[proposal] A new pointcloud map interface for dynamic map loading #2812

kminoda started this conversation in **Design**

kminoda on Aug 23, 2022 Collaborator edited

We, TIER IV, would like to propose a new architecture for point cloud map loading.

Goal: Expand the size limit of the point cloud maps to improve the scalability of Autoware

Introduction

Current Autoware is not scalable in terms of the size of the map, since it loads the whole point cloud (PCD) map at once. As far as we know, the size limit of PCD map in the current Autoware is around 2GB which is determined by the maximum size of a topic message in cycloneDDS. Thus, we are working on a new type of algorithm for loading a PCD map: dynamic map loading (DML).

dynamic_map_loading-2022-06-29_15.19.17_10x.mp4

Through our experiment, however, it turned out that the current interface is not suitable for efficient DML. For example, the naive DML (shown in the above video) newly loads all the PCD maps within the range of 200m from the ego-vehicle every several seconds, which may be too inefficient to perform in real time on limited computational resources.

Thus, we propose a differential DML, which reuses the overlapped PCD grids from the previous loading area to reduce the computation. For example, in the case below, the naive DML loads 38 grids (shown in the gray area), while the differential DML loads 6 grids (shown in blue). Note that the differential DML has to remove 6 grids (shown in red) in this case.

Category

Design

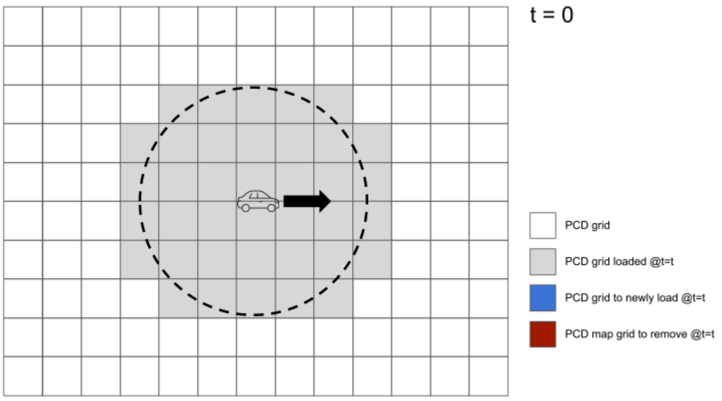
Labels

component:percept...

component:localiza...

component:map

5 participants



Unfortunately, the current map interface cannot provide sufficient information for client nodes to handle this complex management of PCD maps.
Thus, we, TIER IV, would like to propose to the AWF community a new interface to enable more flexible map loading.

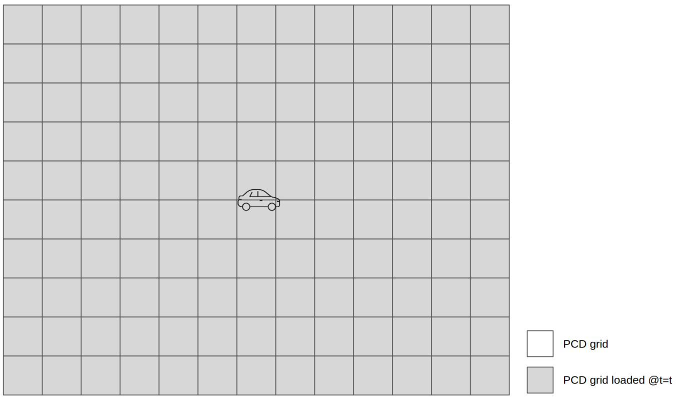
Note that this is a proposal for an additional interface (service) as an option, and is not intended to remove any current interface.
Here we also assume that the PCD map is divided beforehand, e.g. into 20m x 20m grids (see an additional proposal for map dividing format).

Possible map loading scenarios

Here we briefly introduce possible map loading scenarios.

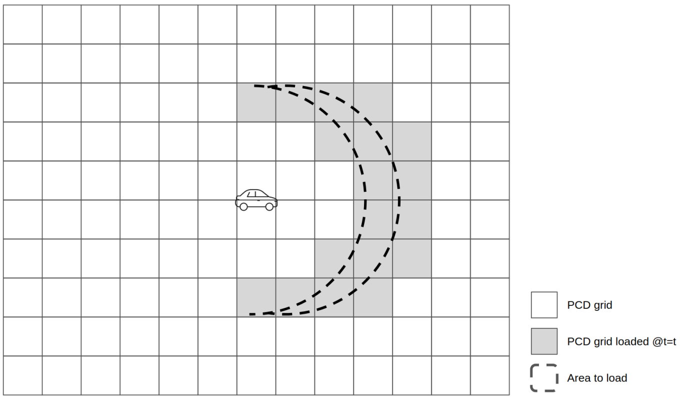
Whole area loading

This is the only scenario that the current Autoware supports, in which the client nodes load the whole available map at once.
As long as the size of point cloud map does not induce any issues (i.e. communication size limit mentioned above), this scenario would be the most simple solution.
Note that you can perform whole map loading in the same way as in the current Autoware, since the proposed architectures have no influence on the existing interface.



Partial area loading

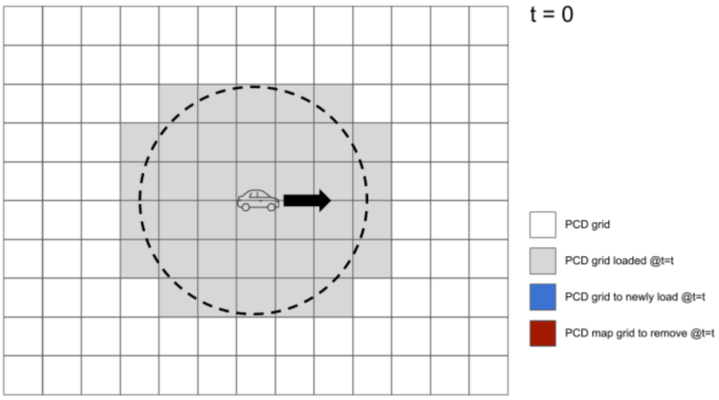
This scenario considers a case when a node (e.g. `pose_initializer`) only wants a limited area from the available PCD map. We assume that, given the area query, the node loads the PCD grids that overlap with the area query. This scenario may be needed when the point cloud map is too large to load at once.



Differential area loading

In this scenario, a node (e.g. `ndt_scan_matcher`) loads additional PCD grid maps (shown in blue) as well as removes maps that are no longer necessary (shown in red) at each step.
By reusing the maps that the node already has (shown in grey), the node can significantly reduce the computation that occurs in loading and preprocessing the map.
If we assume that a client node wants a set of map grids $M(t)$ for $t = t$ with differential area loading, the client node at $t = t + 1$ loads $M(t+1) \backslash M(t)$ and unloads $M(t) \backslash M(t+1)$ where $A \ / \ B = \{x \in A \mid x \notin B\}$.

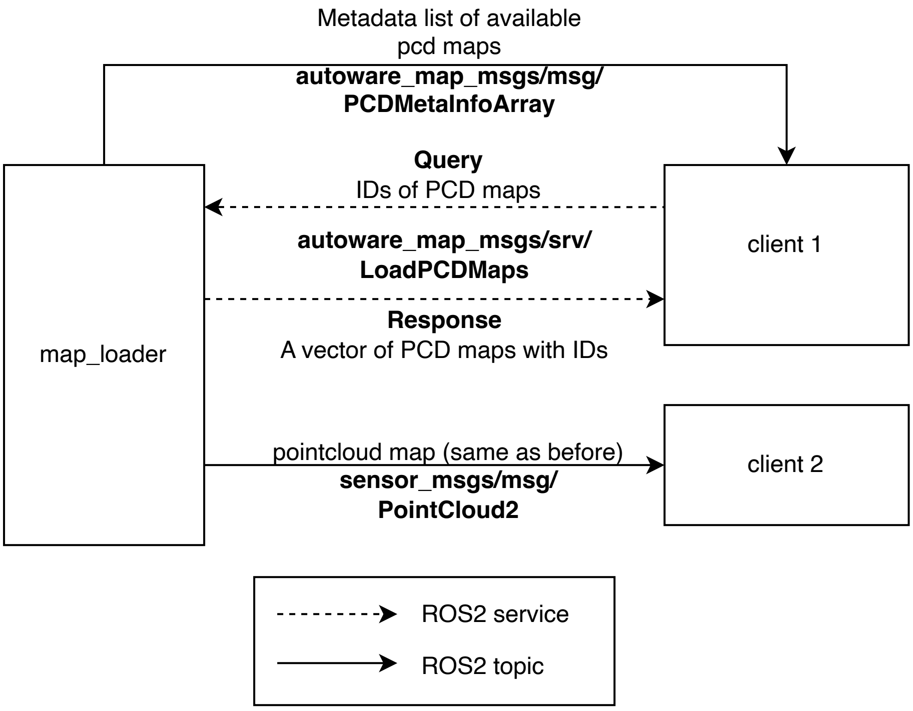
As discussed above, this scenario is needed when the point cloud map is too large and the partial loading scenario is not efficient enough.



Proposed architectures

We have two proposals, both of which can achieve the above-mentioned three scenarios. Since they both have their pros and cons, we would like to ask for your opinions from various perspectives.

Proposal A: sending ids as a query


Metadata list of available pcd maps
autoware_map_msgs/msg/ PCDMetaInfoArray
Query
IDs of PCD maps
autoware_map_msgs/srv/ LoadPCDMaps
Response
A vector of PCD maps with IDs
pointcloud map (same as before)
sensor_msgs/msg/ PointCloud2
map_loader
client 1
client 2
ROS2 service
ROS2 topic

The architecture of proposal A is shown below. A client that want to use the new interface ("`client 1`" in the figure) first subscribes a message (see here for the definition) that contains a dictionary of each grids' ID and its region information.
Using this information, the client selects the maps it wants and throw the query to the `map_loader` with autoware_map_msgs/srv/LoadPCDMaps. `map_loader` loads the required maps and send them back as a response.

Note that in this case, we are also considering creating a library that covers all three scenarios mentioned above.

The three scenarios mentioned above can be achieved as follows:

- Scenario 1 (whole area loading): use the `/map/pointcloud_map` topic as before (as shown in `client 2` in the above figure).
- Scenario 2 (partial area loading): calculate the map IDs you want in client side, using the given metadata list, and request them via the proposed service interface
- Scenario 3 (differential area loading): calculate the map IDs you want in client side, using the given metadata list, and request them via the proposed service interface

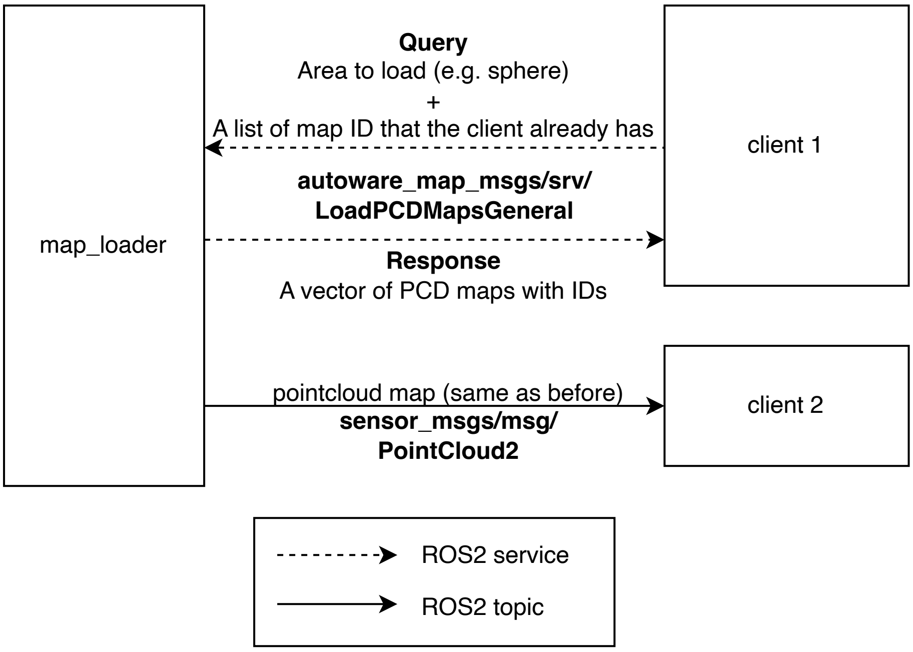
Pros:

- more simple than proposal B, and easier to understand

Cons:

- unnecessarily too general to achieve the above-mentioned scenarios
- heavier implementation cost for the client side (which can be reduced by the library, but still requires a maintenance cost)

Proposal B: passing area and map ids that the client already has


Query
Area to load (e.g. sphere)
+
A list of map ID that the client already has
autoware_map_msgs/srv/
LoadPCDMapsGeneral
Response
A vector of PCD maps with IDs
map_loader
client 1
pointcloud map (same as before)
sensor_msgs/msg/
PointCloud2
client 2
ROS2 service
ROS2 topic

The architecture of proposal B is shown below. In this proposal, the client (`client 1` in the figure) sends the following two data as a query:

1. mode (0: partial area loading, 1: differential area loading)
2. the map area that the client wants (i.e. spherical area)

In case of `mode=0`, given the above two data, the `map_loader` returns all the map grids and their IDs that overlaps with the queried area.
In addition, when the client wants to use differential map loading (e.g. for more efficient computation), the client should additionally sends the following query:

3. map ids that the client already has

In case of `mode=1`, given the above three data, the `map_loader` returns...

- the map grids and their IDs (for the grid that overlaps with the queried area and that is not included in the third data)
- the map grids' IDs (for the grid that overlaps with the queried area and that is included in the third data)

The three scenarios mentioned above can be achieved as follows:

- Scenario 1 (whole area loading): use the `/map/pointcloud_map` topic as before (as shown in `client 2` in the above figure).
- Scenario 2 (partial area loading): use the proposed service interface (leave `already_loaded_ids` empty in this case)
- Scenario 3 (differential area loading): use the proposed service interface

The differential DML is expected to use `mode=1`.
(See also: autoware_map_msgs/srv/LoadPCDMapsGeneral)

Pros:

- necessary and sufficient for achieving the above-mentioned three scenarios (not too general)
- able to reduce the implementation cost of the client side

Cons:

- complicated interface and thus more difficult to understand than proposal A

Other candidates that we had in mind

See here

↑ 7 👍 10 ❤️ 3

8 comments · 10 replies

Oldest Newest Top

xmfcx on Sep 1, 2022 Maintainer

@kminoda

This proposal is very throughly thought and neatly designed, thanks for the work.

I really like the Proposal B. Proposal A is puts a lot of map loading logic on client side. This makes it hard if people want to add different clients.

Potential optimizations

I think the further optimizations can be made by providing precomputed voxelgrid structures to the `ndt_matcher`.

https://github.com/autowarefoundation/autoware.universe/blob/main/localization/ndt_scan_matcher/src/ndt_scan_matcher_core.cpp#L431-L433

Here ndt matcher is initialized with the map points:

```
pcl::shared_ptr<pcl::PointCloud<PointTarget>> map_points_pt
pcl::fromROSMsg(*map_points_msg_ptr, *map_points_ptr);
new_ndt_ptr->setInputTarget(map_points_ptr);
```

Bottlenecks

Ignoring the transmission cost, here is the most time consuming part.

ROS to PCL conversion

```
pcl::shared_ptr<pcl::PointCloud<PointTarget>> map_points_pt
pcl::fromROSMsg(*map_points_msg_ptr, *map_points_ptr);
```

Here the conversion of the messages cost some significant milliseconds depending on the size of the point cloud.

Voxel grid construction

```
new_ndt_ptr->setInputTarget(map_points_ptr);
```

Here, it calls this function https://github.com/tier4/ndt_omp/blob/tier4/main/include/pclomp/ndt_omp.h#L127-L135

```
inline void
        setInputTarget(const PointCloudTargetConstPtr &cloud)
{
        pcl::Registration<PointSource, PointTarget>::setInputTarget
        init();
}
```

I'm guessing the `init()` is the time consuming part here.

And it finally the init() https://github.com/tier4/ndt_omp/blob/tier4/main/include/pclomp/ndt_omp.h#L326-L334 :

```
/** \brief Initiate covariance voxel structure. */
void inline
        init()
{
        target_cells_.setLeafSize(resolution_, resolution_, resolut
        target_cells_.setInputCloud(target_);
        // Initiate voxel structure.
        target_cells_.filter(true);
}
```

So if we could precompute these voxel data structures for each grid (from the Proposal B) then this initialization would be near instantaneous.

How to overcome bottlenecks

As long as we are using native PCL, I think we can't avoid the `ROS to PCL conversion` bottleneck.

And similar can be said for the initialized voxel structure too. But I am open to your opinions on this.

Proceeding without optimizations first

I think to ensure compatibility with current stack and enable initial implementation quickly, your current plan is sound.

My summary of the `mode=1` aka `Scenario 3`:

Extra points:

- How much of a history the client would like to keep would be on the client side configuration. (Like remove by distance, size, time)
- Client to server query can include the metric radius of the required grids too. Like 100m/300m around the vehicle.
- Client to server query can include the shape too (like rectangle / circle) (low priority)

Simplifying the client implementation

Maybe we can provide a helper class for the client. Role would be to keep track of the id's with the protocol defined in `Scenario 3`. If someone wants to implement a new client like NDT, they would use this class and just call a function like `getPointsAround(Pose pose)` and the complexity would be abstracted away.

↑ 1 ❤️ 2 2 replies

kminoda on Sep 1, 2022 Collaborator Author edited

Thank you for the comments! Overall, I agree with your points.

> I think to ensure compatibility with current stack and enable initial implementation quickly, your current plan is sound.

We also think it is better to first define the architecture and integrate DML into the system, and then discuss the reduction of computation as needed.

> Simplifying the client implementation

Thank you! We agree that providing a map loading library (or a helper class) would be important to reduce the implementation cost on the client side as much as possible.

--
BTW, as you mentioned in "potential optimizations", the voxel grid construction is one of the most time comsuming parts in the current `ndt_scan_matcher` (which is here in pclomp). So it would be a reasonable solution to compute the voxel grid construction outside of `ndt_scan_matcher` (e.g. `pointcloud_map_loader`) to address the transmission bottleneck.

However, as far as we have experimented with the prototype DML, communication bottlenecks were not a problem even when pub/sub-ing raw point clouds. Rather, your suggestion should be discussed as a solution to the next challenge of reducing memory bandwidth. We know that DML, which involves exchanging and preprocessing raw map point clouds, consumes a fair amount of memory bandwidth. We believe that reducing this memory bandwidth will eventually become important for more efficient DML/Autoware, and we should consider a wide range of solutions, including the measures you have suggested.

yukkysaito on Sep 22, 2022 Maintainer edited

Thank you for the comment.
We were torn between A and B. I agree with your opinion and Proposal B.

> Potential optimizations

Good idea.
Continued discussion is needed because we must be careful not to make the interface of the map server exclusive for ndt.

❤️ 1

xmfcx on Sep 1, 2022 Maintainer

Another way of handling this is to make map loading module a library instead of a node.

Matcher node would use this map loader library to load maps.

Pros:

- No server-client interfacing complexity
- Right now we have to convert from `PCD file -> PCL msg -> ROS msg -> PCL msg`
  - This method would turn this to `PCD file -> PCL msg` only.

Cons:

- Slightly lesser modularity

↑ 1

1 reply

kminoda on Sep 6, 2022 Collaborator Author

Thanks for the proposal!
Although we also considered that solution, we reached to an conclusion that the proposed DML approach is preferred due to the following reasons:

- Directly loading PCD maps in each nodes will solve the transmission bottleneck, but not scalable enough to solve the

> RAM size bottleneck (thus, even in this case, we will eventually need more scalable solution such as DML in the near future)
> - If we consider a use case in which the multiple nodes that need the pcd maps exist, the above solution will somewhat force the users to put all the nodes in one computer, even if they want to use multiple computers to distribute the Autoware nodes.

kminoda on Sep 22, 2022 Collaborator Author

@yukkysaito @mitsudome-r @YamatoAndo @xmfcx
Do you have any other comments on this?

Currently, I would prefer proposal B, as I believe it is better to focus on making the clients' implementation as simple as possible. If no one has any disagreeing opinion, I would make B as a final decision and close this discussion.

↑ 1 👍 1 2 replies

yukkysaito on Sep 22, 2022 Maintainer

I'm ok 👍 I agree with proposal B

❤️ 1

xmfcx on Sep 28, 2022 Maintainer

Sorry for late response too, I also agree with B.

❤️ 1

kminoda on Sep 27, 2022 Collaborator Author

It seems that there are no other opinions or alternative plans.
We would like to go on and start integrating this new feature in Autoware using proposal B interface.
Related WG log:
https://github.com/orgs/autowarefoundation/discussions/2879

↑ 1 0 replies

simon-t4 on Oct 11, 2022

This may be late but to add to the discussion:

- A central node publishing portions of the map also will help to extend to the case where the map is stored in the cloud and loaded/updated to the vehicle as needed.

- the optimisation is important: perhaps the point cloud loader should be configurable to pre-compute different views of the map (definable by common PCL functions - sub-sampling, voxel grid etc), and then client can select which type of map they request.
- another consideration is the latency in receiving the dynamic map info: is the size of requested area just conservatively estimated (larger) to ensure no latency problem, or is there some logic to predict future needs and "pre-order" map info, thus minimising size of requested areas.

↑ 1

1 reply

kminoda on Oct 11, 2022 Collaborator Author

Thank you for the interesting comments!

As you've mentioned, I think one of the future development goal for this dynamic map loading is to optimize the performance (the cost for communicating the maps, the cost for computing the preprocessing of the maps, etc). I am not facing such necessity for performance optimization although my current implementation is a very simple one (no "pre-compute" nor "pre-order" logics). However, I agree that your two advises are both reasonable approach for further optimization, as well as using zero-copy ROS middleware or pre-computing the preprocessed maps beforehand (even before launching Autoware).

I believe that it is better to move on with proposal B for now (as long as the current dynamic map loading functionality is working fine on most of the reference vehicles), and we can discuss how to optimize the performance if necessary.

Shin-kyoto on Apr 6, 2023 Collaborator

edited ▾

@yukkysaito @mitsudome-r @xmfcx
We, TIER IV, would like to implement Proposal A (sending ids as a query) as an interface for the map_loader, in addition to Proposal B. Can you give us your comment?

Introduction

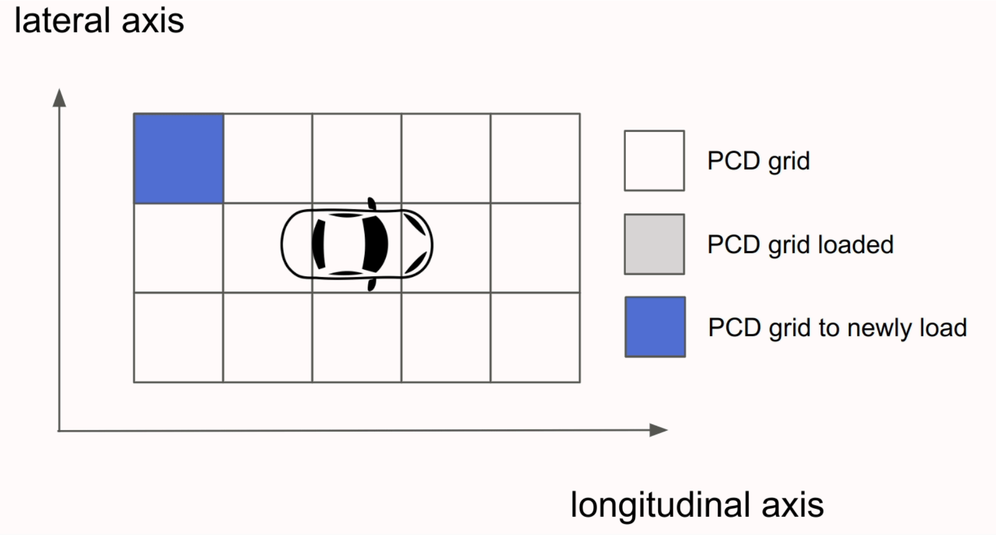
Previous discussions only considered scenarios where the client required a spherical area. And we chose proposal B because it makes the clients' implementation as simple as possible.

However, some nodes (e.g. `elevation_map_loader`) need a more flexible map publication architecture to support elevation maps for large-scale maps . Therefore, we would like to implement an interface that allows the client to specify an ID to receive the map, rather than being limited to a spherical shape. This will allow more nodes to handle large-scale maps in Autoware.

Sequential whole area loading scenario

Here we briefly introduce map loading scenario added from previous discussion.

This scenario considers a case when a node (e.g. `elevation_map_loader`) in initialization step wants a whole area from the available PCD map, the size of which exceeds the size limit of PCD map. (The size limit of PCD map in the current Autoware is around 2GB which is determined by the maximum size of a topic message in cycloneDDS.) So this node can't use Whole area loading. As an alternative, the node can get PCD map cells one by one, sequentially. We assume that each PCD map cell does not exceed the size limit of PCD map.

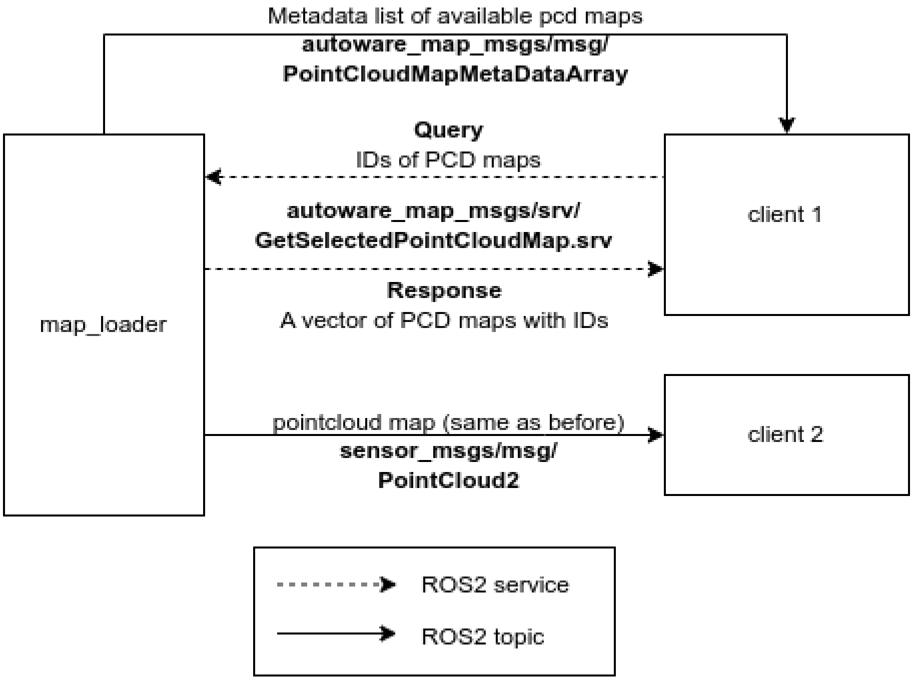


In this scenario, client node can't use Partial area loading implemented by Proposal B to get a whole area from the available PCD map because client node does not know the whole area size. We need to pass the information of PCD map to client.

Proposed architecture

We want to add this architecture, almost equal to proposal A.

A client that want to use the new interface ("client 1" in the figure) first subscribes a message (see here for the definition) that contains a dictionary of each grids' ID and its region information.Using this information, the client selects the maps it wants and throw the query to the map_loader with autoware_map_msgs/srv/GetSelectedPointCloudMap. map_loader loads the required maps and send them back as a response.


Metadata list of available pcd maps
autoware_map_msgs/msg/
PointCloudMapMetaDataArray
map_loader
Query
IDs of PCD maps
autoware_map_msgs/srv/
GetSelectedPointCloudMap.srv
Response
A vector of PCD maps with IDs
client 1
pointcloud map (same as before)
sensor_msgs/msg/
PointCloud2
client 2
ROS2 service
ROS2 topic

In the Sequential whole area loading scenario, the client node select PCD map cell's ID one by one, using the given metadata list. Then, the client requests the map ID and map_loader serves PCD map corresponding to PCD map cell's ID.

↑ 1 👍 1 1 reply

kminoda on Apr 6, 2023 Collaborator Author edited ▾

Thank you! I have one small question:
If the above interface being added, why not removing the current interface (proposal B)? Given that all the scenario can be covered solely by proposal A interface, it would be more persuasive if you could share us any reason for leaving the original interface as well as the new one.

(I'm mostly fine with your proposal, but just want to clarify why using both)

Shin-kyoto on Apr 6, 2023 Collaborator edited ▾

@kminoda

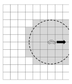
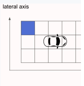
Thank you for your comment!

The Common Cases can be handled using the interface provided in Proposal B, and most clients that use differential map loading behave according to this scenario. For example, nodes such as `ndt_scan_matcher`, `pose_initializer`, and `voxel_based_compare_map_filter` all comply with the Common Cases shown in the figure below. In this case, using Proposal B makes implementation easier, and the client does not need to be aware of the specifics of map(e.g. map cell's id, divided cell's width or length)

On the other hand, the `elevation_map_loader` falls into the exceptional cases category. This node needs to read the whole map by specifying PCD map cell with an ID, rather than using a spherical map loading approach. Proposal B cannot handle this case, so Proposal A needs to be introduced.

It is important to note that, **in almost all cases, the interface provided in Proposal B is the optimal solution**. It is easy to implement and does not require the client to be aware of the specifics of the map. Therefore, removing Proposal B is not reasonable.

In summary, my proposal is to use the interface provided in Proposal B for the common case and the interface provided in Proposal A only for exceptional cases.

case	scenario	example of client node	descri
Common Cases	- Partial area loading - Differential area loading	ndt_scan_matcher pose_initializer voxel_based_compare_map_filter	- The node requse maps includ certair - Load map spheri - Almc scenai can be covere this ca e.g. 
Exceptional Cases	- Sequential whole area loading	elevation_map_loader	- The node reques specif PCD m cell by - On ra occasi the cli node r to spe PCD m cell. e.g. 

↑ 1 👍 2 1 reply

kminoda on Apr 6, 2023 Collaborator Author

Got the point! Thanks

👍 2

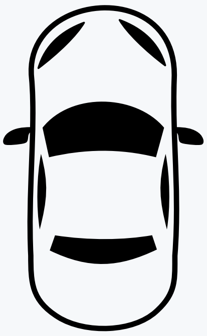
Shin-kyoto on Apr 6, 2023 Collaborator

@xmfcx
Do you have any other comments on this?
https://github.com/orgs/autowarefoundation/discussions/2812#discussioncomment-5540316

↑ 1 2 replies

xmfcx on Apr 7, 2023 Maintainer edited

Can you use following svg file which is downloaded from here in the gif animation?

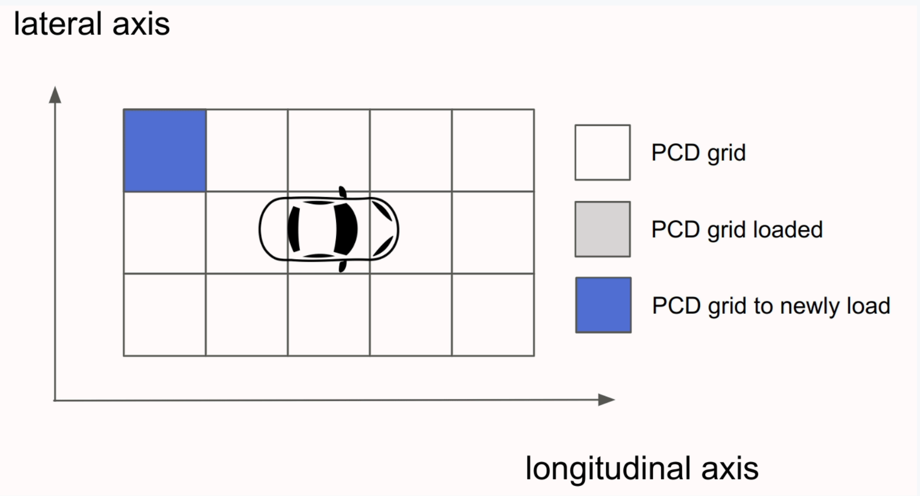


This will make it clear that it is in birds eye view.

Shin-kyoto on Apr 7, 2023 Collaborator edited

@xmfcx

Thank you for your comment! I updated fig.


lateral axis
PCD grid
PCD grid loaded
PCD grid to newly load
longitudinal axis