Appendix for Six-Point Method for Multi-Camera Systems with Reduced Solution Space

Banglei Guan[1†], Ji Zhao[2*†], Saibal Mitra[3], Laurent Kneip[4]

[1]College of Aerospace Science and Engineering, National University of Defense Technology, Changsha, China.
[2*]Independent Researcher, Beijing, China.
[3]Independent Researcher, Hoedekenskerke, Netherlands.
[4]Mobile Perception Lab, ShanghaiTech University, Shanghai, China.

*Corresponding author(s). E-mail(s): zhaoji84@gmail.com;
Contributing authors: guanbanglei12@nudt.edu.cn;
smitra00@gmail.com; lkneip@shanghaitech.edu.cn;
†These authors contributed equally to this work.

Appendix A Proof of a Common Factor

In Section 3.3 of the paper, we say that a factor $q_x^2 + q_y^2 + q_z^2 + 1$ can be factored out in Eqs. (14) and (15) of the paper. We show that this property can be also proven by symbolic computation using Matlab. The Matlab code is listed in Listing 1.

Listing 1: Proof by Symbolic Computation using Matlab

```
%% define an orthogonal matrix via Cayley's formula
syms x y z real
Q = [1+x^2-y^2-z^2, 2*x*y-2*z, 2*y+2*x*z; 2*x*y+2*z, 1-x^2+y^2-z^2,
     2*y*z-2*x; 2*x*z-2*y, 2*x+2*y*z, 1-x^2-y^2+z^2];

%% check orthogonality
% The results are s^2*diag([1, 1, 1]), where s = x^2+y^2+z^2+1.
simplify(Q*Q')

%% construct a 3*3 random matrix N in Eq.(15)
imax = 10;
c = cell(3, 1);
for i = 1:3
    c{i} = cross(randi(imax, [3,1]), Q*randi(imax, [3,1]));
end
C1 = [c{1}, c{2}, c{3}];
eq1 = det(C1);

%% check the determinant has factor x^2+y^2+z^2+1
factor(eq1)

%% construct a 4*4 random matrix N in Eq.(14)
c = cell(4, 1);
mm = [x^2; y^2; z^2; x*y; x*z; y*z; x; y; z; 1];
for i = 1:4
    tmp = randi(imax, [1, 10])*mm;
    c{i} = [cross(randi(imax, [3,1]), Q*randi(imax, [3,1])); tmp];
end
C2 = [c{1}, c{2}, c{3}, c{4}];
eq2 = det(C2);

%% check the determinant has factor x^2+y^2+z^2+1
factor(eq2)
```

It can be seen that 3×3 and 4×4 submatrices of $\mathbf{M}(q_x, q_y, q_z)$ in Eq. (8) of the paper have a common factor $q_x^2 + q_y^2 + q_z^2 + 1$. Note that representing all variables with symbols, that is, changing random numbers in the code to symbols, makes a rigorous proof. However, the runtime of symbolic computation will become about 15 hours.

Based on the existence of the common factor, we can use this property to factor out $q_x^2 + q_y^2 + q_z^2 + 1$ which simplifies the equation system. It leads to the generation of more efficient solvers while also potentially avoiding false roots.

Appendix B Graph Enumeration Using Matlab

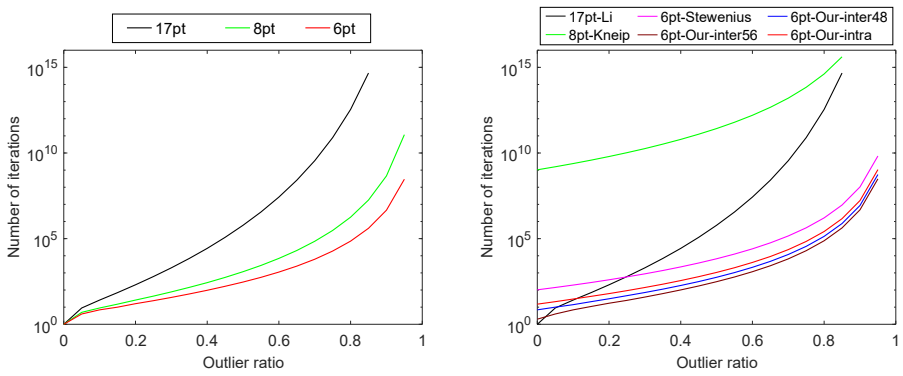
The Matlab code is shown in Listing 2.

Listing 2: Enumerate Distinct Graphs Using Matlab

```
function G_curr = generate_distict_graph(edge_num)
G_curr = {};
for k = 1:edge_num
  G_prev = G_curr;
  G_curr = generate_graph_recursion(G_prev, k);
end

function G_curr = generate_graph_recursion(G_prev, k)
if k == 1
  G_curr = cell(1, 2);
  G_curr{1} = struct('E', [1; 1]);
  G_curr{2} = struct('E', [1; 2]);
  return;
end
curr_sz = 0;
G_curr = {};
for i = 1:numel(G_prev)
  E0 = G_prev{i}.E;
  edge_all = generate_edge_set(E0);
  for j = 1:size(edge_all, 2)
    E = [E0, edge_all(:, j)];
    G = digraph(E(1, :), E(2, :));
    H = digraph(E(2, :), E(1, :));
    if ~is_in_set(G, H, G_curr, curr_sz)
     curr_sz = curr_sz + 1;
     G_curr{curr_sz} = struct('E', E, 'G', G);
    end
  end
end
fprintf('#edge: %d, #found-graphs: %d\n', k, curr_sz);

function edge_all = generate_edge_set(E)
s0 = unique(E(:));
s = [s0(:); max(s0)+1];
n = numel(s);
edge_all = zeros(2, n*n+1);
i1 = kron(1:n, ones(1,n));
i2 = kron(ones(1,n), 1:n);
edge_all(:, 1:n^2) = [s(i1), s(i2)]';
edge_all(:, end) = max(s0)+[1; 2];

function flag = is_in_set(G, H, G_set, curr_sz)
flag = false;
  for ii = 1:curr_sz
    G1 = G_set{ii}.G;
    if (isisomorphic(G1, G) || isisomorphic(G1, H))
      flag = true;
    return;
  end
end
```

Appendix C Degenerate Configurations

For two-camera rigs, there are degenerate cases for `6pt+cayl/quat+generic`. The devised solvers `6pt+cayl/quat+inter` and `6pt+cayl/quat+intra` have better performance than the generic solver in those cases. We also introduce two degenerate cases



(a) Iteration number without accounting for numerical stability.

(b) Iteration number accounting for numerical stability.

Fig. D1: The relationship between the RANSAC iteration number and the outlier ratio in the context of achieving a success probability of 99%.

for these two solvers for two-camera rigs: (1) For inter-camera case, a two-camera rig undergoes pure translation while the baselines of the two cameras are parallel to the translation direction, (2) For intra-camera case, a two-camera rig undergoes pure translation or the cameras move along concentric circles. In these critical configurations, the metric scale of the translation is unobtainable.

Appendix D Experiments

D.1 Expected Runtime vs. Inlier Ratio Accounting for Numerical Stability

The computational time of RANSAC is contingent upon both the runtime of the minimal solver and the number of iterations. For the RANSAC methods, the iteration number is determined by the probability of choosing at least one inlier set. This calculation implicitly assumes that an accurate solution can be obtained given an inlier set. However, this is not always true when considering the numerical stability of the minimal solvers.

Denote p is the probability (usually set to 0.99) that at least one inlier set is sampled. n is the minimum number of required data points in the minimal solver. For example, $s = 6$ for the six-point solvers and $s = 17$ for the 17-point solver. ϵ is the outlier ratio, *i.e.*, the probability that any selected data point is an outlier. N is the iteration number of RANSAC. In N sampling times, the failure probability (one or more outliers are sampled in every sampling) is

$$(1 - (1 - \epsilon)^s)^N = 1 - p, \tag{D1}$$

and

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)}. \tag{D2}$$

When considering numerical stability, suppose p_2 is the probability of obtaining a sufficiently accurate solution given an inlier set. In $\hat{N}$ sampling times, the failure probability accounting for numerical stability is

$$(1-(p_2(1-\epsilon))^s)^{\hat{N}} = 1-p, \tag{D3}$$

and

$$\hat{N} = \frac{\log(1-p)}{\log(1-(p_2(1-\epsilon))^s)}. \tag{D4}$$

Since $p_2 \in [0,1]$ as a probability, we have $\hat{N} \geq N$ when accounting for numerical stability.

The left question is how to determine p_2 for a given minimal solver. For each minimal solver, we can empirically obtain the probability that the estimation errors $\varepsilon_{\mathbf{R}}$ and $\varepsilon_{\mathbf{t}}$ on noise-free observations are below specified thresholds. In this paper, we set the thresholds as 10^{-3}. For the solvers `17pt-Li` (Li et al., 2008), `8pt-Kneip` (Kneip and Li, 2014), `6pt-Stewénius` (Stewénius et al., 2005), `6pt-Our-inter56`, `6pt-Our-inter48` and `6pt-Our-intra`, p_2 can be computed as 1.00, 0.09, 0.61, 0.99, 0.90 and 0.81, respectively. Fig. D1 shows RANSAC iteration numbers with respect to the outlier ratio for success probability 99%.

References

Kneip, L., Li, H.: Efficient computation of relative pose for multi-camera systems. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 446–453 (2014)

Li, H., Hartley, R., Kim, J.-h.: A linear approach to motion estimation using generalized camera models. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2008)

Stewénius, H., Oskarsson, M., Aström, K., Nistér, D.: Solutions to minimal generalized relative pose problems. In: Workshop on Omnidirectional Vision in Conjunction with ICCV, pp. 1–8 (2005)