Department of Engineering,
Trumpington Street,
University of Cambridge
CB2 1PZ

Simplified Jacobians in 6-DoF Camera Tracking

Ankur Handa
August 18, 2014

1 ICP

Aligning point clouds forms the front end of many visual odometry, 3D reconstruction and SLAM systems. Such alignment of point clouds from two or more different views allows to obtain an overall change in the translation and rotation relative to a given reference view. In this report, we focus on such problem of aligning point clouds particularly looking at the Iterative Closest Point (ICP) [2] which is one of the most popular and simplest algorithms for point cloud alignment. The algorithm alternates between finding the best possible correspondences and optimising over the 6-DoF robot pose given these correspondences. Over the years many different variations [4] have emerged that yield better and robust performance. We use the recommended point-plane variant of ICP in the following.

Denoting $\mathcal{R} = \{\mathbf{r}_i\}_{i=1}^N$ as the set of 3D points in the reference view and $\mathcal{L} = \{\mathbf{l}_j\}_{j=1}^N$, the set in the new incoming live view. ICP then seeks to obtain the $\mathsf{SE}(3)$ transformation T_{rl} (read it as live to ref transformation) that aligns the point in the live view to the reference view. The point-plane variant of ICP measures the perpendicular distance of the point from the plane and allows to slide the planar regions on top of each other. However, unlike point-point ICP which requires only the 3D positions of the points, this variant comes with an additional computational expense requiring 3D point surface normals to compute the point-plane distance. Next, we formulate the cost functions that measure the discrepency error which is optimised until a standard convergence criteria is satisfied that confirms that points are best possibly aligned. We assume in the following that

the correspondences have been obtained from some blanket search algorithm *e.g.* kD trees, projected data association and the task is to obtain the transformation given these correspondences. The cost function then reads

$$C(\mathsf{T}_{\mathsf{destination\_source}}) = \sum_{i=0}^{N}(\mathbf{n}_{\mathsf{destination}} \cdot (\mathsf{T}_{\mathsf{destination\_source}}\mathbf{l}_{\mathsf{source}} - \mathbf{r}_{\mathsf{destination}}))^2$$

where $\mathbf{n}$ is a normal vector of size 3×1 as well as $\mathbf{r}$ and $\mathbf{l}$ are of size 3×1. $\mathsf{T}_{\mathsf{destination\_source}}$ is the transformation required to map the points from source to destination view (Note that the transformation $\mathsf{T}_{\mathsf{x\_y}}$ is read the transformation that maps points from y to x.) In our case this would be T_{rl} which transforms from live to reference view. Rewriting this in our notation, the cost function is

$$C(\mathsf{T}_{\mathsf{rl}}) = \sum_{i=0}^{N}(\mathbf{n}_{\mathbf{r}_i}^T(\mathsf{T}_{\mathsf{rl}}\mathbf{l}_i - \mathbf{r}_i))^2$$

We seek to obtain the transformation T_{rl} that overlays points on top of each other with minimum cost. Using the well-known trick of linearising the transformation around a previous estimate $\hat{\mathsf{T}}_{\mathsf{rl}}$, the cost function simplifies to the following

$$C(\boldsymbol{\delta}) = \sum_{i=0}^{N}(\mathbf{n}_{\mathbf{r}_i}^T(\exp(\boldsymbol{\delta})\hat{\mathsf{T}}_{\mathsf{rl}}\mathbf{l}_i - \mathbf{r}_i))^2$$

The residual at each point is denoted by $e_i(\boldsymbol{\delta})$.

$$e_i(\boldsymbol{\delta}) = \mathbf{n}_{\mathbf{r}_i}^T(\exp(\boldsymbol{\delta})\hat{\mathsf{T}}_{\mathsf{rl}}\mathbf{l}_i - \mathbf{r}_i)$$

The first order taylor approximation yields

$$\begin{aligned} e_i(\boldsymbol{\delta}) &= e_i(\mathbf{0}) + \mathsf{J}_i\boldsymbol{\delta} \\ C(\boldsymbol{\delta}) &= \sum_{i=0}^{N}(e_i(\mathbf{0}) + \mathsf{J}_i\delta)^2 \end{aligned}$$

The Jacobians are defined as

$$\begin{aligned} \mathsf{J}_i &= \frac{\partial e_i(\boldsymbol{\delta})}{\partial \boldsymbol{\delta}} \\ \mathsf{J}_i &= \frac{\partial e_i(\boldsymbol{\delta})}{\partial \boldsymbol{\delta}} = \mathbf{n}_{\mathbf{r}_i} \cdot \frac{\partial \exp(\delta)\hat{\mathbf{r}}_i}{\partial \boldsymbol{\delta}} \end{aligned}$$

where

$$\hat{\mathbf{r}}_i = \hat{\mathsf{T}}_{\mathsf{rl}}\mathbf{l}_i$$

$$\Rightarrow \mathsf{J}_i \quad = \quad \begin{pmatrix} \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_0 \hat{\mathbf{r}}_i & \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_1 \hat{\mathbf{r}}_i & \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_2 \hat{\mathbf{r}}_i & \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_3 \hat{\mathbf{r}}_i & \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_4 \hat{\mathbf{r}}_i & \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_5 \hat{\mathbf{r}}_i \end{pmatrix}$$

where the $\mathbf{G}_i$ are the $\mathsf{SE3}$ generators defined as

$$\mathbf{G}_0 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \mathbf{G}_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \mathbf{G}_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{G}_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \mathbf{G}_5 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \mathbf{G}_6 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Even though the Generators are 4×4, the last rows can be removed to allow the dot products with $3{\times}1$ or $1{\times}3$ vectors. Simplifying the first three terms in the Jacobians we arrive at very simple expressions

$$\begin{aligned} \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_0 \hat{\mathbf{r}}_i &= \mathbf{n}_{\mathbf{r}_i} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = (\mathbf{n}_{\mathbf{r}_i})_x \\ \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_1 \hat{\mathbf{r}}_i &= \mathbf{n}_{\mathbf{r}_i} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = (\mathbf{n}_{\mathbf{r}_i})_y \\ \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_2 \hat{\mathbf{r}}_i &= \mathbf{n}_{\mathbf{r}_i} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = (\mathbf{n}_{\mathbf{r}_i})_z \end{aligned}$$

Therefore, the first terms simplify to the normal vector that is computed at the reference point avoiding the need of any computation since it is computed already.

$$\Rightarrow \begin{pmatrix} \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_0 \hat{\mathbf{r}}_i & \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_1 \hat{\mathbf{r}}_i & \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_2 \hat{\mathbf{r}}_i \end{pmatrix} \quad = \quad \mathbf{n}_{\mathbf{r}_i}^T$$

The rest of three terms give

$$\begin{aligned} \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_3 \hat{\mathbf{r}}_i &= \mathbf{n}_{\mathbf{r}_i} \cdot \begin{pmatrix} 0 \\ -(\hat{\mathbf{r}}_i)_z \\ (\hat{\mathbf{r}}_i)_y \end{pmatrix} \\ \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_4 \hat{\mathbf{r}}_i &= \mathbf{n}_{\mathbf{r}_i} \cdot \begin{pmatrix} (\hat{\mathbf{r}}_i)_z \\ 0 \\ -(\hat{\mathbf{r}}_i)_x \end{pmatrix} \\ \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_5 \hat{\mathbf{r}}_i &= \mathbf{n}_{\mathbf{r}_i} \cdot \begin{pmatrix} -(\hat{\mathbf{r}}_i)_y \\ (\hat{\mathbf{r}}_i)_x \\ 0 \end{pmatrix} \end{aligned}$$

It is interesting to see that this is nothing but the cross product of the projected point in the reference frame and the normal at it's correspondence in the reference frame.

$$\begin{pmatrix} \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_3\hat{\mathbf{r}}_i & \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_4\hat{\mathbf{r}}_i & \mathbf{n}_{\mathbf{r}_i} \cdot \mathbf{G}_5\hat{\mathbf{r}}_i \end{pmatrix} = \mathbf{n}_{\mathbf{r}_i}^T \begin{pmatrix} 0 & (\hat{\mathbf{r}}_i)_z & -(\hat{\mathbf{r}}_i)_y \\ -(\hat{\mathbf{r}}_i)_z & 0 & (\hat{\mathbf{r}}_i)_x \\ (\hat{\mathbf{r}}_i)_y & -(\hat{\mathbf{r}}_i)_x & 0 \end{pmatrix} = (\hat{\mathbf{r}}_i \times \mathbf{n}_{\mathbf{r}_i})^T$$

Therefore, the 1×6 Jacobians have a very simplified form involving **cross product**

$$\Rightarrow \mathsf{J}_i = \begin{pmatrix} \mathbf{n}_{\mathbf{r}_i}^T & (\hat{\mathbf{r}}_i \times \mathbf{n}_{\mathbf{r}_i})^T \end{pmatrix}$$

Remember that the transposes here are only to flatten the row vectors to their respective columnar counterparts. The update then comes down to

$$\begin{aligned} \boldsymbol{\delta} &= \big(\sum_{i=0}^{N} \mathsf{J}_i^T \mathsf{J}_i\big)^{-1} \big(\sum_{i=0}^{N} \mathsf{J}_i^T e_i(\mathbf{0})\big) \\ \Rightarrow \hat{\mathsf{T}}_{\mathsf{rl}}^{k+1} &= \exp(\boldsymbol{\delta}) \hat{\mathsf{T}}_{\mathsf{rl}}^{k} \end{aligned}$$

In the following subsections, we take a look at other different ways to align point clouds that have emerged recently.

1.1 Direct Visual Odometry(DVO)

If ICP aligns two point clouds from different views, DVO [5] assumes that there are also two images that are achored to the respective point clouds. Under the assumption that the 3D structure that is being viewed is the same, one can project the colour of one image onto the other using the depth-map and the relative camera pose and check for any colour inconsistency as a measure to quantify the error in the transformation. The RGBD cost function measure exactly that and optimises over the transformation until the two images have as small as possible colour consistency error.

$$C(\mathsf{T}_{\mathsf{rl}}) = \sum_{i=0}^{N} \big(\mathcal{I}_r(\pi(\mathsf{K}\mathsf{T}_{\mathsf{rl}}\dot{\mathbf{p}})) - \mathcal{I}_l(\mathbf{x}_i)\big)^2$$

$$\begin{aligned} C(\boldsymbol{\delta}) &= \sum_{i=0}^{N} \big(\mathcal{I}_r(\pi(\mathsf{K}\exp(\boldsymbol{\delta})\mathsf{T}_{\mathsf{rl}}\dot{\mathbf{p}})) - \mathcal{I}_l(\mathbf{x}_i)\big)^2 \\ e_i(\boldsymbol{\delta}) &= \mathcal{I}_r(\pi(\mathsf{K}\exp(\boldsymbol{\delta})\mathsf{T}_{\mathsf{rl}}\dot{\mathbf{p}})) - \mathcal{I}_l(\mathbf{x}_i) \\ e_i(\boldsymbol{\delta}) &= e_i(\mathbf{0}) + \mathsf{J}_i\boldsymbol{\delta} \\ C(\boldsymbol{\delta}) &= \sum_{i=0}^{N} (e_i(\mathbf{0}) + \mathsf{J}_i\boldsymbol{\delta})^2 \end{aligned}$$

$$
\begin{aligned}
\mathsf{J}_i &= \frac{\partial e(\boldsymbol{\delta})}{\partial \boldsymbol{\delta}} = \frac{\partial \mathcal{I}_r(\pi(\mathsf{K}\mathsf{T}_{\mathsf{rl}}\dot{\mathbf{p}}))}{\partial \boldsymbol{\delta}} \\
\mathsf{J}_i &= \underbrace{\frac{\partial \mathcal{I}_r(\pi(\mathsf{K}\mathsf{T}_{\mathsf{rl}}\dot{\mathbf{p}}_i))}{\partial \pi(\mathsf{K}\mathsf{T}_{\mathsf{rl}}\dot{\mathbf{p}}_i)} \frac{\partial \pi(\mathsf{K}\mathsf{T}_{\mathsf{rl}}\dot{\mathbf{p}}_i)}{\partial \mathsf{K}\mathsf{T}_{\mathsf{rl}}\dot{\mathbf{p}}_i}}_{\mathsf{gradpi}_i} \frac{\partial \mathsf{K}\mathsf{T}_{\mathsf{rl}}\dot{\mathbf{p}}_i}{\partial \boldsymbol{\delta}} \\
\mathsf{J}_i &= \mathbf{gradpi}_i^T \ \mathsf{K} \ \frac{\partial \mathsf{T}_{\mathsf{rl}}\dot{\mathbf{p}}_i}{\partial \boldsymbol{\delta}} \\
\mathsf{J}_i &= \mathbf{gradpi}_i^T \ \mathsf{K} \ \frac{\partial \exp(\boldsymbol{\delta})\hat{\mathsf{T}}_{\mathsf{rl}}\dot{\mathbf{p}}_i}{\partial \boldsymbol{\delta}}
\end{aligned}
$$

gradpi is 3×1 and K of size 3×3 therefore $\mathbf{gradpi}^T\mathsf{K}$ is of size 1×3. Rewriting $\mathbf{gradpi}^T\mathsf{K}$ as **gpK** as a 1×3 vector. We arrive at a similar expression where Jacobian is of the form

$$
\mathsf{J}_i = \mathbf{gpK}_i \ \frac{\partial \exp(\boldsymbol{\delta})\hat{\mathsf{T}}_{\mathsf{rl}}\dot{\mathbf{p}}_i}{\partial \boldsymbol{\delta}}
$$

The Jacobian further simplies to

$$
\mathsf{J}_i = \begin{pmatrix} \mathbf{gpK}_i & (\hat{\mathbf{p}}_i \times \mathbf{gpK}_i)^T \end{pmatrix}
$$

where

$$
\hat{\mathbf{p}}_i = \hat{\mathsf{T}}_{\mathsf{rl}}\dot{\mathbf{p}}_i
$$

and

$$
\hat{\mathbf{p}}_i \times \mathbf{gpK}_i
$$

is the **cross product** involving the two terms.

1.2 ICP+DVO

This method is amalgamation of point-plane ICP and DVO [6]. It not only measures the geometric error in the point clouds but also the colour consistency error in the images. However, both these cost functions have different units and dimensions and as a result a weight is added to weigh one over the other. This overall cost function has an added advantage of being able to still give a sensible transformation if either one of them fails in a given scenario. For instance, in planar regions it is difficult for point-plane ICP to obtain a unique transformation to register two point clouds; therefore, the colour based term still operates and tries to pull the data from one frame to the other.

$$
C(\mathsf{T}_{\mathsf{rl}}) = \sum_{i=0}^{N} (\mathbf{n}_{\mathbf{r}_i}^T(\mathsf{T}_{\mathsf{rl}}\mathbf{l}_i - \mathbf{r}_i))^2 + w \sum_{i=0}^{N} \left(\mathcal{I}_r(\pi(\mathsf{K}\mathsf{T}_{\mathsf{rl}}\dot{\mathbf{p}})) - \mathcal{I}_l(\mathbf{x}_i)\right)^2
$$

$$
\mathsf{J}_i = \begin{pmatrix} \mathbf{n}_{\mathbf{r}_i}^T & (\hat{\mathbf{r}}_i \times \mathbf{n}_{\mathbf{r}_i})^T \\ \sqrt{w}\mathbf{gpK}_i & \sqrt{w}(\hat{\mathbf{p}}_i \times \mathbf{gpK}_i)^T \end{pmatrix}
$$

1.3 SDFTracking

The SDF tracking [1] based method assumes that the scene is available in the classic discrete voxel grid representation [3]. As a result, one can directly throw the incoming points into the voxel grid and read off the signed distance function (SDF) values sidestepping the correspondence part of the optimisation in the previous methods. The cost function is optimised over the transformtion until the read SDF values are as close to zero as possible. The function ψ returns the SDF value at any given 3D location in the voxel grid. It is important to remember that for non-integer 3D positions, the SDF values are interpolated with favourite interpolation scheme *e.g.* bilinear interpolation *etc.*

$$\begin{aligned} C(\mathsf{T}_{\mathsf{rl}}) &= \sum_{i=0}^{N} (\psi(\mathsf{T}_{\mathsf{wl}}\mathbf{l}_i))^2 \\ &= \sum_{i=0}^{N} (\psi(\exp(\boldsymbol{\delta})\hat{\mathsf{T}}_{\mathsf{wl}}\mathbf{l}_i))^2 \\ &= \sum_{i=0}^{N} (\psi(\exp(\mathbf{0})\hat{\mathsf{T}}_{\mathsf{wl}}\mathbf{l}_i) + \mathsf{J}_i^T\boldsymbol{\delta})^2 \end{aligned}$$

$$\begin{aligned} \mathsf{J}_i &= \frac{\partial\psi(\exp(\boldsymbol{\delta})\hat{\mathsf{T}}_{\mathsf{wl}}\mathbf{l}_i)}{\partial\boldsymbol{\delta}} \\ &= \underbrace{\frac{\partial\psi(\hat{\mathsf{T}}_{\mathsf{wl}}\mathbf{l}_i)}{\partial(\hat{\mathsf{T}}_{\mathsf{wl}}\mathbf{l}_i)}}_{\mathsf{s}_i} \frac{\partial(\exp\boldsymbol{\delta})\hat{\mathsf{T}}_{\mathsf{wl}}\mathbf{l}_i}{\partial\boldsymbol{\delta}} \end{aligned}$$

Writing

$$\hat{\mathbf{l}}_{\mathbf{i}} = \hat{\mathsf{T}}_{\mathsf{wl}}\mathbf{l}_i$$

and s_i is the derivative vector of size 1×3, the Jacobian simplifies to

$$\begin{aligned} \mathsf{J}_i &= \mathsf{s}_i \frac{\partial\exp(\boldsymbol{\delta})\hat{\mathbf{l}}_{\mathbf{i}}}{\partial\boldsymbol{\delta}} \\ \Rightarrow \mathsf{J}_i &= \begin{pmatrix} \mathsf{s}_i & (\hat{\mathbf{l}}_{\mathbf{i}} \times \mathsf{s}_i)^T \end{pmatrix} \end{aligned}$$

References

[1] Daniel R Canelhas, Todor Stoyanov, and Achim J Lilienthal. SDF tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, pages 3671–3676, 2013.

[2] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. *Image and Vision Computing (IVC)*, 10(3):145–155, 1992.

[3] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proceedings of SIGGRAPH*, 1996.

[4] S. Rusinkiewicz and M. Levoy. Efficient Variants of the ICP Algorithm. In *Proceedings of the IEEE International Workshop on 3D Digital Imaging and Modeling (3DIM)*, 2001.

[5] F. Steinbrucker, J. Sturm, and D. Cremers. Real-Time Visual Odometry from Dense RGB-D Images. In *Workshop on Live Dense Reconstruction from Moving Cameras at ICCV*, 2011.

[6] T. Tykkala, C. Audras, and A. I. Comport. Direct Iterative Closest Point for real-time visual odometry. In *ICCV Workshops*, 2011.