AUVE Practical Session :
ICP and Occupancy Grid Mapping

Fall Semester

1 Content of this lab

The goal of this lab is to try out ICP algorithm for localization and Occupancy Grid Mapping on real driving data. The chosen dataset is the recently-released NuScenes which contains a variety of driving scenarios (mostly in urban environment) with high-fidelity annotation. Provided along side with this subject, there are some skeleton codes laying out the necessary steps of both algorithms. Your task is to finishing these codes and evaluate the result.

1.1 `nuscenes-devkit`

To interface with NuScenes dataset, Motional (formerly known as Nutonomy) provides a Python package named `nuscenes-devkit`. This package parses a number of `.json` files defining the meta data of each partition of NuScenes dataset into the database whose structure shown in Fig.1. Data is queried from this database via token - a distinguish string and data type (e.g. sample (i.e. keyframe), sample data (i.e. sensor measurement), annotation). In short, `nuscenes-devkit` provides a search engine, to get a specific data, you need to input the type of the data you want, and the data's token.

Each scene, illustrated by the rectangle named `scene` in Fig.1, comes in the form of a Python `Dict`. Its keys and their associated values are listed below

— `name` : a string represents the name of the scene
— `description` : a `str` explaining the main events in this scene
— `log_token` : a `str`
— `nbr_samples` : an `int` - number of keyframes also refered as **samples** in this scene
— `first_sample_token` : a `str` - token of the first sample (i.e. keyframe)
— `first_sample_token` : a `str` - token of the last sample (i.e. keyframe)

Among these keys, `first_sample_token` and `last_sample_token` are the most important since they enable access to keyframes of a scene.

A keyframe or a **sample** takes place when all sensors are in sync, meaning their measurements have roughly the same timestamp. A **sample** has

— `timestamp` : a `str` represents unix timestamp when this sample took place
— `scene_token` : a `str` - token of the scene where this sample belongs to
— `next` : a `str` - token of the sample right after to this one
— `prev` : a `str` - token of the sample right before to this one
— `data` : a `Dict` - contains the token of every sensory measurement in this sample

A `sample_data` is a `Dict` represents a sensory measurement. Its contains

— `ego_pose_token` : a `str` - link to the `Dict` showing the pose in the world frame of the ego vehicle when this sensory measurement was taken

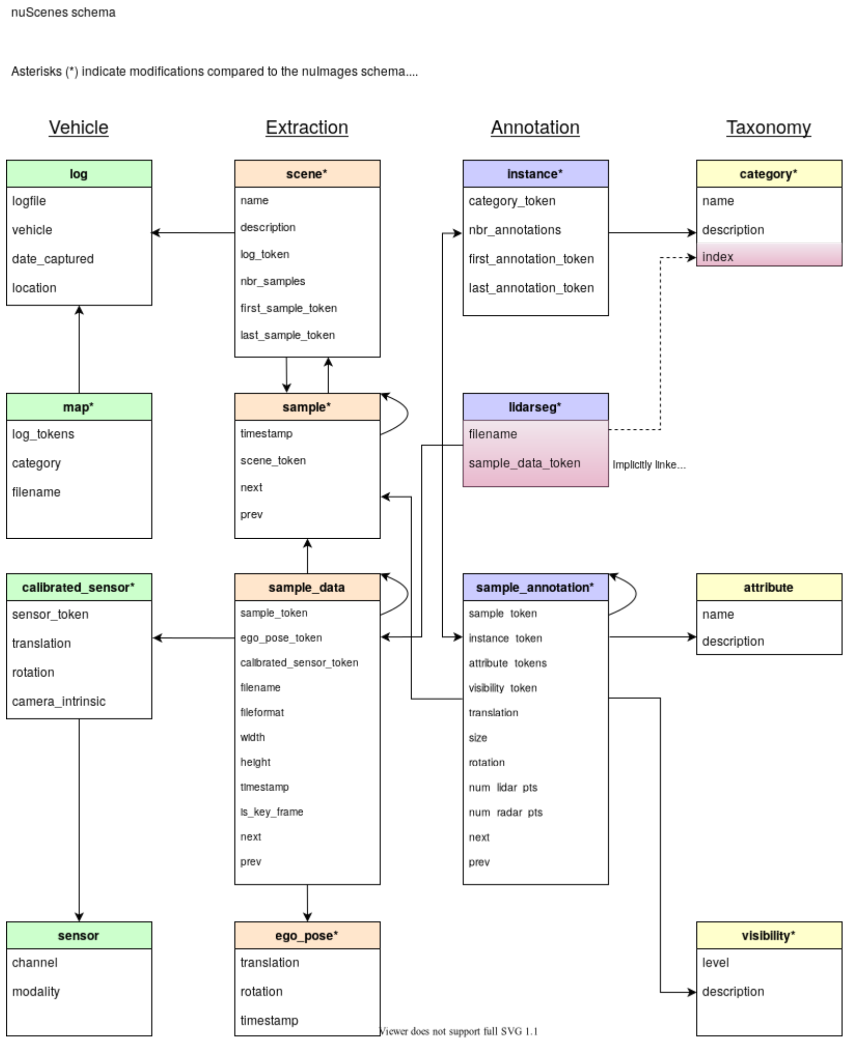

FIGURE 1 – NuScenes schema. Each named rectangle is a Python `Dict` whose keys are listed below the name. Each arrow denotes the connection from one `Dict` to another established by storing the other's token.

- `calibrated_sensor_token` : a `str` - link to the `Dict` showing the pose of the sensor in the ego vehicle's frame when this sensory measurement was taken
- `filename` : a `str` - relative path of the file containing the sensory measurement

and other less important information.

The necessary interfacing with NuScenes has been already implemented in the skeleton code. However, you will have better experience in lab if you are familiar with NuScenes. An easy way to do this is to go through this tutorial.

1.2 Iterative Closest Point with `open3d`

As also shown during the lecture, it is also possible to perform odometry (motion estimation, 2D rotation and translation) using only LiDAR pointcloud and the Iterative Closest Point (ICP) algorithm. A simple-to-use, yet efficient and accurate, implementation of ICP is provided by `open3d` in its registration pipeline. It's important to notice the naming convention of `open3d` 's registration pipeline. This pipeline takes two input a `target` cloud and a `source` cloud. Its output is homogeneous transformation (a 4x4 matrix) that maps the `source` cloud to the `target` cloud.

In this lab, the localization is perform incrementally as in Algorithm.1

Algorithme 1 : ICP in NuScenes

Result : A `list` of ego vehicle's poses in world frame during a scene
Initialize the `target` cloud by the cloud collected at the first ego vehicle's pose ;
Initialize the world frame by the first ego vehicle's pose (${}^{\texttt{world}}\mathbf{M}_{\texttt{target}} := \mathbf{I}_4$) ;
for *sample in scene* **do**
    `source` cloud := pointcloud of this sample mapped into ego vehicle's frame ;
    Invoke `open3d` 's registration pipeline with `target` cloud and `source` cloud to get ${}^{\texttt{target}}\mathbf{M}_{\texttt{src}}$;
    Compute the transformation, ${}^{\texttt{world}}\mathbf{M}_{\texttt{src}}$, from `source` frame to `world` frame ;
    Update `target` cloud with `source` cloud ;
    Update ${}^{\texttt{world}}\mathbf{M}_{\texttt{target}}$ with ${}^{\texttt{world}}\mathbf{M}_{\texttt{src}}$;
end

1.3 Occupancy Grid Mapping with Python

Using LiDAR pointcloud together with ego vehicle poses, it is possible to perform an Occupancy Grid Mapping. To do so, the grid is built based on the log-odds representation given during the lecture. An implementation of such an approach is provided in the skeleton code. You will see that in 2-D, the occupancy of a grid cell using range sensors is achieved using the Bresenham algorithm [1].

2 Expected work

This lab is to help you have the first experience working with a real driving dataset, in the meantime trying out two popular algorithms concerning pointcloud. For this purpose, the majority of work has been done in the skeleton code. Your task is to fill in the lines marked by **TODO** in

— `nuscenes_icp_2.py`
— `occupancy_main.py`
— `occupancy_grid.py`

to complete ICP and Occupancy Grid Mapping.

1. http://code.activestate.com/recipes/578112-bresenhams-line-algorithm-in-n-dimensions/