Point Types #2480

xmfcx started this conversation in **Design**

xmfcx on Mar 16, 2022 Maintainer edited

Main Sensing/Perception Discussion: #2456
This discussion is for deciding the point types accross the new autoware.

I think everybody agrees that we will use the http://docs.ros.org/en/lunar/api/sensor_msgs/html/msg/PointCloud2.html for point clouds.

Here is the link for PointFields to choose from.

My initial proposal is to go with PointXYZI for most of the stack.

name	datatype
x	FLOAT32
y	FLOAT32
z	FLOAT32
intensity	FLOAT32

A `FLOAT32` is 32 bits = 4 bytes. And each point is 16 Bytes, which is a tightly packed structure size, avoiding padding.

Time Field Management for the Motion Distortion Correction

This information is costly and there are multiple ways of adding this information.

Each point contains a UTC time with nanosecond precision

This is the cleanest way to implement and requires adding

name	datatype
unix_seconds	UINT32
nanoseconds	UINT32

fields to each point.

Pros:

Category
Design

Labels
component:percept...
component:sensing

6 participants

- This format is very clean because any point is stamped in an absolute way.
- It is straightforward to implement across different lidar vendors.

Cons:

- It uses 8 Bytes of extra space per point.

The message header contains the UTC time information, point times offset from there

This method attempts to reduce the repeated information since the points from the cloud will belong to a small time window.

Header contains the old ros::Time message which contains the time that belongs to the first point in the cloud.

And rest of the points only contain the following field:

name	datatype
nanoseconds	UINT32

And it denotes the nanoseconds passed from the first point in the cloud.

Note that a `UINT32` can represent 2^32 numbers (including 0 :p)

And it can represent 2^32 * 10^-9 = 4.294967295 seconds of time. So if a scan is longer than this, it will overflow.

Pros:

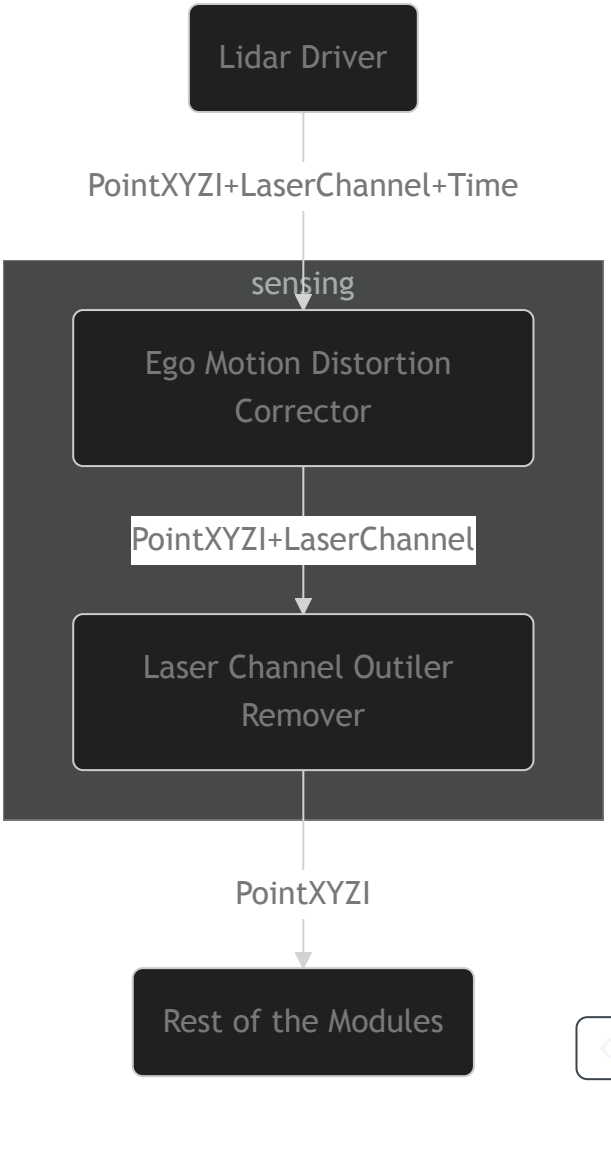
- It uses 4 Bytes of extra space per point.

Cons:

- It is more cumbersome to implement, encode/decode.

`laser_channel` Field for the Laser Channel based processing

This is for denoting the `laser_channel` field for `channels` in rotating lidar scanners. And represents `lines` for solid state lidars.



↑ 1 👍 1

5 comments · 10 replies

Oldest | Newest | Top

xmfcx on Mar 29, 2022 Maintainer Author

https://github.com/autowarefoundation/autoware.universe/blob/2f3e335ab289c2e940229bd659589d081f6408ea/common/autoware_point_types/include/autoware_point_types/types.hpp#L57-L76

https://github.com/autowarefoundation/autoware.universe/blob/2f3e335ab289c2e940229bd659589d081f6408ea/sensing/pointcloud_preprocessor/docs/dual-return-outlier-filter.md

↑ 1 0 replies

chishengshih on Mar 30, 2022 Collaborator

Hi,

I suggest adding 'clock source' into UTC format.

A complex system may have several clock sources and will fuse the data from multiple sensors, which use different clock sources. For example, the bus may have one GPS clock for the computer and another GPS clock for each LIDAR. The drift among clock sources may lead to wrong results if the system is NOT aware of the difference in clock sources. Clock drift within 100ms will prolong the length of the detected objects. The clock source can be represented by the UUID of the device, for example, or URL to the NNTP server..

Daniel

↑ 1 0 replies

ralwing on Apr 1, 2022

Hello,
The PointCloud is currently heavily used by Lidar Perception, especially by ray_ground_estimator. Currently, this module use ray casting, which will be impossible after the distortion correction.
Another problem is that the ray_cast should be able to know the source of each ray to properly evaluate the ray angles - The longer the distance is from base_link the higher is the evaluation error.

↑ 1 3 replies

xmfcx on Apr 5, 2022 Maintainer Author

Hi **@ralwing**,

> Currently, this module use ray casting, which will be impossible after the distortion correction.

I'm assuming you are talking about the ray_ground_classifier.

And its design doc.

This filter divides the point cloud into regions (rays), filtered by the local heights, sorted by x-y euclidean distance, and classifies the points as ground or not if difference is too high.

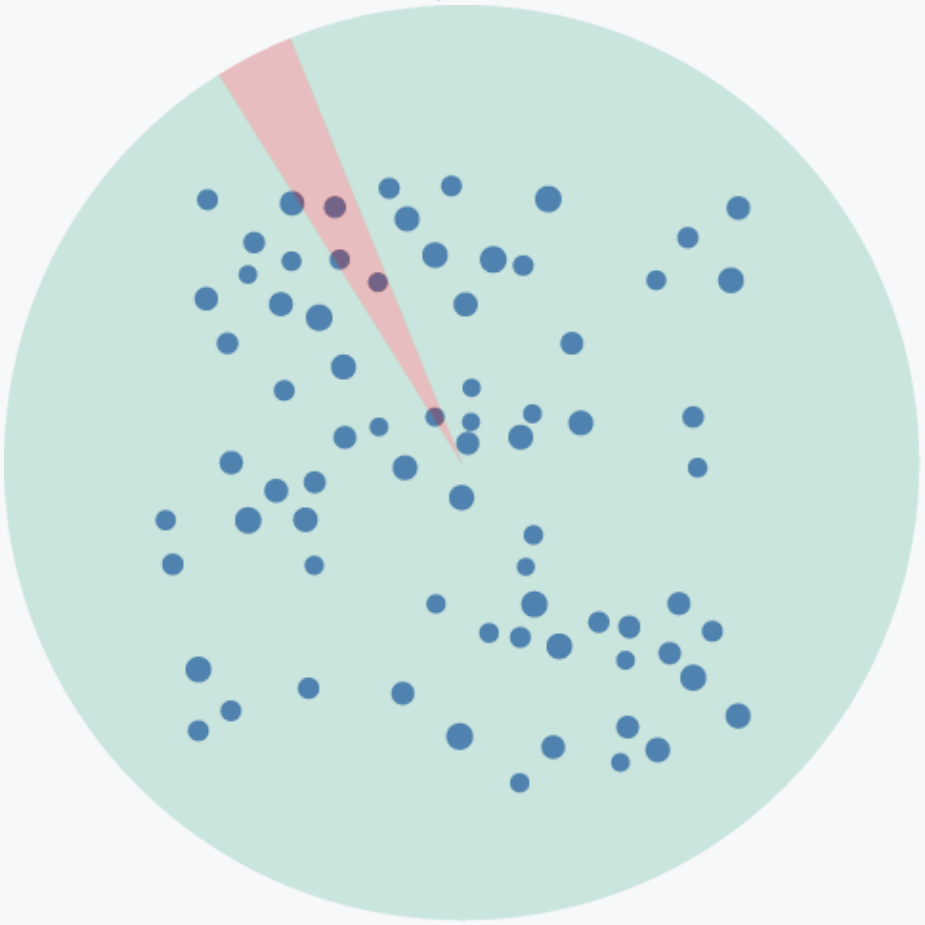
It doesn't do ray casting to my knowledge. Lidars are ray casting sensors themselves.

I'd say that with egomotion correction done, the full point cloud will make more sense and the ray_ground_classifier will work better.

> Another problem is that the ray_cast should be able to know the source of each ray to properly evaluate the ray angles - The longer the distance is from base_link the higher is the evaluation error.

A ray is basically a top down pie section of the point cloud. You can divide the point cloud into "rays" in any angle size you want. And that angle size is a parameter.

The red part would represent a ray in the filter.



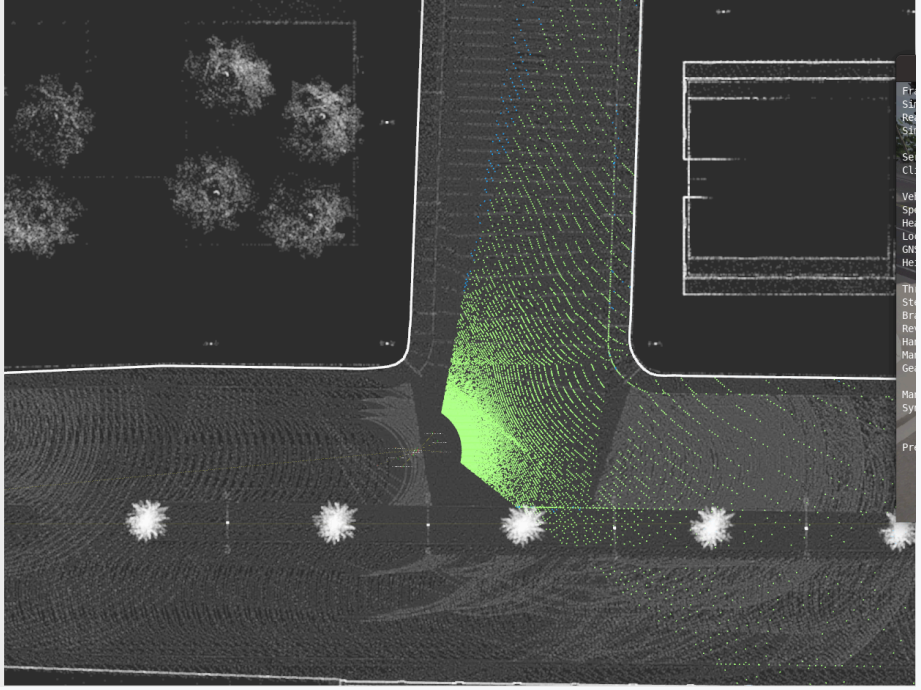
(normally it'd be of 0.01rad or 0.57deg)

ralwing on Apr 6, 2022 edited

Ray Ground Classifier (RGC) works by getting a point cloud, dividing it into rays, and then analyzing the divided points to classify them as points on the ground or above the ground. The division into rays should reflect the operation of lidar - the RGC should see the world in the same way as lidar to work optimally.

Currently, the RGC gets points in the base_link frame and creates rays from that point. So the rays created this way do not correspond to the rays from lidar because they are created from a different location. This can cause misclassification of the outermost points, as you can see in the attached image (the problem becomes apparent when standing in front of a hill).

Her lidar is about 2 meters in from of base_link:



The green points are on the ground, the blue points are above. The points on the left have been misclassified as blue, because for the radius so created (relative to base_link) the initial points are missing and the first points in that radius are already on the hill.

Once we set base_link into the same location, all points are correctly classified.

xmfcx on Apr 6, 2022 Maintainer Author

@ralwing Of course the origin of the lidar should be known for RGC to work correctly. I agree with that. But ego-motion distortion correction doesn't translate the point cloud so this shouldn't be an issue.

piotr-zyskowski-rai on Apr 5, 2022 Collaborator edited

I believe, it is rather important to limit the data size of a point cloud. In our experience, big messages are not well supported in DDS/RMW implementations.

Considering a format like this:

name	datatype
x	FLOAT32
y	FLOAT32
z	FLOAT32
intensity	FLOAT32
unix_seconds	UINT32
nanoseconds	UINT32

name	datatype
laser_channel	UINT8 (assuming lidars have no more than 256 lines)

It has 25 bytes per point. In the case of Velodyne VLS 128 set for 600RPM, it produces over 230k points each revolution. A single message has around 5.5MB of data, which is quite a lot for DDS/RMW implementation to handle with 10Hz frequency. Size reduction would be very desireable, I think.

Intensity

Is there a reason for intensity to be as high resolution as FLOAT32? Wouldn't the resolution provided with UINT16 or even UINT8 be sufficient for Autoware applications?

Laser channel

What would be the data type of the `laser_channel` data field? It seems UINT8 would be sufficient for most lidar models.

Time representation

I think a better approach would be to use a full timestamp in the header and offsets in the points since it reduces the overall message size.

To further reduce the point cloud size, offsets could be set for a ranges of points rather than to each point. Such metadata might be sent on the dedicated topic and nodes interested could use it. However, the question is whether such an approach would be accurate enough to be used for distortion reduction or other required applications.

X,Y, Z position information

FLOAT32 seems fine to represent points. However, depending on the expected resolution data size might be reduced by half when representing the position in INT16. With 1cm level accuracy, it can represent a range <-327,68; 327,67>m which seems enough for most Lidars. Or with 0.5cm level accuracy with range reduced by half.

Of course, the drawback here would be greatly limited distance resolution. And it is not the standard approach to point representation.

↑ 1 🚀 1 7 replies

Show 2 previous replies

mitsudome-r on May 10, 2022 Maintainer

I can create a sample nodes to investigate iceoryx zero-copy feature against a very large pointcloud message.

mitsudome-r on May 10, 2022 Maintainer

cc. @piotr-zyskowski-rai

piotr-zyskowski... on May 17, 2022 Collaborator edited

@xmfcx

> Even if we make the point cloud size constant and use zero-copy method from iceoryx?
> I'm asking because I've not tried it yet.

I looked over the description of this solution and I have one concern. From iceoryx page:

> iceoryx uses a true zero-copy, shared memory approach that allows to transfer data from publishers to subscribers without a single copy

So it would be great in case of publisher and subscriber are on the same machine, but it won't work via the network I suppose. Can it be expected that no big point cloud will be sent over the network?

@mitsudome-r I did not perform the iceoryx performance tests yet. I will take care of them soon however

xmfcx on May 17, 2022 Maintainer Author

- publish from drivers in each `epoch time + n * 101ms`
- https://github.com/google/draco

piotr-zyskowski... on May 24, 2022 Collaborator edited

I did some tests of iceoryx

I used `rmw` implementation described on iceoryx github page, except instead of calling `./install/iceoryx_posh/bin/iox-roudi` to run the daemon, I run `/opt/ros/galactic/bin/iox-roudi`

I created an artificial point cloud formatted as x,y,z floats

I planned to test point clouds from 100'000 to 15'000'000 points which was feasible for `cyclone_dds`. But for iceoryx after reaching 400'000 (around 4,5MB) I got the following error:

```
2022-05-24 18:44:33.707 [ Fatal ]: The following mempool
are available:  MemPool [ ChunkSize = 168, ChunkPayloadSize
= 128, ChunkCount = 10000 ]  MemPool [ ChunkSize = 1064,
ChunkPayloadSize = 1024, ChunkCount = 5000 ]  MemPool [
ChunkSize = 16424, ChunkPayloadSize = 16384, ChunkCount =
1000 ]  MemPool [ ChunkSize = 131112, ChunkPayloadSize =
131072, ChunkCount = 200 ]  MemPool [ ChunkSize = 524328,
ChunkPayloadSize = 524288, ChunkCount = 50 ]  MemPool [
ChunkSize = 1048616, ChunkPayloadSize = 1048576, ChunkCount
= 30 ]  MemPool [ ChunkSize = 4194344, ChunkPayloadSize =
4194304, ChunkCount = 10 ]Could not find a fitting mempool
for a chunk of size 4800149
```

```
2022-05-24 18:44:33.708 [Warning]: ICEORYX error!
MEPOO__MEMPOOL_GETCHUNK_CHUNK_IS_TOO_LARGE
```

Apparently, it would not fit a single point cloud of mentioned Velodyne VLS 128 (with format as stated before) that has over 5,5MB of data.

I do not know if the implementation could be change to fit more data but I would conclude that usage of iceoryx might be more cumbersome than anticipated.

aohsato on Jun 28, 2022 Collaborator

@xmfcx (**@badai-nguyen** **@drwnz** **@miursh**)
Hi! We'd like to discuss azimuth in addition.

We currently use a pointcloud_ex topic that also includes distance, azimuth, and return type for some sensing modules (the data size is large, so we use a container as shown in velodyne_node_container.launch.py).

Currently, the azimuth output from sensor drivers is a lidar-dependent value, and the type is not fixed.
We'd like to standardize it in the following issue, so would you please join us in the discussion?
autowarefoundation/autoware.universe#1127

↑ 1 👍 1 0 replies