Supplemental Material

A. Deriving the second order approximations (DQ2ndOrd)

In this appendix we give a detailed derivation of the expanded solution in small μ around the relaxed solution (21). We first define an expansion of the quaternion solution (eigenvector) and the smallest eigenvalue of (12)

$$q = \sum_{k=0}^{\infty} \tilde{q}_k \mu^k, \quad \lambda = \sum_{k=0}^{\infty} \tilde{\lambda}_k \mu^k, \tag{31}$$

such that (12) is solved order by order in μ, and the constraint $q^2 = 1$ is satisfied. We can fix μ by using the second constraint $q \cdot q' = 0$. We use $Z_0 q_a = \lambda_a q_a$, where we order the solutions so that $\lambda_0 < \lambda_1 < \lambda_2 < \lambda_3$, as a basis for the quaternions at every order,so that

$$\tilde{q}_k = \sum_{a=0}^{3} c_{k,a} q_a, \tag{32}$$

where $c_{k,a}$ are coefficients to be fixed. Plugging these into (12) and the constraint equations we find the following recursion relations (for $k \geq 1$)

$$\begin{aligned} c_{k,0} &= -\frac{1}{2}\sum_{n=1}^{k-1} \tilde{q}_{k-n} \cdot \tilde{q}_n, \\ \tilde{\lambda}_k &= q_0 (Z_1 \tilde{q}_{k-1} - Z_2 \tilde{q}_{k-1}) - \sum_{l=1}^{k-1} \tilde{\lambda}_{k-l} c_{l,0}, \\ c_{k,a} &= \frac{q_a (Z_1 \tilde{q}_{k-1} - Z_2 \tilde{q}_{k-2}) - \sum_{l=1}^{k-1} \tilde{\lambda}_{k-l} c_{l,a}}{\lambda_0 - \lambda_a}, \end{aligned} \tag{33}$$

with $c_{0,0} = 1$, $c_{0,a} = 0$, $\tilde{q}_{k<0} = 0$, and $\tilde{\lambda}_0 = \lambda_0$. Thus we can easily compute $\lambda(\mu)$ to arbitrary order in μ, and then solve $\frac{d\lambda}{d\mu}|_{\mu^*} = 0$. Since this is a polynomial equation in μ, we can solve it analytically for low orders. In practice we find that the second order approximation yields extremely good results. In that case we have

$$\mu^*_{(2)} = \frac{q_0 Z_1 q_0 / 2}{q_0 Z_2 q_0 - \sum_{a=1}^{3} \frac{(q_a Z_1 q_0)^2}{\lambda_0 - \lambda_a}}. \tag{34}$$

After getting $q_{(2)}$ (q to second order in μ) using $\mu^*_{(2)}$ and the expansion above and normalizing it, we get the corresponding q' using (20).

In the main text we described another possible expansion in the dimensionless parameter λ. As explained in that sec. 3.2.4, we expand again around the relaxed solution (and not around $\lambda = 0$), so $\lambda = \lambda_0 + \Delta\lambda$ and the expansion is in $\Delta\lambda$, and λ_0 corresponds to the smallest eigenvalue of Z_0. Carrying the expansion to second order leads to

$$\begin{aligned} &\tilde{\mu}_0 = 0, \quad , \tilde{\mu}_1 = \frac{1}{q_0 Z_1 q_0}, \quad \tilde{\mu}_2 = -\frac{\tilde{\mu}_1 q_0 Z_1 \tilde{q}_1 + \tilde{\mu}_1^2 q_0 Z_2 q_0}{q_0 Z_1 q_0}, \\ &\tilde{q}_1 = \sum_{a=1}^{3} \frac{\tilde{\mu}_1 q_a Z_1 q_0}{\lambda_0 - \lambda_a} q_a, \\ &\tilde{q}_2 = \sum_{a=1}^{3} \frac{q_a Z_1 (\tilde{\mu}_1 \tilde{q}_1 + \tilde{\mu}_2 q_0) + \tilde{\mu}_1^2 q_a Z_2 q_0 - q_a \tilde{q}_1}{\lambda_0 - \lambda_a} q_a - \frac{\tilde{q}_1^2}{2}. \end{aligned} \tag{35}$$

Using these expressions we can compute the constraint to second order

$$\begin{aligned} 0 = &q(\mu Z_2 + \frac{Z_1}{2}) q = \frac{1}{2} q_0 Z_1 q_0 + \Delta\lambda q_0 \left(Z_1 \tilde{q}_1 + \tilde{\mu}_1 Z_2 q_0\right) \\ &+ \Delta\lambda^2 \left(q_0 Z_1 \tilde{q}_2 + \frac{1}{2} \tilde{q}_1 Z_1 \tilde{q}_1 + q_0 Z_2 (2\tilde{\mu}_1 \tilde{q}_1 + \tilde{\mu}_2 q_0) \right) \\ \equiv &c_0 + \Delta\lambda c_1 + \Delta\lambda^2 c_2, \end{aligned} \tag{36}$$

solve for $\Delta\lambda = (-c_1 + \sqrt{c_1^2 - 4c_0 c_2})/2$ and plug for $\tilde{\mu} = \Delta\lambda(\tilde{\mu}_1 + \tilde{\mu}_2 \Delta\lambda)$, and compute the corresponding solution for q and then the unique q' using (20).

B. Optimal polynomial solution (DQOptPoly)

In sec. 3.1.2 we explained that the optimal solution to the hand-eye calibration problem can be found in a closed form by solving a 28-th degree univariant polynomial. However, the explicit polynomial coefficients in terms of the matrix entries of (12) are quite complicated, consisting of thousands of terms which makes this approach less attractive for practical use. Therefore, we presented an efficient way to arrive at the solution without directly solving the polynomial. In this appendix we present two more approaches to arrive at the optimal solution which are also based on e 1D line search, but of a very different function. These methods yield the same solution, but are a bit slower than the one presented in sec. 3.1.2. However, since they are conceptually different we find it useful to describe them in this section.

Throughout this section we assume that the 25 polynomial coefficient of the algebraic curve (15) have been extracted from the matrices Z_i (their explicit form is quite lengthy and not very illuminating, so we do not state it here. In any case it is straight forward to extract it from (15).).

An alternative approach is to find the λ root of the resultant for the μ polynomial with λ dependent coefficients numerically. Namely finding the root of a function that return the resultant given the 25 coefficients and λ which is easy to compute (in contrast to the explicit coefficients of the 28-degree polynomial). Alternatively we can repeat the same procedure for the λ polynomial with μ dependent coefficients. The root search can be bounded, see for example appendix C.

Another approach to arrive at the solution is to count the number of real roots of the 8-th degree μ polynomial with λ

dependent coefficients. For $\lambda = 0$ we have eight real roots, this number remains as we increase λ, and jumps to six exactly as we pass the optimal solution (λ corresponding to the optimal solution). The analytic expression for the number of roots is known in terms of Sturm's theorem. To compute the number of real roots one needs to compute the Sturm sequence defined by $P_0(\mu)$ which is the polynomial, $P_1 = \dot{P}_0$, $P_2 = -\text{rem}(P_0, P_1)$ and so on up to $P_8 = -\text{rem}(P_6, P_7)$ which is a constant. Then, plugging $\pm\infty$ and computing the difference in the number sign changes between the elements of the sequence gives the number of real roots. Thus, we can set the polynomial at $\lambda = 0$ such that the sequences signs at $-\infty$ and $+\infty$ are $(-, +, -, +, -, +, -, +, -)$ and $(+, +, +, +, +, +, +, +, +)$ respectively, which in turn implies there are eight real solutions. A drop from eight to six real solutions occurs when the first (P_0) or the last (P_8) term in the sequence changes sign, however, the sign of $P_0(\pm\infty)$ does not depend on λ, so we are only interested in the λ dependence of P_8, and more concretely, on the minimal positive root of P_8 with respect to λ which corresponds to our solution.

As before, the explicit analytic expression is quite involved and in practice one can use 1D numerical root finder for the explicit real roots counting number function to detect the jump from eight to six.

C. Analytic bounds on the parameter space

We can analytically bound the Lagrange multiplier variables as follows. First, we are only interested in the $\lambda \geq 0$ region, since as mentioned in the main text λ equals the cost function, which is non-negative by construction, and so any λ corresponding to a solution must be non-negative.

Next, we can also bound μ by noticing that we can rewrite (14) as $\mu = \frac{(Lq)^T K (Lq)}{|Lq|^2}$, where $M^{-1} = L^T L$ is the Cholesky decomposition and $K \equiv \frac{1}{2}(L W^T L^{-1} + L^{-T} W L^T)$. From this expression we get a bound on μ in terms of the largest and smallest eigenvalues of K.

D. Experiments

We validate out algorithms on a combination of real world and synthetic datasets. Each dataset consists of a set of corresponding noisy measurements $\Delta\tilde{P}_1^i$, $\Delta\tilde{P}_2^i$ of relative poses from two rigidly attached sensors moving in 3D.

From each dataset draw $N = 1000$ samples of $n = 100$ corresponding relative poses (with replacement between the samples) from which we generate a histogram of translation and rotation errors for each algorithm. In order to reduce the impact of temporal correlations within the samples the n relative poses are drawn randomly from entire trajectory and are thus not necessarily consecutive in time.

In order to account for the effect of the weighting parameter we repeat the experiments for 100 values of α evenly distributed on a logarithmic scale between 10^{-2} and $10^{1.7}$ (we found that no algorithm performs well outside this range for the used datasets). In Table 2 we display the median rotation and translation error together with the 25% and 75% percentiles for the α that gives the lowest median rotation and translation error respectively for each dataset and algorithm.

D.1. Real data experiments

First, we validate our algorithms on four datasets generated by a multi-camera tracking system. Two ZED stereo cameras[4] were rigidly attached at circa 28 cm from each other at an angle of $50°$, maintaining partially overlapping fields of view as shown in fig. 4. The multi-camera setup was manually moved on a large area (circa 130 m^2) following several 3D trajectories shown in fig. 5.

Two SLAM components running on the stereo camera provided 6 d.o.f. poses at 5 Hz (C_i and H_i in fig. 1). We ran the two visual tracking systems at limited resolution settings (1280x720 pixels) independently from each sensor, *i.e.* no shared features and different reference frames as shown in fig. 1). Loop closure and mapping was also disabled to avoid drift and scale correction over long, redundant trajectories. The true extrinsic X was estimated offline, by placing a large fiducial marker in the overlapping FOV of the cameras. Multiple high resolution pictures (3840x2160 pixels), taken from the two cameras, were used to determine the position of the marker w.r.t. the camera optical centers. The sensor manufacturer's intrinsics and extrinsics were then used to compute the relative pose between the two stereo camera centers, corresponding to the tracked points. The estimated relative pose (X in fig. 1) resulting from this calibration procedure had translation: [-0.7, 28.1, -0.1] cm and axis-angles: [2.35, -0.92, -48.93] deg.

In order to reduce the impact of outliers from the tracking algorithms we pre-filtered the data by discarding any datapoints where either of the cameras has moved more than 10 cm or $11.5°$ between two consecutive frames, as that indicates tracking error. These thresholds were empirically determined considering the speed with which the two cameras were moved during the experiments. In order to reduce the effect of synchronization issues we also removed any datapoints where the timestamps from the cameras differ by more than 0.1 seconds, the poses that are left differ in timestamp by roughly 0.05 seconds.

In our experiments we recorded four types of motion (see fig. 5). First we recorded a general 3d motion where the cameras device was moved freely by hand in a large room. The other types of motion are approximately planar, where we placed the device on a cart which was moved by hand across the room. In this way we created three more trajectories: a general planar motion, a linear motion (where

[4]ZED stereo camera: https://www.stereolabs.com/zed/

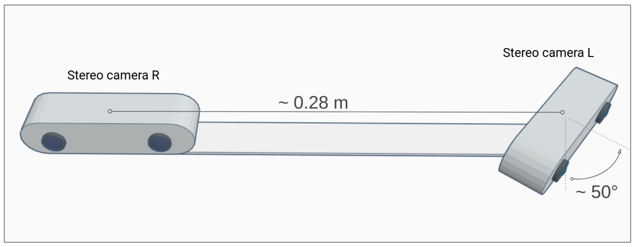

Figure 4: Our experimental setup. Two rigidly attached stereo cameras.

the cart was moved only forward) and a circular motion (where the cart was rotated).

D.2. Synthetic datasets

We also test our algorithm on three different scenarios of synthetically generated data, where the ground-truth is known exactly. For each scenario we generate a set of N corresponding relative poses ΔP_1^i, ΔP_2^i with

$$\Delta P_2^i = X \circ \Delta P_1^i \circ X^{-1}. \tag{37}$$

We simulate noisy measurements $\Delta \tilde{P}_k^i$ of ΔP_k^i by applying small $SE(3)$-perturbations:

$$\Delta \tilde{P}_k^i = \Delta P_k^i \circ [\delta R_k^i(\sigma_r) | \delta t_k^i(\sigma_t)], \tag{38}$$

where $\delta R_k^i(\sigma_r)$ is generated by drawing its rotation axis from a uniform $SO(3)$ distribution [15] and its angle from a normal distribution[5] with variance σ_t^2. $\delta t_k^i(\sigma_t)$ is generated by drawing its elements from a normal distribution with variance σ_t^2. In Table 2 we have used $\sigma_r = 0.57°$ and $\sigma_t = 0.01$ m for the pose noise.

We consider the three following scenarios:

- **Random** The relative poses ΔP_1^i of the first trajectory are generated by drawing the rotational part from a uniform distribution on $SO(3)$ [15], and the translational part from $U(0,1)^3$.
- **Line** Constant velocity along the x-axis from $x = 0$ to $x = 2$ and constant orientation.
- **Circle** One revolution of a circle with radius 2.

In order to break the symmetry for the line and circle scenarios we apply a perturbation with $\sigma_r = 0.57°$ and $\sigma_t = 0.01$ m to each ΔP_1^i before computing ΔP_2^i, simulating a slight jitter in the trajectory. For the Random scenario we plot the algorithm performance as a function of α at various noise levels in fig. 6.

[5]For simplicity we use a normal distribution for the angle, though it would be more appropriate to use the more complicated wrapped normal distribution that takes the periodicity into account.

E. Algorithms summary

In this appendix we summarize in pseudo-code our algorithms which appear in the main text. We describe the algorithms after processing the data, namely, given the matrices S, W and M in (9). The Z_i matrices are defined in (12). Throughout this section, by smallest eigenvector we mean the eigenvector corresponding to the smallest eigenvalue.

```
Data: $S, W, M$
Result: $q_*, q'_*$
1: function $f(\mu)$
2:     $q :=$ smallest Eigenvector$(Z_0 + \mu Z_1 - \mu^2 Z_2)$
3:     return $q^T \left(\mu Z_2 - \frac{1}{2} Z_1\right) q$
4: Use root finding procedure to solve, $f(\mu_*) = 0$
5: $q_* :=$ smallest Eigenvector$(Z_0 + \mu_* Z_1 - \mu_*^2 Z_2)$
6: $q'_* := Z_2(\mu_* - W^T) q_*$
```
Algorithm 1: 1D line search (**DQOpt**)

```
Data: $W, M$
Result: $q_*, q'_*$
1: $q_* :=$ smallest Eigenvector$(M)$
2: $q'_* := Z_2 \left(\frac{1}{2} \frac{q_*^T Z_1 q_*}{q_*^T Z_2 q_*} - W^T\right) q_*$
```
Algorithm 2: 2 steps algorithm (**DQ2Steps**)

```
Data: $S, W, M$
Result: $q_*, q'_*$
1: $q_* :=$ smallest Eigenvector$(Z_0)$
2: $q'_* := Z_2 \left(\frac{1}{2} \frac{q_*^T Z_1 q_*}{q_*^T Z_2 q_*} - W^T\right) q_*$
```
Algorithm 3: Convex relaxation algorithm (**DQConvRlx**)

```
Data: $S, W, M$
Result: $q_*, q'_*$
1: Solve $Z_0 q_i = \lambda_i q_i$ for $q_i$ and $\lambda_i$, $i = 0, ..., 3$
2: Compute $q_{(2)}$ using (23)
3: $q_* = q_{(2)} / q_{(2)}^2$
4: $q'_* := Z_2 \left(\frac{1}{2} \frac{q_*^T Z_1 q_*}{q_*^T Z_2 q_*} - W^T\right) q_*$
```
Algorithm 4: second order algorithm (**DQ2ndOrd**)

```
Data: $S, W, M, \epsilon$
Result: $q_*, q'_*$
$\mu := 0$;
while $\Delta > \epsilon$ do
    $q :=$ smallest Eigenvector$(Z_0 + \mu Z_1 - \mu^2 Z_2)$;
    $\mu_{\text{new}} = \frac{1}{2} \frac{q(\mu)^T Z_1 q(\mu)}{q(\mu)^T Z_2 q(\mu)}$;
    $\Delta := |\mu - \mu_{\text{new}}|$;
    $\mu := \mu_{\text{new}}$;
end
$q_* := q$
$q'_* := Z_2(\mu_* - W^T) q_*$
```
Algorithm 5: Iterative algorithm (**DQItr**)

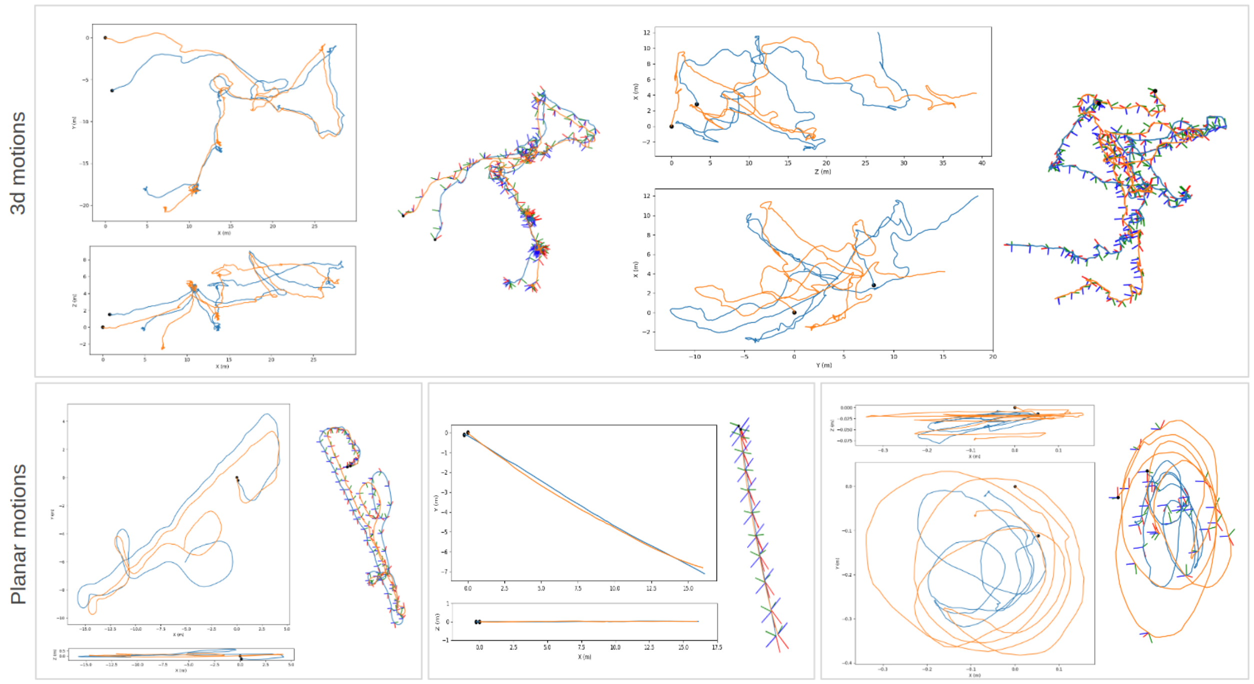

Figure 5: The two estimated camera trajectories from the real data experiments shown in blue and orange respectively. We show both 2d projections and a 3d plot for each dataset. The data used here has been filtered by taking away relative poses with large errors in rotation or translation w.r.t. the ground truth calibration, which we consider as outliers where the SLAM algorithm fails. The upper plots show the general 3d motion trajectories and the lower plots show the Planar, Linear and Rotation trajectories from left to right.

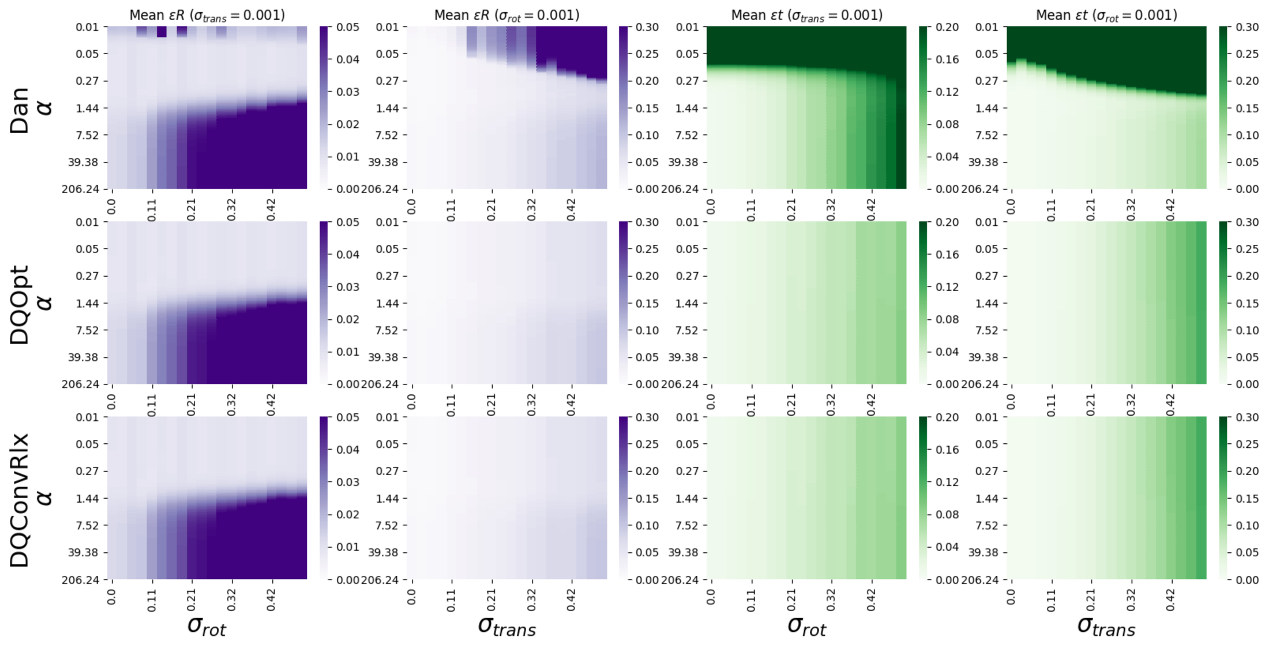

Figure 6: Effect of α on the translation and rotation errors for **Dan**, **DQOpt** and **DQConvRlx** for the Random synthetic scenario with various noise levels. The first two columns (in purple) show the mean rotation errors and the last two (in green) show the mean translation errors, each averaged over 60 samples. We vary the noise level on the translation or rotation component respectively, while keeping the noise on the remaining component fixed.