ITTIK

Olaf Booij

February 5, 2021

The KITTI dataset is a standard for evaluating R&D in autonomous driving and mapping. An important part of the used sensor setup is a surround LiDAR sensor which captures ranges and reflectivity of the structures and objects around the vehicle. This LiDAR data is not presented as raw sensor readings, which are hard to interpret. Rather, the data is preprocessed into easy to comprehend single scan 3D point clouds measured in a single 360 degrees rotation. Generic point cloud software can be used to visualize and process such data.

A drawback of the point clouds is that it is difficult to get the precise time for each of the 3D point measurements. In addition, it is difficult to compute the exact origin of the cast LiDAR rays. A more general concern is that the point clouds do not reflect the dense and structured form of the LiDAR measurement.

ITTIK tries to get back the raw LiDAR sensor data by undoing the steps performed during the KITTI preprocessing. It does so using the KITTI point clouds, as well as the available calibration data.

1 Velodyne data

The LiDAR sensor used for the KITTI dataset is a Velodyne HDL-64E. This device fires its 64 individual lasers with a frequency of 20 KHz, while the device is rotating at a rate of 10 Hz, resulting in approximately 130,000 measurements per scan and 1.3 million measurements per second. The 64 lasers do not fire at the same time. Instead, at each timestep, two of the lasers fire in a fixed sequence. Because the device keeps on rotating, each pair of points has a unique measurement time as well as a unique device rotational position.

For each firing laser, the distance and reflectivity of the scene point is measured. In addition, for each lower and upper block of 32 laser fires the rotational position is measured. For each packet of 6 lower and upper blocks, a timestamp is sent, which can be used to synchronize the data with measurements from other sensors.

The rays of each laser have a different vertical and horizontal direction. In addition, the lasers are distributed over 4 locations on the device resulting in a slightly different offset of the ray origin. The exact value of these directions and offsets are given in the calibration stored on the device (more on this in the next section). In Figure 1 the different directions of the laser rays are shown. As can be seen the vertical view-angle is approximately 28 degrees and the horizontal view-angle 19 degrees.

It is important to understand the implications of this horizontal view angle. Points from the same scan that are close to each other in the scene, have a different measurement time. Take for example laser #41 and laser #46 (see Figure 1). Vertically, they are “neighbors”, but horizontally they are 19 degrees apart.

Thus, it is not easy to interpret the measurements of a single fire of all lasers. It is like a depth camera where only 64 pixels are working spread over the image plane. A point cloud representations is much more useful.

Still, for this point cloud we should consider that the device for the example lasers has to rotate 19 degrees, before they measure the same structure. Taking the rotation speed into account, two very close points in the scene will have been measured with a time difference of $\frac{19}{360} 100\,\text{ms} = 5\,\text{ms}$.

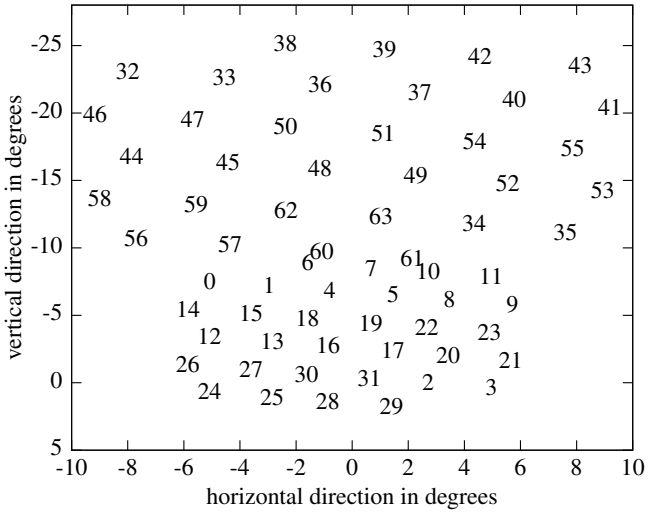


Figure 1: Direction of each laser, plotted with there index, as indicated in the calibration file in degrees.

This time difference has to be taken into account of course when processing the data. KITTI contains, in addition to the "raw" point clouds, motion corrected data, in which the vehicle motion is compensated for, for each laser. For moving objects, however, this is of course more difficult.

This project is mainly meant to counter another disadvantage of the point cloud representation. The spread of the lasers, results in a non-uniform point cloud, hard to feed into standard image processing tools such as convolutional neural networks. This is the main reason to try to get the original Velodyne data back.

An extra note on the amount of measurements and the timing of the lasers. In the manual, it is stated that the device measures points at a rate of around 1.3 million per second. However, when looking at the timing of the device each pair of lasers is fired after $.73\mu s$. If you would assume a monotonic firing behavior of the device this would result in $\frac{2}{.73 \cdot 10^{-6}} \approx 2.74$ million points per second. Later (Figure 4) we will indeed see that the device has a bit peculiar firing behavior.

2 Preprocessing

In order to easily interpret the Velodyne data, the raw byte stream is preprocessed using a fixed process. This process has two parts. The first part, computes for each laser fire from the laser id, the measured distance and measured rotational position, the 3D point location. The second part, computes for the raw intensity measurement, the surface reflectivity of the measured point. We will (for now) only focus on the first part.

The preprocessing uses some parameters. Some of these are fixed for all devices of the used LiDAR type, such as the location of the laser on the device. There are, however, also device specific parameters, such as the precise direction of the lasers, which are determined in the factory using a specific calibration procedure and stored on the device. These values are also transmitted by the device to be stored and used by the preprocessing procedure.

For the KITTI dataset these calibration values are provided on the website in the standard xml form: `velodynecalib_S2_factory_flatness_intensity.xml`. The values in this file are used in ITTIK, and for visualizations in this document such as in Figure 1.

We first thought this was the preprocessing algorithm described in the User's Manual and Programming Guide. However, after implementing and trying it out, this seemed not to be correct. For later reference, and possible use for newer datasets, it is given in Appendix 5. We found out by turning of some parts in that quite complicated and hard to reverse algorithm, that the actual algorithm used is much simpler. In addition, it does not use some of the calibration parameters, which is actually a bit strange.

In the following, we describe the main part of the preprocessing algorithm. Some trivial parts on rescaling the raw byte stream data is left out.

The function to compute the 3D point $\mathbf{p}$, given a distance measurement d, the rotational position θ and the laser id l is:

$$\mathbf{p} = f(d, \theta, l) = \begin{pmatrix} (d + \Delta d_l) * \cos\phi_l * \sin(\theta - \Delta\theta_l) - h_l \cos(\theta - \Delta\theta_l) \\ (d + \Delta d_l) * \cos\phi_l * \cos(\theta - \Delta\theta_l) + h_l \sin(\theta - \Delta\theta_l) \\ (d + \Delta d_l) * \sin\phi_l + v_l \end{pmatrix}, \quad (1)$$

in which the variables with subscript l are the calibration parameters specific to laser l relative to the device. The values θ_l and ϕ_l are the horizontal and vertical direction, named vertCorrection and rotCorrection. The values h_l and b_l are the horizontal and vertical position, named horizOffsetCorrection and vertOffsetCorrection, and Δd_l is the distance correction, named distCorrection.

Apart from some rescaling that is left out, we also emitted some offset from the process called "pos" in the User's Manual, which is not explained further. This needs further investigation.

3 Reversing the preprocessing

The aim is to reverse Equation (7), getting back the distance measurement d, the rotational position of the device θ and the laser id l given a point p.

The laser id can perhaps be computed by determining the vertical direction of the point and matching it with the fixed ϕ_l's of each laser. However, it is easier to use the order of the points in the KITTI point clouds, which group the point per laser starting with the most upward pointing laser. This approach is straightforward and described nicely in Triess et al. (2020). The order of the vertical angles ϕ_l described in calibration can now be used to determine the laser id for each point.

Getting the other parameters is quite straightforward. We give the equations here.

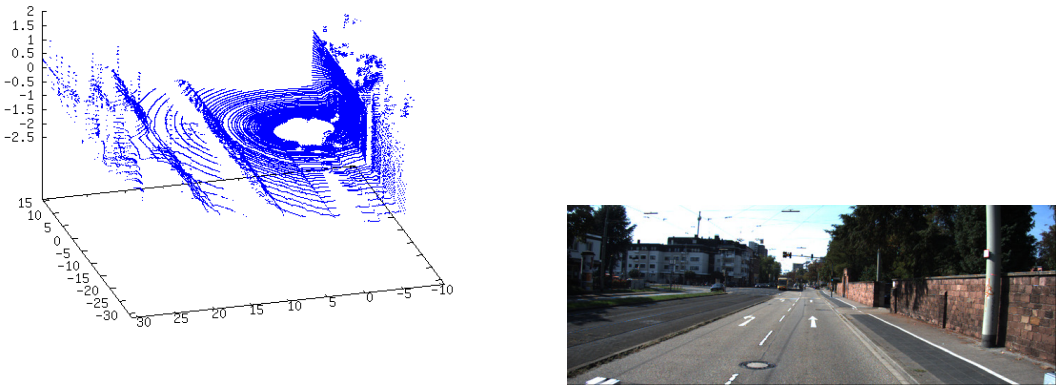
$$\theta = \operatorname{atan2}(\mathbf{p}_x, \mathbf{p}_y) + \operatorname{atan2}\left(h_l, \sqrt{\mathbf{p}_x^2 + \mathbf{p}_y^2}\right) + \Delta\theta_l, \quad (2)$$

$$d = \frac{\sqrt{\mathbf{p}_x^2 + \mathbf{p}_y^2 - h_l^2}}{\cos(\phi_l)} - \Delta d. \quad (3)$$

4 Results

To assess if the reverse method described in the previous section works correctly, we apply it on some of the KITTI data.

We first apply the algorithm on an arbitrary KITTI scan (see Figure 2 to get the raw measurements and then apply the standard preprocessing algorithm, as described in Section 2 to reconstruct back the 3D points. In theory these reconstructed points should be the same as in the original scan. In practice, of course, there is the quantification error, while the KITTI point clouds are stored in millimeter precision.



(a) LiDAR point cloud

(b) Image still

Figure 2: Example scan used in the results. This is scan number 71 and an image still from the raw dataset `2011_09_26_drive_0002`.

The mean distance between the original 3D point and the reconstructed point for the scan is 2.88 mm. In Figure 3, this is split over the different lasers sorted by vertical angle. It can be seen that lasers pointing down have an average error up to 11 mm. This is much larger than expected given the millimeter precision of the original points. This could also be related with the distance of the points in the scene (the distance is on average smaller for lasers pointing down).

The angle of the point in the horizontal plane of the sensor is the same for the original and the reconstructed point. It is interesting to see that there is no error here. It seems thus that the main error is in the vertical direction. The mean error of the distance of the point to the sensor is .77 mm.

Let us now look at the computed device rotational position per point. Figure 4(a) shows them for some of the points of the used scan. As can be seen the computed rotation almost aligns for each of the lasers. There is only a small jitter visible up to .3 mrad. A large part, but not all, can be explained by the discretization error, due to the mm accuracy of the point clouds.In

5 Conclusion

By reversing the Velodyne preprocessing, we can obtain the original measured depth and rotational position. However, there still seems to be something wrong. The cause is still unknown. It might be an error in the method, an error in the calibration file, the emitted offset “pos”, or a combination of these. Or, of course, it could be just a silly bug...

The used preprocessing is different from what the User’s Manual and Programming Guide, with version number `63HDL64E S2 Rev F DEC11`, describes. Perhaps an older version of the firmware was used corresponding to a different preprocessing. What is extra strange is that the given calibration file lists calibration parameters (distCorrectionX and distCorrectionY) that apparently were not used.

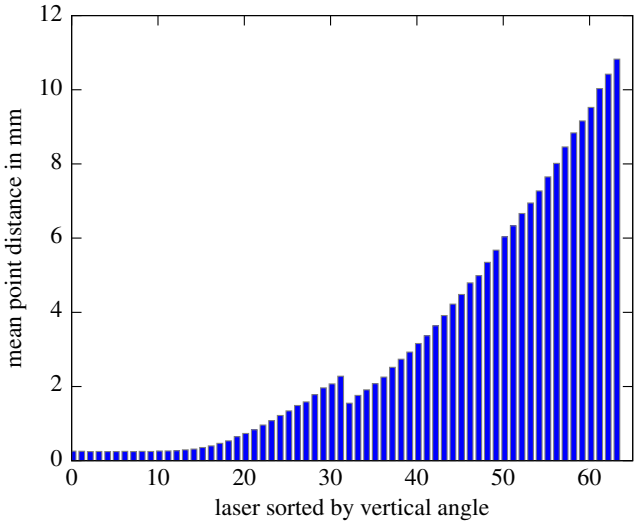
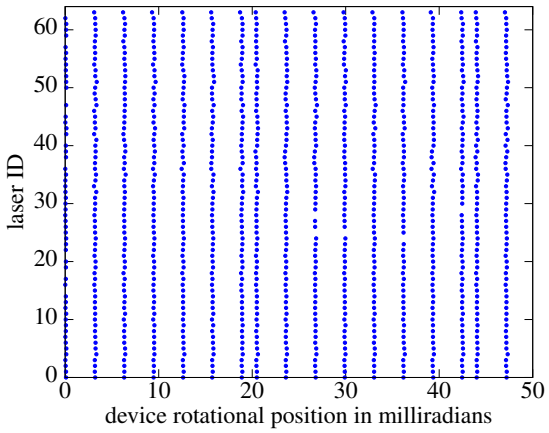


Figure 3: Mean 3D distance between the original point and reconstructed point for each laser.



(a) measured

Figure 4: Rotational position of the device for the measured points of the different lasers. Lasers are ordered here by their ID (not by the vertical angle). The figures only show part of the scan in the range of 0 to 0.05 radians. (a) The computed rotations from the point cloud using the reverse method for each point.

Appendix A - Preprocessing according to the manual

According to the User's Manual and Programming Guide, the preprocessing of the raw Velodyne data is a bit more complicated.

We start with a similar function as in Equation 1, this time taking an extra paramater Δd:

$$\mathbf{p}' = f'(d, \theta, l, \Delta d) = \begin{pmatrix} (d + \Delta d) * \cos\phi_l * \sin(\theta - \Delta\theta_l) - h_l \cos(\theta - \Delta\theta_l) \\ (d + \Delta d) * \cos\phi_l * \cos(\theta - \Delta\theta_l) + h_l \sin(\theta - \Delta\theta_l) \\ (d + \Delta d) * \sin\phi_l + v_l \end{pmatrix}. \quad (4)$$

Then, there is a non-symmetric adjustment for the x and y component. They are based on parameters Δd_l, Δdx_l and Δdy_l, named distCorrection, distCorrectionX and distCorrectionY, which are obtained using a factory calibration procedure with objects placed at specific positions from the sensor. These parameters are used as follows to get x and y specific distance adjustments:

$$\Delta dx = \Delta dx_l + \frac{f'_x(d, \theta, l, \Delta d_l)(\Delta d_l - \Delta dx_l) - 2.40}{25.04 - 2.40} \quad (5)$$

$$\Delta dy = \Delta dy_l + \frac{f'_y(d, \theta, l, \Delta d_l)(\Delta d_l - \Delta dy_l) - 1.93}{25.04 - 1.93}, \quad (6)$$

with $f'(\cdot) = \left(f'_x(\cdot), f'_y(\cdot), f'_z(\cdot)\right)^T$.

These adjustment are then used for computing the actual point, with the interesting note that the z value is computed using the y-specific distance adjustment:

$$\mathbf{p} = f(d, \theta, l) = \begin{pmatrix} f'_x(d, \theta, l, \Delta dx) \\ f'_y(d, \theta, l, \Delta dy) \\ f'_z(d, \theta, l, \Delta dy) \end{pmatrix}. \quad (7)$$

Again, apart from some rescaling that is left out, we here emitted an offset from the process called "pos" in the User's Manual, which is not explained further. This needs further investigation.

Appendix B - Reverting the more difficult preprocessing

Reverting the more complicated preprocessing described in Appendix 5 is not that straightforward. Getting the laser ID l can be done as in Section 3, but getting the other parameters requires an iterative method. First, an approximate value of both the distance and the rotation is computed as initial values:

$$\theta = \text{atan2}(\mathbf{p}_x, \mathbf{p}_y) + \theta_l, \quad (8)$$

$$d = ||p|| - \Delta d. \quad (9)$$

These values are then iteratively optimized. The rotational position is optimized using:

$$\theta \leftarrow \text{atan2}\left(\frac{\mathbf{p}_x + h_l * \cos(\theta - \Delta\theta_l)}{(d + \Delta dx) * \cos\phi_l}, \frac{\mathbf{p}_y - h_l * \sin(\theta - \Delta\theta_l)}{(d + \Delta dy) * \cos\phi_l}\right) + \Delta\theta_l, \quad (10)$$

where Δdx and Δdy are calculated using Equations (4) to (6) again.

The distance measurement is optimized as:

$$d \leftarrow \frac{p_x + h_l * \cos(\theta - \Delta\theta_l)}{\cos\phi_l * \sin(\theta - \Delta\theta_l)} - \Delta dx. \tag{11}$$

Or, to increase numerical accuracy, if $p_y > p_x$:

$$d \leftarrow \frac{p_y - h_l * \sin(\theta - \Delta\theta_l)}{\cos\phi_l * \cos(\theta - \Delta\theta_l)} - \Delta dy. \tag{12}$$

The parameters d and θ are refined using these equations until they converge. This usually takes only 2 or 3 steps. Our implementation therefore used a fixed number of 10 iterations (following well established decimal tradition).

Note, that we also formulated a closed form inverse of Equation (7). However, the solutions we found were not numerically stable and given the limited accuracy of the point clouds (millimeter accurate) the iterative approach is more feasible.

References

Triess, L. T., Peter, D., Rist, C. B., and Zöllner, J. M. (2020). Scan-based Semantic Segmentation of LiDAR Point Clouds: An Experimental Study. In *Proceedings of the IEEE Intelligent Vehicles Symposium Robotics*, pages 1116–1121.