Lidar FIltering Solution Overview

My primary decision was to operate solely with single lidar frame data, without incorporating any IMU data or temporal filtering. This approach typically offers greater robustness, as it relies on fewer data sources and thus remains less susceptible to synchronization and noise-related challenges. Moreover, when integrated into a complex scene perception system, it avoids introducing additional loopback connections that could potentially degrade quality through unwanted oscillations.

The most notable enhancement to the dust detection system that I envision involves leveraging the second response for each LiDAR pulse, a feature commonly provided by Ouster LiDARs. This data could offer direct insights to differentiate between dust and non-dust particles with greater accuracy and precision.

The filtering process I've devised involves splitting a lidar frame into two distinct sets: the Signal set and the Noise set: https://github.com/isapient/lidar_select_objects

This process unfolds in two key steps:

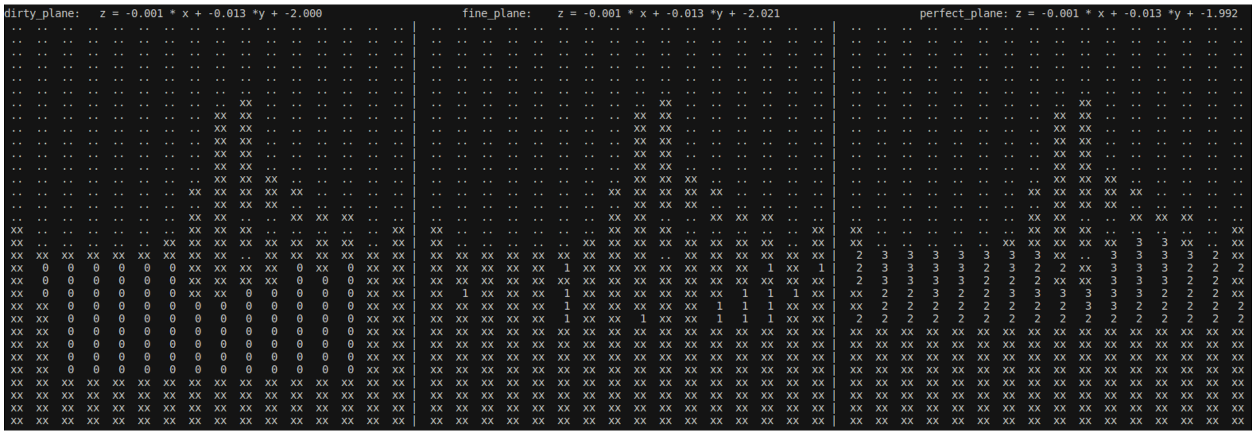
- ground detection
- object classification.

Ground Detection

Ground point detection relies on grid maps aligned with the lidar input aperture, facilitating a natural order processing of points within the PointCloud. These grid maps boast a 16x32 resolution, housing between 64 to 128 points, contingent upon the radar's resolution (OS1 generates a 512x64 cloud, while OS2 and OS3 produce a 512x128 cloud).

The primary objective of this phase is to estimate the elevation (z-coordinate) of the ground level within each grid-map cell, finally segregating all points lying below this threshold as "ground points." This process unfolds across three refining steps, wherein each step receives initial elevation estimations and a plane model ($z=ax+by+c$) defining the "zero level."

This step can be activated to produce textual debug output within the ROS package.



```
dirty_plane:   z = -0.001 * x + -0.013 *y + -2.000             fine_plane:    z = -0.001 * x + -0.013 *y + -2.021             perfect_plane: z = -0.001 * x + -0.013 *y + -1.992
 ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  ..  xx  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  xx  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  ..  xx  ..  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  xx  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  xx  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  xx  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  xx  xx  xx  xx  xx  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  xx  xx  xx  xx  xx  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  xx  xx  xx  xx  xx  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  xx  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  xx  ..  ..  ..  ..  .. | ..  ..  ..  ..  ..  ..  ..  ..  xx  xx  xx  ..  ..  ..  ..  ..
 ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  xx  xx  xx  ..  .. | ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  xx  xx  xx  ..  .. | ..  ..  ..  ..  ..  ..  ..  xx  xx  ..  ..  xx  xx  xx  ..  ..
 xx  ..  ..  ..  ..  ..  ..  xx  xx  xx  ..  ..  ..  ..  ..  xx | xx  ..  ..  ..  ..  ..  ..  xx  xx  xx  ..  ..  ..  ..  ..  xx | xx  ..  ..  ..  ..  ..  ..  xx  xx  xx  ..  ..  ..  ..  ..  xx
 xx  ..  ..  ..  ..  ..  xx  xx  xx  xx  xx  xx  xx  xx  ..  xx | xx  ..  ..  ..  ..  ..  xx  xx  xx  xx  xx  xx  xx  xx  ..  xx | xx  ..  ..  ..  ..  ..  xx  xx  xx  xx  xx   3   3  xx  ..  xx
 xx  xx  xx  xx  xx  xx  xx  xx  xx  ..  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  ..  xx  xx  xx  xx  xx  xx |  2   3   3   3   3   3   3   3  xx  ..   3   3   3   3   2  xx
 xx   0   0   0   0   0   0  xx  xx  xx  xx   0  xx   0  xx  xx | xx  xx  xx  xx  xx   1  xx  xx  xx  xx  xx  xx  xx   1  xx   1 |  2   3   3   3   3   2   3   2   2  xx   3   3   3   2   2   2
 xx   0   0   0   0   0   0  xx  xx  xx  xx   0   0   0  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx |  2   3   3   3   3   2   2   2  xx  xx   3   3   3   2   2  xx
 xx   0   0   0   0   0   0  xx  xx   0   0   0   0   0  xx  xx | xx   1  xx  xx  xx   1  xx  xx  xx  xx  xx  xx   1   1   1  xx | xx   2   2   3   2   2   3   3   3   3   3   3   2   2   2  xx
 xx  xx   0   0   0   0   0   0   0   0   0   0   0   0  xx  xx | xx  xx  xx  xx  xx   1  xx  xx  xx  xx  xx   1   1   1  xx  xx | xx   2   2   2   2   2   2   2   2   3   3   2   2   2   2   2
 xx  xx   0   0   0   0   0   0   0   0   0   0   0   0  xx  xx | xx  xx  xx  xx  xx   1  xx  xx   1  xx  xx   1   1   1  xx  xx |  2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
 xx  xx   0   0   0   0   0   0   0   0   0   0   0   0  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx
 xx  xx   0   0   0   0   0   0   0   0   0   0   0   0  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx
 xx  xx   0   0   0   0   0   0   0   0   0   0   0   0  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx
 xx  xx   0   0   0   0   0   0   0   0   0   0   0   0  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx
 xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx
 xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx
 xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx
 xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx | xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx
```

To optimize computation, I introduce two distinct cell values: **<xx>** signifying unknown ground level and **<..>** representing an empty cell.

By iterating over **<..>**, I streamline computation by skipping entire cells. Following each estimation, I extrapolate known ground levels to unknown cells via manual median filtration for grid maps.

For instance, I may initially designate <**xx**> to cells containing ground points within 3 meters' proximity, marking them as less confident due to potential obstructions like baggy hulls, wheels, and dust. Subsequently, I extrapolate these cell values using more distant cells.

The estimation of potential ground points is based on the average and mean square (msq) of the z-component for all points within the cell:
z - plane_z(x,y) < AVG(z) + K * MSE(z).

This adaptive approach offers robustness and efficacy, even with limited data.

```
const float lowpart_sigma_shift = -0.7;    // keeping ~25% of samples under threshold
const float distillate_sigma_shift = +0.7; // keeping 75% of remaining samples (18% of total samples)
const float normal_th_sigma_shift = +2.5;  // more than 3 sigma above average ground level
```

By adjusting the parameter K, I can regulate the quantity of samples falling below the threshold. The initial two refining steps retain approximately 18% of the lowest points, while the final 3rd step readjusts the threshold above the surface. If a cell lacks sufficient points below the threshold, its estimation defaults to <xx>.

I leverage points falling below the threshold for RANSAC estimation of the plane. This practice helps eliminate erroneous estimations on overly contaminated data. The resultant plane model is then employed alongside the grid map.

To address non-flat surfaces I use the grid elevation structure and also I've incorporated a "manual minimal elevation" parameter, typically ranging from 5 to 25 cm:
z - plane_z(x,y) < AVG(z) + min(elevation, K * MSE(z))

Following this step, all points are partitioned into two distinct sets: the "ground" set and the "other" set.

As a deliberate decision, I opted to incorporate all points within a 3-meter proximity to the ground set. This inclusion encompasses the buggy hull, typically devoid of points belonging to obstacles or even ground.

Object Classification

Once ground points are eliminated, the remaining points are grouped into smaller clusters, each potentially indicating obstacles, dust clouds, or noise. These distinct clusters facilitate the classification of different objects based on their properties.

Initially, the entire set of points is clustered using a cluster tolerance radius of 4.5 meters, a distance sufficient for accommodating the passage of a buggy. To ensure efficient computational performance, a kd-tree is used.

Subsequently, I iteratively transfer as much as possible non-obstacle or uncertain clusters to the "**noise**" set while retaining potential obstacles in the "**signal**" set.

This process is guided by various criteria:

Size

Remove clouds smaller than 9 points (smallest recognizable objects)

Penetrability

I introduce a point-based metric called "penetrability" for each cluster. To calculate it, I examine each point within the cluster and then average the values to derive an overall metric.

The process begins by identifying the 7 nearest neighboring points. These neighbors share LiDAR rays that are adjacent to the ray used for the analyzed point. To accomplish this, I create a duplicate set of cluster points projected onto a sphere, with each point having a corresponding prototype in the original cloud. Then, I utilize the kd-treeFLANN to locate the seven neighbors in the projected set and determine the prototypes of these neighbors.

The concept of penetrability relies on the premise that rigid objects have continuous surfaces, whereas dust clouds can be penetrated by laser beams, resulting in greater variation in the cloud along the rays. To quantify this, we measure the distance between the analyzed point and its neighboring point and divide it by the distance between the corresponding points' projections. Since the projection is perpendicular to the rays, a higher ratio indicates greater penetrability of an object in this neighborhood.

I compute the average value of the neighborhood using geometric averaging and then average it over all points in the cloud using arithmetic averaging, resulting in the PENETRABILITY feature.

This feature serves as the primary metric for distinguishing between dust clouds and objects. Notably, trees exhibit values closest to those of real-world objects due to the penetrability of their leaves. Therefore, I fine-tune the thresholds for this feature to effectively differentiate between dust clouds and trees.

Additionally, this feature's distribution varies depending on the LiDAR's ring structure, with distinct statistical distributions for 64-ring and 128-ring LiDARs. Consequently, I introduce two different thresholds to accommodate different hardware point sources.

Clusters with PENETRABILITY higher than threshold are moved from the “object” set.

Chaoticity and Sparseness

Since I already have the distances to neighbors, calculating additional metrics won't add to the computational complexity.

Here are two more metrics I can derive:

Sparseness:

This metric measures the average distance normalized by the distance from cluster centroid to the LiDAR itself. It indicates the percentage of rays inside the cloud without reflections. Higher values of SPARSENESS suggest the presence of dust clouds. This is sensor dependent metrics, so I tuned separate thresholds for 64 and 128 ring LiDARs.

Chaoticity:

This metric helps differentiate between structured surfaces like walls or poles, which exhibit uniform point patterns, and chaotic clouds such as dust.

I calculate the distances from each point to its neighbors and analyze these distances as statistical distributions. For structured surfaces, all distributions will be similar, whereas for chaotic dust clouds, they will vary.

I then find the average value for each distribution and compute two additional values: one for distances above the average (V1) and one for distances below the average (V2). The scalar V2/V1 varies more for irregular structures than for regular ones. Therefore, I calculate the CHAOTICITY as the Mean Squared Error (MSE) of V2/V1 over all points in the cluster.

CHAOTICITY serves as an invariant metric, allowing me to set a single threshold for any LiDAR.

By applying thresholds for CHAOTICITY and SPARSENESS, I effectively reduce the number of False Acceptances after the PENETRABILITY step.

Output Point Clouds

The ROS package I've developed publishes all points classified as obstacles in the signal topic **“/lidar_filter/signal”**.

The "ground" and "dust" point clouds are merged and published under the noise topic, which is accessible at **"/lidar_filter/noise"**.

Technical notes

Reduced point information

The original bag data includes fields for intensity, ambient, and reflectivity. While at least two of these fields provide independent data, I couldn't discern any patterns that would allow for effective data separation. As a result, I opted to utilize the PointXYZ type in the source code. This type is more compact, and the spatial information it provides is likely sufficient for subsequent perception algorithms.

In fact, I have an idea how to develop a "general dust detector." When a fine-grained dust cloud covers the lidar itself, it often results in the nearest ground zone exhibiting lower reflectivity. This phenomenon can be detected if necessary.

Load of Average and MSE

I find the operations of averaging and calculating Mean Squared Error (MSE) particularly valuable. They offer exceptional robustness for small datasets with O(1) computational complexity. As demonstrated in my system, these operations enable the design of intricate features effectively.