The Geometry of Spinning Lidar Sensors

Per-Erik Forssén

November 26, 2021

1 Introduction

This document describes the geometry of spinning lidar sensors such as Ouster OS-1 64 and Velodyne HDL-64E. In particular we will look at motion based scan correction (the equivalent of a rolling shutter distortion correction on a camera).

2 Scan to point-cloud mapping

Sensors such as Ouster OS-1 64 [5] and Velodyne HDL-64E [9] are spinning lidars. At each point in time, 64 range values[1] are received from different angles. By spinning the sensor a full circle, one so called *point cloud* can be obtained, see figure 1. By spinning at different rates, temporal resolution can be traded for angular resolution. E.g. in the Kitti dataset[2], the sensor spins at 10Hz, which results in 2000 directions [1]. By switching to 5Hz, 4000 directions would be obtained instead in each point cloud.

2.1 Range images

The lidar outputs data in the form of a range image (which contains distances computed from time of flight), see figure 1, top, for an example of a range image. The lidar also outputs a reflectance image (which contains received signal strengths), but this will not be discussed here. The points in a range image can be unpacked to raw point clouds, using the calibration angles (ϕ, θ) (pitch and yaw) provided with the dataset, by converting each angle pair to a unit direction vector $\hat{\mathbf{x}}(\phi, \theta)$ and scaling this with the corresponding range value. By denoting the pixels in the range image by r_{kl}, with $k \in [0, 63]$ for 64 channels, and $l \in [0, 869]$ for 870 directions, we get the following expression:

$$\mathbf{X}_{\text{raw}}(t) = r_{kl}\hat{\mathbf{x}}_k \,, \text{where } \hat{\mathbf{x}}_k = \begin{bmatrix} \cos\theta_k \cos\phi_k \\ -\sin\theta_k \cos\phi_k \\ \sin\phi_k \end{bmatrix} . \tag{1}$$

[1] There are also 16, 32, and 128 channel lidars, but 64 is more common.

[2] At `http://www.cvlibs.net/datasets/kitti/raw_data.php` the lidar scans can be found.

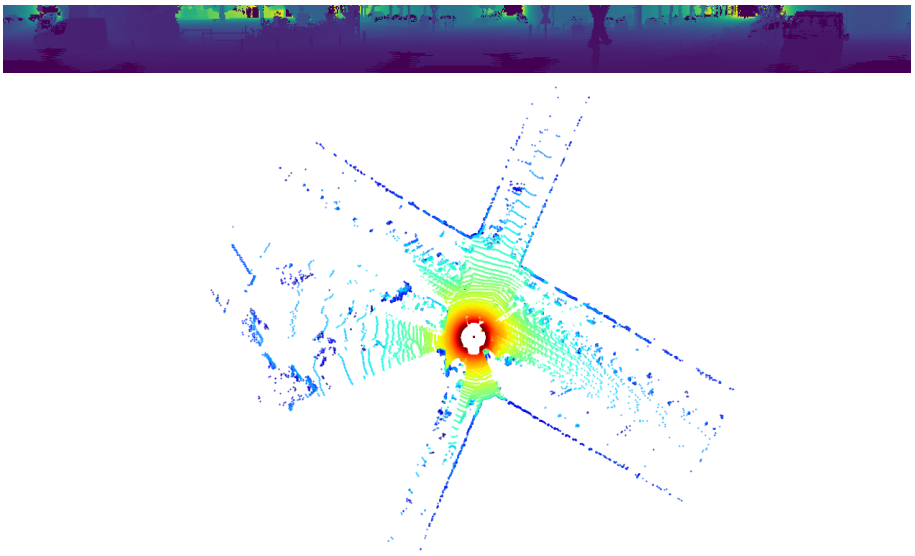

Figure 1: Top: Range image with 64 lines and 870 directions (from the VelodyneSLAM dataset). Bottom: The unpacked point cloud in the x-y plane. Colors indicate different channels.

The scan time in (1) is $t = t_0 + l\Delta t$, for some start time t_0 and time delta Δt. E.g. for 870 directions and a 10 Hz spin rate, we have $\Delta t = 0.1/870 \approx 0.115$ ms.

The points $\mathbf{X}_{\text{raw}}(t)$ in the raw scan lines, can now be mapped to the car coordinate system using the expression:

$$\begin{bmatrix} \mathbf{X}_{\text{car}}(t) \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}(t) & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\text{raw}}(t) \\ 1 \end{bmatrix} . \tag{2}$$

The result of (1) and (2) is shown in figure 1. (Data from the VelodyneSLAM dataset[3]).

We can express the rotation in (2) as a matrix exponentiation $\mathbf{R}(t) = e^{\theta(t)[\hat{\mathbf{n}}]_\times}$, where $\hat{\mathbf{n}}$ is the axis of rotation (here the up vector). Assuming a constant spin rate, the rotation angle becomes $\theta(t) = 2\pi(t - t_0)/T$. Here t_0 is the time at the start of the scan, and $T = 0.1$ is the revolution period (assuming 10Hz). The $[\cdot]_\times$ is the cross product operator. It generates the unique skew-symmetric matrix that satisfies $[\mathbf{a}]_\times \mathbf{b} = \mathbf{a} \times \mathbf{b}$.

Using Rodrigues' rotation formula [4], the rotation can be expanded into:

$$\mathbf{R}(t) = e^{\theta(t)[\hat{\mathbf{n}}]_\times} = \mathbf{I} + \sin\theta(t)\,[\hat{\mathbf{n}}]_\times + (1 - \cos\theta(t))\,[\hat{\mathbf{n}}]_\times^2 \tag{3}$$

[3]See `https://www.mrt.kit.edu/z/publ/download/velodyneslam/dataset.html`.

Figure 2: Scan with missing data, from the VelodyneSLAM dataset. Each of the 870 columns in the original image corresponds to one set of 64 laser readings, but here they have been de-staggered to get a nice image, by applying cyclic shifts to the rows. As the data dropout starts at a particular time, the sawtooth pattern is revealed.

In the special case of $\hat{\mathbf{n}} = [0, 0, 1]^T$ we get:

$$\mathbf{R}(t) = \begin{bmatrix} \cos\theta(t) & -\sin\theta(t) & 0 \\ \sin\theta(t) & \cos\theta(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

2.2 De-staggering

To reduce interference between beams, both the OS-1 and HDL64E have their lidar beam angles arranged in a sawtooth pattern. This pattern is often undone by applying a cyclic shift to the rows of the range image. This operation is called *de-staggering*. Indeed, this has been done on the image in figure 1.

If we look at a scan with data dropouts, the sawtooth pattern is revealed, see figure 2. The reason for this is that the dropout starts at a particular time. If motion compensation is to be applied, it is important that de-staggering is *not applied*, as the image columns are then no longer recorded at the same time.

In the Ouster source code[4] de-staggering is applied at the host, according to a calibration table. We can thus simply omit de-staggering when we want to do motion compensation.

2.3 Motion compensation

Assuming that we have point clouds in the car coordinate system, see (2), we are in a position to apply motion compensation, aka. scan correction. This means that we should make use of the continuous-time motion of the car to convert car coordinates into world coordinates, see figure 3.

The motion trajectory of the car can be expressed as one signal $\mathbf{R}(t) \in \mathrm{SO}(3)$ for the orientation, and one $\mathbf{p}(t) \in \mathbb{R}^3$ for the position [7]. For a motion expressed in the world coordinate system, motion compensation should be applied as follows:

$$\begin{bmatrix} \mathbf{X}_{\text{world}}(t) \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}(t) & \mathbf{p}(t) \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\text{car}}(t) \\ 1 \end{bmatrix}. \tag{5}$$

A preliminary registration of a sequence of $\mathbf{X}_{\text{car}}$ point sets, e.g. using [2, 3], would result in a discrete sequence of poses $\{\mathbf{R}_n, \mathbf{p}_n\}_{n=1}^N$. Such a pose sequence can then be interpolated using a split spline in $(\mathbb{R}^3, \mathrm{SO}(3))$ see [7], to

[4] See `https://github.com/ouster-lidar/ouster_example`

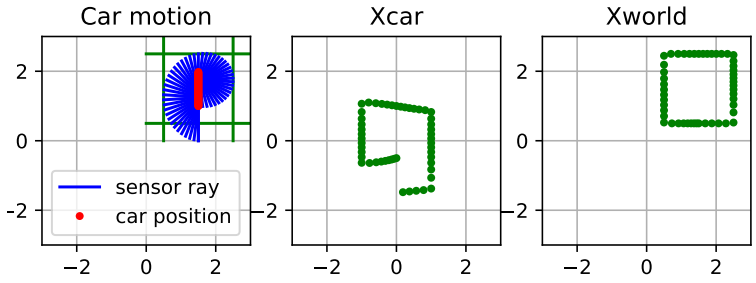

Figure 3: Illustration of motion compensation in 2D. Left: sensor positions(RED), sensor rays(BLUE) and walls (GREEN). Centre: range values unpacked to the car coordinate system. Right: Motion compensation results in a corrected point set.

obtain the continuous signals needed in (5). Such a motion estimate is however typically much smoother than what actually took place. For more detailed motion estimates, it is thus common to also use an IMU.

2.4 Motion estimate from the IMU

A common way to obtain a motion estimate is to use the motion sensed by an *inertial measurement unit* (IMU). Lidar sensors such as the Ouster OS-1 have a built in IMU, and many cars are also equipped with intertial sensors. As an IMU senses linear acceleration and angular velocity, we first need to integrate the signal to obtain a motion estimate. Assuming we have signals $\boldsymbol{\omega}(t)$ and $\mathbf{a}(t)$ from the IMU, corresponding to angular velocities and linear accelerations, respectively. These can be integrated into orientations and displacements:

$$\boldsymbol{\Theta}(t) = \oint_{t_0}^{t} \exp((\boldsymbol{\omega}(t) - \mathbf{b}_{\text{gyro}})dt)\,, \text{ and } \mathbf{p}(t) = \int_{t_0}^{t}\int_{t_0}^{t} (\mathbf{a}(t) - \mathbf{b}_{\text{acc}})d^2t\,. \quad (6)$$

Here $\mathbf{b}_{\text{gyro}}$ and $\mathbf{b}_{\text{acc}}$ are bias vectors, that are in general unknown. These are slowly drifting over time, but for practical purposes they can be assumed to be constant. Alternatively, if a motion estimate from a preliminary scan registration is available (see section 2.3) this could be used to estimate the bias.

The integrals (6) need to be evaluated numerically, e.g. using a forward Euler scheme. Note also that the orientation integration is not linear, and should be computed while respecting the properties of the rotation group. This implies that small delta rotations should be generated, and multiplied together in sequence, e.g. using unit quaternions [8]. Given the integrated signals we can now do motion compensation according to:

$$\begin{bmatrix}\mathbf{X}_{\text{world}}(t)\\ 1\end{bmatrix} = \mathbf{T}_{\text{c2i}}^{-1}\begin{bmatrix}\boldsymbol{\Theta}(t) & \mathbf{p}(t)\\ \mathbf{0}^T & 1\end{bmatrix}\mathbf{T}_{\text{c2i}}\begin{bmatrix}\mathbf{X}_{\text{car}}(t)\\ 1\end{bmatrix}. \quad (7)$$

Compared to (5) we here have the additional transformation $\mathbf{T}_{\text{c2i}}$. This is a mapping from the CCS to the IMU Coordinate System (ICS). Note that it has to be applied twice, first to move from car coordinates to IMU coordinates, and then after the motion estimate has been applied, we need to go back to the original frame again (if we are to have world coordinates of the car, and not the IMU) [6].

References

[1] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research (IJRR)*, 32(11):1231–1237, September 2013. https://doi.org/10.1177/0278364913491297.

[2] Felix Järemo Lawin, Martin Danelljan, Fahad Khan, Per-Erik Forssén, and Michael Felsberg. Density adaptive point set registration. In *IEEE Conference on Computer Vision and Pattern Recognition*, Salt Lake City, Utah, USA, June 2018. Computer Vision Foundation.

[3] Felix Järemo Lawin and Per-Erik Forssén. Registration loss learning for deep probabilistic point set registration. In *International Virtual Conference on 3D Vision (3DV 2020)*, November 2020.

[4] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[5] Ouster. Os-1-64/16 high resolution imaging lidar software user guide. `https://levelfivesupplies.com/wp-content/uploads/2019/03/OS-1-User-Guide-Software.pdf`.

[6] Hannes Ovrén and Per-Erik Forssén. Gyroscope-based video stabilisation with auto-calibration. In *IEEE International Conference on Robotics and Automation ICRA'15*, Seattle, USA, May 2015. IEEE. VR Project: Learnable Camera Motion Models, 2014-5928, SSF Project: The Virtual Photo Set, IIS11-0081.

[7] Hannes Ovrén and Per-Erik Forssén. Trajectory representation and landmark projection for continuous-time structure from motion. *International Journal of Robotics Research*, 38(6):686–701, May 2019. Accepted 2019-01-23.

[8] Erik Ringaby and Per-Erik Forssén. A virtual tripod for hand-held video stacking on smartphones. In *IEEE International Conference on Computational Photography (ICCP)*. IEEE, IEEE, May 2014.

[9] Velodyne. Velodyne hdl-64e user manual. `https://www.velodynelidar.com/lidar/products/manual/HDL-64E%20Manual.pdf`.