Notes on Colored Point Cloud Registration

Hanzhe Teng

April 2022

This note aims to provide in-depth details about the math used in the paper: Colored Point Cloud Registration Revisited, ICCV 2017

The most difficult step in an non-linear (least-squares) optimization problem is to find correct residuals and Jacobian matrices. Once they are identified, we can then find a descent direction in each iteration (though linearized locally) and repeat this procedure until convergence.

We will primarily focus on the math notes on residuals and Jacobian matrices for geometric and color components respectively.

1 Geometric Component

1.1 Residual

According to the setup in the paper, $\mathbf{p}$ is a point on the *target* cloud (which does not move), $\mathbf{q}$ is a point on the *source* cloud (which will be transformed in each iteration), and $\mathbf{n_p}$ is the normal of point $\mathbf{p}$.

The residual function of the geometric component is

$$r_G^{(\mathbf{p},\mathbf{q})}(\mathbf{T}) = (\mathbf{s}(\mathbf{q},\mathbf{T}) - \mathbf{p})^\top \mathbf{n_p} \tag{19}$$

where $\mathbf{s}(\mathbf{q},\mathbf{T}) = \mathbf{Tq}$, and $\mathbf{T}$ is the 4-by-4 transformation matrix and $\mathbf{q}$ is in the homogeneous coordinate. Intuitively, $\mathbf{s}(\mathbf{q},\mathbf{T})$ describes a point $\tilde{\mathbf{q}} = \mathbf{Tq}$ on the transformed source cloud (in the current iteration of the ICP algorithm). The point normal term $\mathbf{n_p}$ comes from the point-to-plane ICP algorithm. All the three terms $\tilde{\mathbf{q}}$, $\mathbf{p}$, and $\mathbf{n_p}$ are accessible directly in each iteration.

Discussion. Recall that a typical point-to-point ICP algorithm minimize the objective function

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \|\mathbf{p} - \mathbf{Tq}\|^2$$

and the point-to-plane ICP algorithm improves it by adding a point normal term in the objective function

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} ((\mathbf{p} - \mathbf{Tq}) \cdot \mathbf{n_p})^2$$

We can observe that if only the geometric component is used, the proposed algorithm is equivalent to a point-to-plane ICP algorithm.

1.2 Jacobian

Recall in the paper that each element in the Jacobian of the residual function with respect to ξ is

$$\nabla r_G^{(\mathbf{p},\mathbf{q})}(\mathbf{T}) = \mathbf{n}_\mathbf{p}^\top \mathbf{J}_\mathbf{s}(\xi) \tag{30}$$

where $\mathbf{J}_\mathbf{s}(\xi)$ is the Jacobian of $\mathbf{s}$ with respect to ξ, derived from (4) and (20). We illustrate the derivation and computation of $\mathbf{J}_\mathbf{s}(\xi)$ in the following.

Eq. (4) introduces the definition of $\mathbf{s}$, i.e. $\mathbf{s}(\mathbf{q},\mathbf{T}) = \mathbf{T}\mathbf{q}$. Eq. (20) describes the linearization of the pose vector. In each iteration, we lineralize $\mathbf{T}$ locally as a 6-vector $\xi = (\alpha, \beta, \gamma, a, b, c)$. $\mathbf{T}$ is approximated by a linear function of ξ.

$$\mathbf{T} \approx \begin{pmatrix} 1 & -\gamma & \beta & a \\ \gamma & 1 & -\alpha & b \\ -\beta & \alpha & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{T}^k \tag{20}$$

Specifically, let $\tilde{\mathbf{q}} = \mathbf{T}^k\mathbf{q}$ and denote $\tilde{\mathbf{q}} = (q_1, q_2, q_3)$ explicitly in its each entry. If homogeneous coordinate is needed, then $\tilde{\mathbf{q}} = (q_1, q_2, q_3, 1)$. Then we can have the following derivation.

$$\mathbf{s}(\mathbf{q},\mathbf{T}) = \mathbf{T}\mathbf{q} = \begin{pmatrix} 1 & -\gamma & \beta & a \\ \gamma & 1 & -\alpha & b \\ -\beta & \alpha & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix} \tilde{\mathbf{q}} = \begin{pmatrix} q_1 - \gamma q_2 + \beta q_3 + a \\ \gamma q_1 + q_2 - \alpha q_3 + b \\ -\beta q_1 + \alpha q_2 + q_3 + c \\ 1 \end{pmatrix}$$

Taking partial derivative of $\mathbf{s}(\mathbf{q},\mathbf{T})$ with respect to the 6-vector $\xi = (\alpha, \beta, \gamma, a, b, c)$, we can have

$$\mathbf{J}_\mathbf{s}(\xi) = \begin{pmatrix} 0 & q_3 & -q_2 & 1 & 0 & 0 \\ -q_3 & 0 & q_1 & 0 & 1 & 0 \\ q_2 & -q_1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

where the last row that comes from homogeneous coordinate is redundant. It is kept herein for completeness, but can be removed for simplicity. The result does not contain any entry of ξ thanks to the linearization formulation.

Plug it back into Eq. (30), we have

$$
\begin{aligned}
\nabla r_G^{(\mathbf{p},\mathbf{q})}(\mathbf{T}) &= \mathbf{n}_{\mathbf{p}}^\top \mathbf{J}_{\mathbf{s}}(\xi) \\
&= \begin{pmatrix} n_1 & n_2 & n_3 \end{pmatrix} \begin{pmatrix} 0 & q_3 & -q_2 & 1 & 0 & 0 \\ -q_3 & 0 & q_1 & 0 & 1 & 0 \\ q_2 & -q_1 & 0 & 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} n_3 q_2 - n_2 q_3 \\ -n_3 q_1 + n_1 q_3 \\ n_2 q_1 - n_1 q_2 \\ n_1 \\ n_2 \\ n_3 \end{pmatrix} \\
&= \begin{pmatrix} \tilde{\mathbf{q}} \times \mathbf{n}_{\mathbf{p}} \\ \mathbf{n}_{\mathbf{p}} \end{pmatrix}
\end{aligned}
$$

where we denote $\mathbf{n}_{\mathbf{p}} = (n_1, n_2, n_3)$ explicitly in each entry as needed. The final results $\tilde{\mathbf{q}} \times \mathbf{n}_{\mathbf{p}}$ and $\mathbf{n}_{\mathbf{p}}$ are the actual code we implement in C++. Only two terms $\tilde{\mathbf{q}}$ and $\mathbf{n}_{\mathbf{p}}$ are involved, and they are both accessible in each iteration.

Discussion. In fact, the linearized rotation matrix comes from the identity matrix and the bracket notation (skew-symmetric matrix so(3) representation) of $\xi' = (\alpha, \beta, \gamma)$.

$$
\begin{pmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & -\gamma & \beta \\ \gamma & 0 & -\alpha \\ -\beta & \alpha & 0 \end{pmatrix} = \mathbf{I} + [\xi']
$$

In the actual implementation, this linearized matrix is mapped to SO(3) using a multiplication of axis-angle representation of rotation matrices in an order of ZYX via Eigen library.

2 Color Component

In a typical ICP algorithm, two point clouds move closer to each other as the number of iteration increases (assuming that we are optimizing towards the correct direction). In this geometric-only setup, the residual is a *continuous function* of transformation $\mathbf{T}$. This indicates that we can observe various rewards (in the sense of residual errors) as we perturb the pose of the source cloud, and the best reward is obtained at the descent direction of the optimization.

However, this is normally not the case for color components (because perturbations happen in 3D space rather than color space). Once correspondences are established, the color differences (residuals) between source points and target points are fixed regardless of the pose of two point clouds. Then it would be impossible to estimate a transformation provided these constant residuals, because there is no information to indicate towards which direction we should move. Therefore, there is need to make color residual a continuous function of transformation as well.

Algorithm Outline. The method proposed in this paper is to introduce a virtual orthogonal camera for each point $\mathbf{p}$ in the target cloud. It is configured to observe $\mathbf{p}$ along the normal $\mathbf{n_p}$, and the image plane of this virtual camera is the tangent plane at $\mathbf{p}$. For each point $\tilde{\mathbf{q}} = \mathbf{s}(\mathbf{q}, \mathbf{T}) = \mathbf{Tq}$ in the transformed source cloud in current iteration, we project $\tilde{\mathbf{q}}$ onto the virtual image plane of its correspondence point $\mathbf{p}$ in the target cloud, using Eq. (9).

$$\mathbf{f}(\tilde{\mathbf{q}}) = \tilde{\mathbf{q}} - \mathbf{n_p}(\tilde{\mathbf{q}} - \mathbf{p})^\top \mathbf{n_p} \tag{9}$$

We denote the projected point as $\mathbf{q}' = \mathbf{f}(\tilde{\mathbf{q}})$. Then we take the directional vector $\mathbf{q}' - \mathbf{p}$ on the virtual image plane and compute its dot product with the color gradient at point $\mathbf{p}$ (denoted as $\mathbf{d_p}$). This is essentially a projection onto the gradient direction. Since the projected point $\mathbf{q}'$ is a continuous function of transformation $\mathbf{T}$, the term

$$\mathbf{d}_\mathbf{p}^\top(\mathbf{q}' - \mathbf{p})$$

we computed herein is also a continuous function of transformation $\mathbf{T}$. This is eventually the term we add to the residual function that makes the whole story different. We will discuss residuals and Jacobian matrices for color component in the following, and describe how we obtain the color gradient $\mathbf{d_p}$ in the next section.

2.1 Residual

Recall in the paper that the residual function of the color component is

$$r_C^{(\mathbf{p},\mathbf{q})}(\mathbf{T}) = C_\mathbf{p}(\mathbf{f}(\mathbf{s}(\mathbf{q}, \mathbf{T}))) - C(\mathbf{q}) \tag{18}$$

It can be re-organized as

$$\begin{aligned} r_C^{(\mathbf{p},\mathbf{q})}(\mathbf{T}) &= C_\mathbf{p}(\mathbf{f}(\tilde{\mathbf{q}})) - C(\mathbf{q}) \\ &= C_\mathbf{p}(\mathbf{q}') - C(\mathbf{q}) \end{aligned}$$

where $C(\mathbf{q})$ retrieves the intensity at point $\mathbf{q}$, and $C_\mathbf{p}(\mathbf{q}')$ is a continuous color function at point $\mathbf{p}$ that can change its value according to the position of the projected point $\mathbf{q}'$. The function $C_\mathbf{p}(\mathbf{q}')$ can be approximated by its first-order approximation $C_\mathbf{p}(\mathbf{q}') \approx C(\mathbf{p}) + \mathbf{d}_\mathbf{p}^\top(\mathbf{q}' - \mathbf{p})$ which retrieves the sum of the intensity at point $\mathbf{p}$ and the aforementioned continuous term $\mathbf{d}_\mathbf{p}^\top(\mathbf{q}' - \mathbf{p})$. The residual function ends up with the form

$$r_C^{(\mathbf{p},\mathbf{q})}(\mathbf{T}) \approx C(\mathbf{p}) + \mathbf{d}_\mathbf{p}^\top(\mathbf{q}' - \mathbf{p}) - C(\mathbf{q})$$

All the terms in this function are accessible in each iteration of the algorithm, and we can implement in C++ accordingly now. ($\mathbf{d_p}$ is a fixed term and can be computed during pre-processing. We will discuss the estimation of $\mathbf{d_p}$ in the last section.)

2.2 Jacobian

Recall in the paper that each element in the Jacobian of the residual function with respect to ξ is

$$\begin{aligned}\nabla r_C^{(\mathbf{p},\mathbf{q})}(\mathbf{T}) &= \frac{\partial}{\partial \xi_i}\left(C_{\mathbf{p}} \circ \mathbf{f} \circ \mathbf{s}\right) \\ &= \nabla C_{\mathbf{p}}(\mathbf{f}) \mathbf{J}_{\mathbf{f}}(\mathbf{s}) \mathbf{J}_{\mathbf{s}}(\xi)\end{aligned} \tag{28-29}$$

From the definition in the paper,

$$C_{\mathbf{p}}(\mathbf{f}) \approx C(\mathbf{p}) + \mathbf{d}_{\mathbf{p}}^{\top}\mathbf{f} \tag{8}$$

$$\mathbf{f}(\mathbf{s}) = \mathbf{s} - \mathbf{n}_{\mathbf{p}}(\mathbf{s}-\mathbf{p})^{\top}\mathbf{n}_{\mathbf{p}} \tag{9}$$

we can derive their Jacobian matrices as follows.

$$\nabla C_{\mathbf{p}}(\mathbf{f}) = \mathbf{d}_{\mathbf{p}}^{\top}$$

$$\mathbf{J}_{\mathbf{f}}(\mathbf{s}) = \mathbf{I} - \mathbf{n}_{\mathbf{p}}\mathbf{n}_{\mathbf{p}}^{\top}$$

Let

$$\mathbf{m}_{\mathbf{p}}^{\top} = \nabla C_{\mathbf{p}}(\mathbf{f}) \mathbf{J}_{\mathbf{f}}(\mathbf{s}) = \mathbf{d}_{\mathbf{p}}^{\top}(\mathbf{I} - \mathbf{n}_{\mathbf{p}}\mathbf{n}_{\mathbf{p}}^{\top})$$

be the shorthand notation, which is essentially a 1-by-3 vector.

Recall that we have derived for geometric component early in this note that

$$\mathbf{n}_{\mathbf{p}}^{\top}\mathbf{J}_{\mathbf{s}}(\xi) = \begin{pmatrix} \tilde{\mathbf{q}} \times \mathbf{n}_{\mathbf{p}} \\ \mathbf{n}_{\mathbf{p}} \end{pmatrix}$$

In this equation, if we replace $\mathbf{n}_{\mathbf{p}}^{\top}$ with $\mathbf{m}_{\mathbf{p}}^{\top}$, we can obtain the Jacobian in the following form

$$\begin{aligned}\nabla r_C^{(\mathbf{p},\mathbf{q})}(\mathbf{T}) &= \nabla C_{\mathbf{p}}(\mathbf{f}) \mathbf{J}_{\mathbf{f}}(\mathbf{s}) \mathbf{J}_{\mathbf{s}}(\xi) \\ &= \mathbf{m}_{\mathbf{p}}^{\top}\mathbf{J}_{\mathbf{s}}(\xi) \\ &= \begin{pmatrix} \tilde{\mathbf{q}} \times \mathbf{m}_{\mathbf{p}} \\ \mathbf{m}_{\mathbf{p}} \end{pmatrix}\end{aligned}$$

This is the final form we can use in C++ implementation. All terms used in this Jacobian are accessible in each iteration.

3 Gradient Estimation of Color Component

The estimation of the color gradient of the target cloud is a linear least-squares problem, and can be solved in closed-form by computing the pseudoinverse of a system of linear equations of the form $\mathbf{A}\mathbf{x} = \mathbf{b}$. We begin with a review of the linear least-squares problem, and then discuss how the problem in the paper fits this scheme.

3.1 Review of Linear Least-Squares Problems

The Linear Least-Squares (LLS) problem considers an overdetermined system of linear equations $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is a m-by-n matrix, $\mathbf{x}$ is a n-by-1 vector, $\mathbf{b}$ is a m-by-1 vector, and m is significantly greater than n. Provided the known linear model/relationship, we can formulate our observation data into $\mathbf{A}$ and $\mathbf{b}$ in their matrix forms, and estimate $\mathbf{x}$ by its best compromise. The corresponding objective function is of the form $L = \sum_{i=1}^{m}(\mathbf{A}_i\mathbf{x}-\mathbf{b}_i)^2$, and the best approximate solution of $\mathbf{x}$ can minimize this objective function.

Mathematically, $\mathbf{Ax} = \mathbf{b}$ has no exact solution, and the best approximate solution is $\mathbf{x} = \mathbf{A}^{+}\mathbf{b}$ where $\mathbf{A}^{+} = (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top$ is the left pseudoinverse of $\mathbf{A}$. We take the pseudoinverse in its current form (left-inverse) because $\mathbf{A}$ is (high likely) a full-column rank matrix (since m is significantly greater than n) and therefore $\mathbf{A}^\top\mathbf{A}$ is invertible. In practice, to avoid inverse operations and save computational time, we often keep the linear equation of the form $\mathbf{A}^\top\mathbf{Ax} = \mathbf{A}^\top\mathbf{b}$ and solve this equation using efficient linear solvers. For example, we often use $(\mathbf{A}^\top\mathbf{A})$.ldlt().solve($\mathbf{A}^\top\mathbf{b}$) in C++ implementation.

3.2 LLS Problem in the Paper

Recall in the paper that the objective function for computing $\mathbf{d_p}$ is

$$\begin{aligned} L(\mathbf{d_p}) &= \sum_{\mathbf{p}'\in\mathcal{N}_\mathbf{p}} \left(C_\mathbf{p}(\mathbf{f}(\mathbf{p}')-\mathbf{p}) - C(\mathbf{p}')\right)^2 \\ &\approx \sum_{\mathbf{p}'\in\mathcal{N}_\mathbf{p}} \left(C(\mathbf{p}) + \mathbf{d}_\mathbf{p}^\top(\mathbf{f}(\mathbf{p}')-\mathbf{p}) - C(\mathbf{p}')\right)^2 \end{aligned} \tag{10}$$

It can be re-organized as

$$L(\mathbf{d_p}) \approx \sum_{\mathbf{p}'\in\mathcal{N}_\mathbf{p}} \Big((\mathbf{f}(\mathbf{p}')-\mathbf{p})^\top\mathbf{d_p} - (C(\mathbf{p}')-C(\mathbf{p}))\Big)^2$$

which is of the form $L = \sum_i(\mathbf{A}_i\mathbf{x}-\mathbf{b}_i)^2$ where

$$\begin{aligned} \mathbf{A}_i &= (\mathbf{f}(\mathbf{p}')-\mathbf{p})^\top \\ \mathbf{x} &= \mathbf{d_p} \\ \mathbf{b}_i &= C(\mathbf{p}')-C(\mathbf{p}) \end{aligned}$$

All terms involved herein are accessible during the pre-processing step for the target point cloud. For completeness, recall that $\mathbf{p}' \in \mathcal{N}_\mathbf{p}$ is the neighbor point of $\mathbf{p}$ within a certain search radius, function $C(\mathbf{p})$ retrieves the intensity at point $\mathbf{p}$, function $\mathbf{f}(\mathbf{p}')$ projects the neighbor point $\mathbf{p}'$ onto the virtual image plane of point $\mathbf{p}$, and $\mathbf{d_p}$ is the color gradient at point $\mathbf{p}$.

With this, we can compute $\mathbf{A}_i$ and $\mathbf{b}_i$ for each point in the target cloud, and find the linear least-square solution of $\mathbf{x}$ by calling $(\mathbf{A}^\top\mathbf{A})$.ldlt().solve($\mathbf{A}^\top\mathbf{b}$) in the C++ implementation.

Discussion. Linear least-squares problems are convex and have a closed-form solution that is unique and optimal (in the sense of MSE). In contrast, non-linear least-squares problems generally must be solved by an iterative procedure, and the problems can be non-convex with multiple optima for the objective function. In this problem (i.e. estimate $\mathbf{d_p}$), the variable to be estimated is a linear vector in 3D space and therefore can fit into a LLS scheme. In the general transformation estimation problems, the variable is highly non-linear in 6D space (though can be linearized locally) and therefore needs to be solved by an iterative approach with Jacobian matrix involved.

4 Notations

We summarized some of the notations in the table below.

Table 1: Table of Notations

Notation	Comments
$\mathbf{C}(\mathbf{p})$	intensity at point $\mathbf{p}$, discrete function
$\mathbf{C_p}(\mathbf{u})$	$\mathbf{C_p}(\mathbf{u}) \approx \mathbf{C}(\mathbf{p}) + \mathbf{d_p}^\top \mathbf{u}$; continuous color function
$\mathbf{d_p}$	gradient of $\mathbf{C_p}(\mathbf{u})$, to be solved by least square fitting
$\mathbf{u}$	vector emanating from $\mathbf{p}$ on the tangent plane
$\mathbf{p}'$	neighbors of $\mathbf{p}$
$\mathbf{q}'$	$\mathbf{q}' = \mathbf{f}(\mathbf{s}(\mathbf{q}, \mathbf{T}))$; $\mathbf{q}$ is projected to $\mathbf{q}'$ on the tangent plane of $\mathbf{p}$
$\mathbf{s}$	$\mathbf{s}(\mathbf{h}, \mathbf{T}) = \mathbf{Th}$; apply transformation $\mathbf{T}$ onto point $\mathbf{h}$
$\mathbf{f}(\mathbf{s})$	$\mathbf{f}(\mathbf{s}) = \mathbf{s} - \mathbf{n_p}(\mathbf{s} - \mathbf{p})^\top \mathbf{n_p}$; project point $\mathbf{s}$ to the tangent plane of $\mathbf{p}$