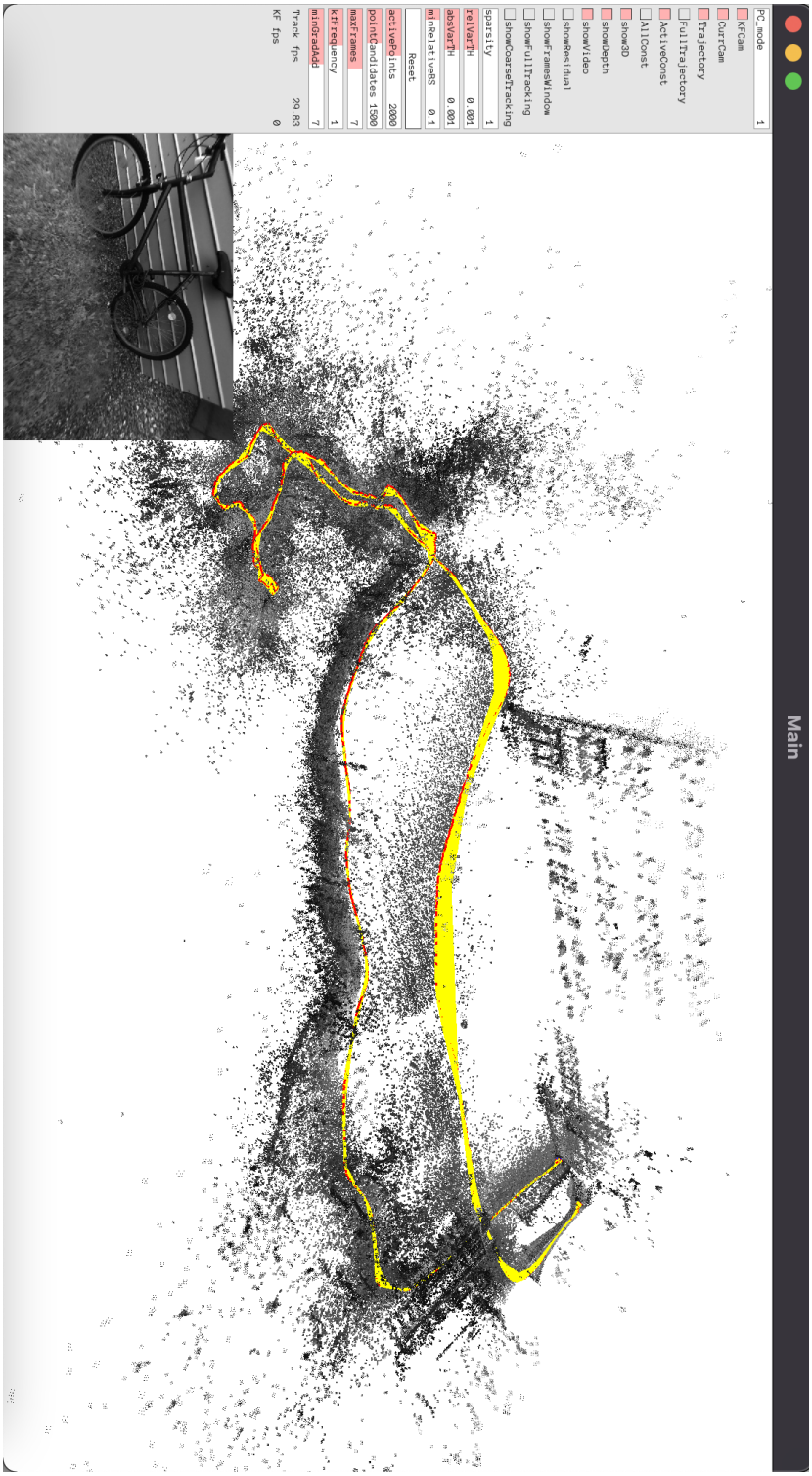


Fig. 4. The I.C.Sens mapping results.

C. Loop Closure of SLAM System

The PnP algorithm is sometimes utilized for loop closure pose estimation in SLAM system. The proposed QPEP method currently supports the loop closure system of the direct sparse odometry (DSO). Please come to `https://github.com/zarathustr/DSO-QPEP-Loop-Closure` for codes. After installation, a typical run performance is shown as



REFERENCES

[1] H. Nguyen and Q.-C. Pham, "On the Covariance of X in $\boldsymbol{AX} = \boldsymbol{XB}$," *IEEE Trans. Robot.*, vol. 34, no. 6, pp. 1651–1658, 2018.

[2] J. F. Sturm, "Using SeDuMi 1.02, A MATLAB Toolbox for Optimization over Symmetric Cones," *Optim. Methods and Softw.*, vol. 11, no. 1-4, 1999.

[3] J. Wu, Z. Zhou, B. Gao, R. Li, Y. Cheng, and H. Fourati, "Fast Linear Quaternion Attitude Estimator Using Vector Observations," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 1, pp. 307–319, 2018.

[4] J. Wu, M. Liu, and Y. Qi, "Computationally Efficient Robust Algorithm for Generalized Sensor Calibration Problem $\boldsymbol{AR} = \boldsymbol{RB}$," *IEEE Sens. J.*, vol. 19, no. 20, pp. 9512–9521, 2019.