LIDAR BASED SLAM

Zhiyong Zhang - 001833676

Venkat Prasad Krishamurthy - 001890724

Tejas Krishna Reddy - 001423166

DATASET: Car_IR_RGB_Lidar

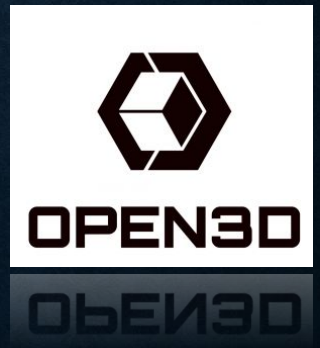
Sensors:

- 2 Velodyne VLP-16 LIDAR's
  - MSG TYPES:
    - sensor_msgs/PointCloud2
  - TOPICS:
    - /ns1/velodyne_points - Left LIDAR
    - /ns2/velodyne_points - Right LIDAR
- VN-100 IMU
  - MSG TYPES:
    - sensor_msgs/Imu
  - TOPIC
    - /imu/imu
- GPS
  - MSG TYPES:
    - sensor_msgs/NavSatFix
  - TOPIC
    - /vehicle/gps/fix

- OTHER SENSORS:
  - RGB CAMERAS
  - IR CAMERA

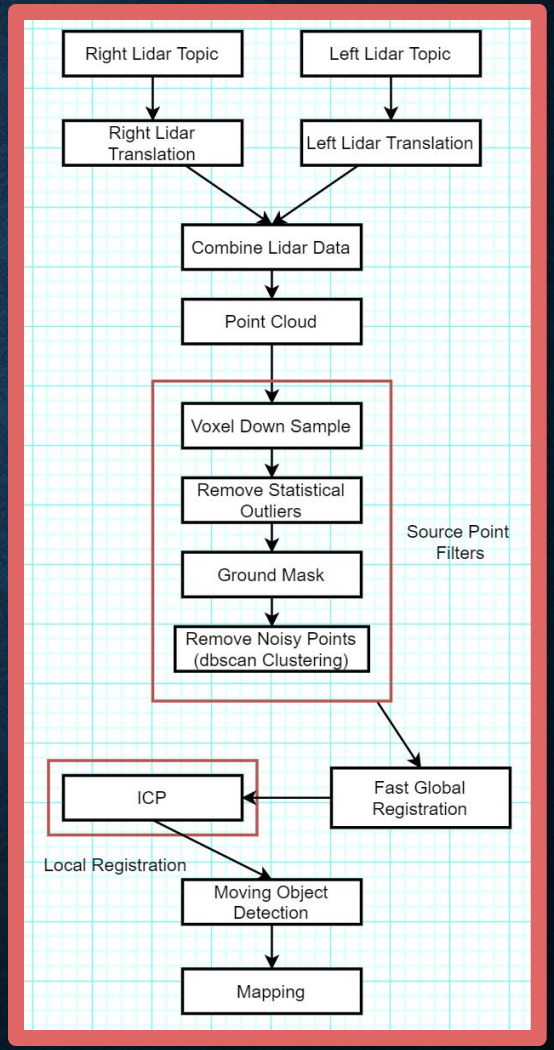
SOFTWARE REQUIREMENTS

- Libraries Used:
  - numpy - For matrix representation
  - utm - For converting gps coordinates
    - utm_tf
  - matplotlib - for plotting
    - pyplot
  - mpl_toolkits - for plotting 3D graphs
    - Axes3D
    - Poly3DCollection
  - open3d - for 3D data processing and visualization
    - fast global registration
    - icp registration
    - get point cloud fpfh (Fast point Feature Histogram)
- Environment:
  - ROS



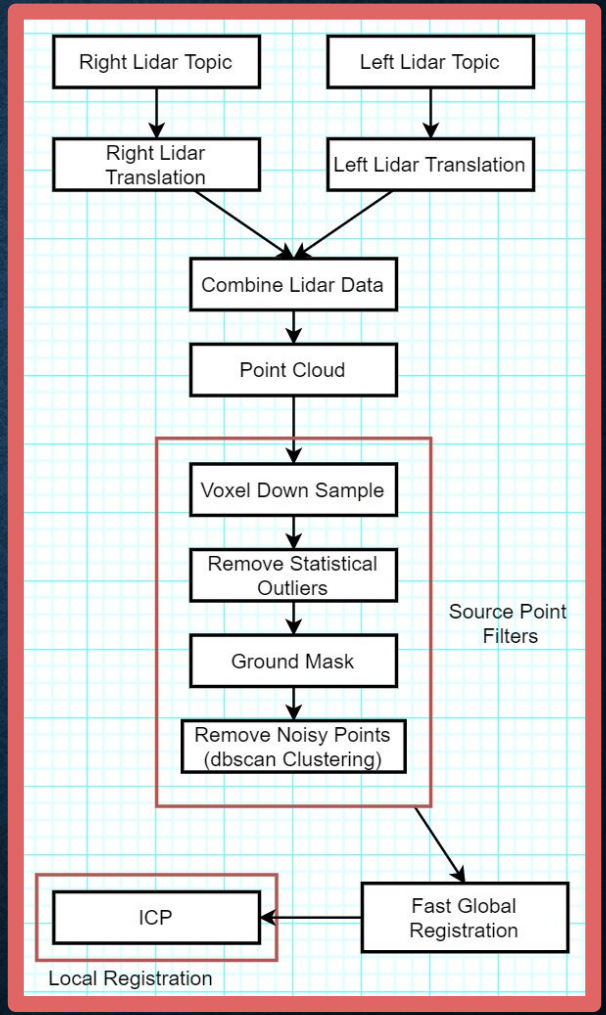
Functions implemented:

- Sensor axis alignment
- Filtering
- Feature Extraction
- Point Cloud Registration
- Localization
- Moving object detection
- Mapping



Selecting Source Points:

- Reducing raw sample points by downsample each voxel. All the points in that voxel are represented as its gravity center.
- Filters Applied:
  - Voxel Down Sampling
  - Statistical Outlier Removal
  - Ground Points Removal
  - Cluster to Remove Noisy Points and Car Itself



- Voxel Down Sample:
  - An Voxel is an array representing a value in 3D space.
  - Image is divided into several voxels.
  - From each voxel a single point is extracted.

- Statistical Outlier Removal:
  - Removes points further away from its neighbours.
  - A distance measure is calculated from its neighbours for each point.
  - A threshold is set on the average distances based on Standard deviation to identify outliers.

- Ground Point Removal:
  - The lidars record data of the road/ground around the car.
  - We implement our method to accomplish this.

- Dbscan Clustering:
  - Noisy points that don't belong to any cluster were removed in this step.
  - Car body cluster points were also removed.

Ground Point Removal

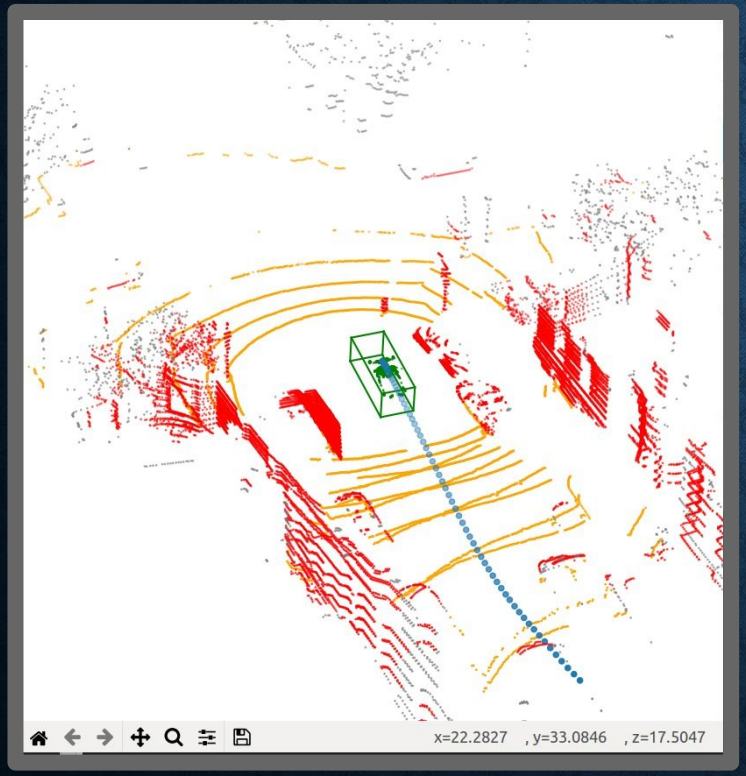
Assumptions:

- The inclination of vehicle and ground is within 2 degrees
- Road height difference is within 0.3 meters

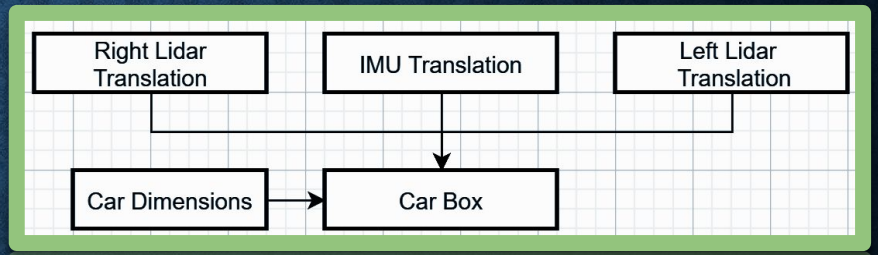
Algorithm:

- Get the plane parallel to the car
- Filter out all the points within 0.3 meters of the plane
- Randomly select 3 points from them and form a ground plane
- Check the angle between the above two planes is less than 2 degrees
- Count how many points is close to the plane (threshold = 0.3)
- Save the plane covers most points as the best result
- Iterate for certain times

Post-Filtering Results:



Car Is Visualised By The Following Process:



POINT CLOUD REGISTRATION

- Point Cloud Registration is the process of finding a spatial transformation that aligns two point cloud
- Point Cloud Registration steps:
  - Fast Global Registration - (Global Registration)
  - Iterative Closest Point Registration - (Local Registration)
- Global Registration
  - Global Registration does not require an alignment for initialization
  - It produces less tight alignment which can be used as an input to ICP algorithm
- Fast Global Registration
  - It is faster than Global registration
  - There is no model proposal and evaluation involved for each iteration

POINT CLOUD REGISTRATION

- Iterative Closest Point -
  - Algorithm:
    - Find Correspondence set K = {(p,q)} from target point cloud P, and source point cloud Q transformed with current transformation matrix T
    - Update the transformation T by minimizing an objective function E(T) defined over the correspondence set K
  - Types:
    - Point-to-Point

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \|\mathbf{p} - \mathbf{T}\mathbf{q}\|^2.$$

    - Point-to-Plane

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \left((\mathbf{p} - \mathbf{T}\mathbf{q}) \cdot \mathbf{n}_\mathbf{p}\right)^2$$

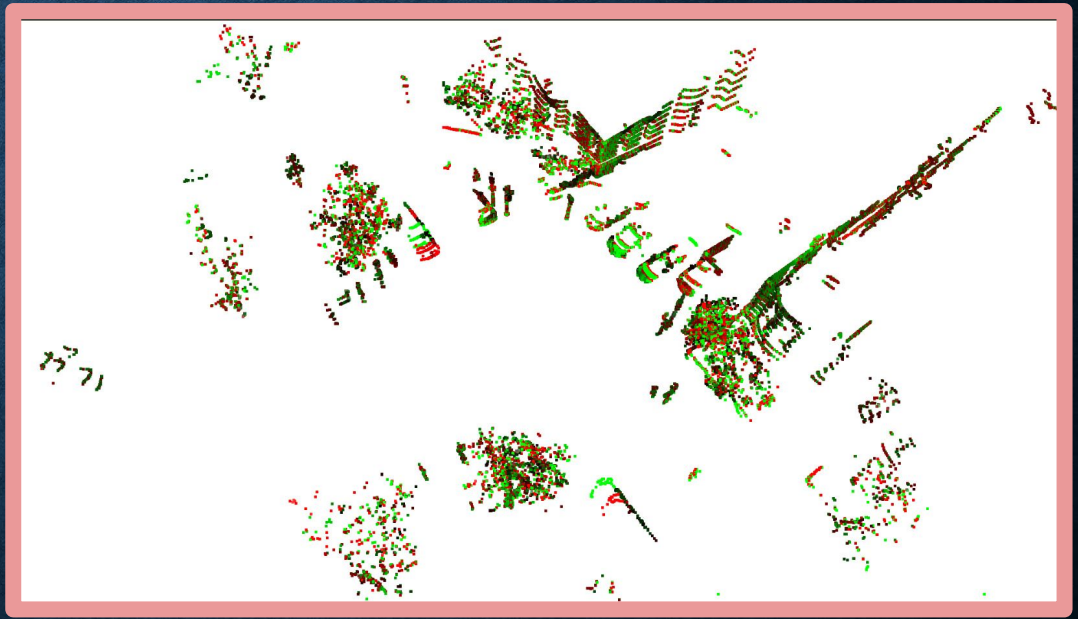
      np is the normal of point p

POINT CLOUD REGISTRATION

Red points are transformed source point cloud (previous frame)

Green points are target point cloud (Current frame)

Black points are the result of perfect registration of red and green points (red+green=black)

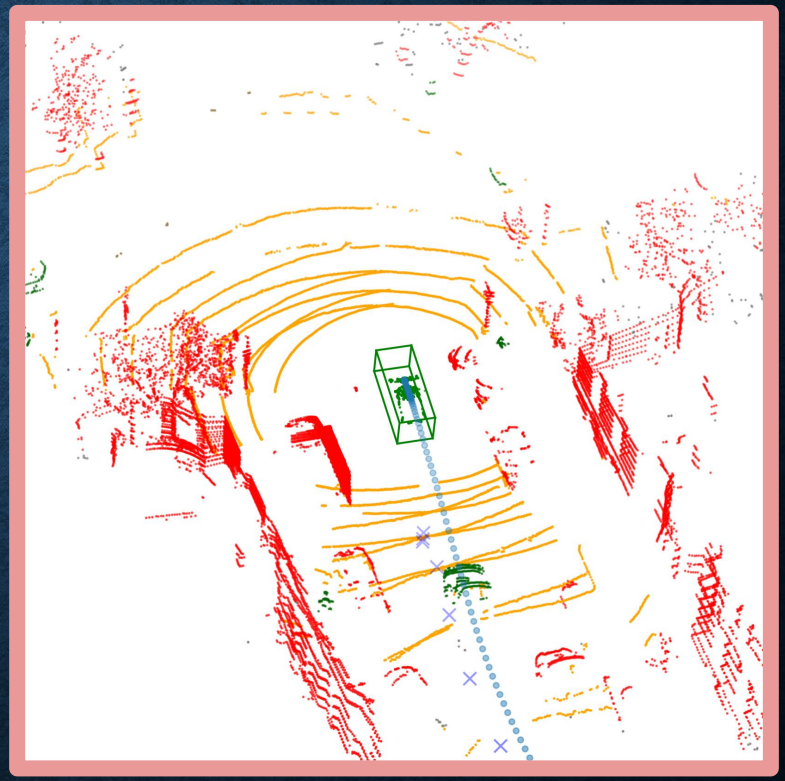


Localization

Blue points are estimated trajectory result of point cloud registration

Blue crosses are UTM points

The estimated trajectory is drifting over time

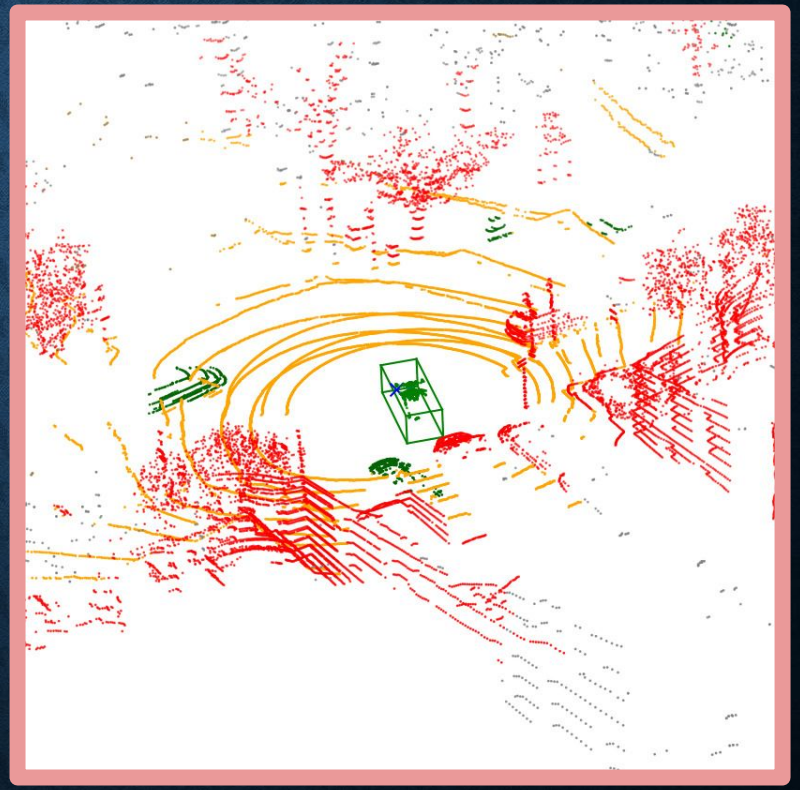


Moving Objects Recognition (Our implementation)

- After Local Registration, each frame was divided into segments by clustering algorithm. We find all the points of each cluster in the current frame, and the corresponding points of the previous frame.

- Check the mean distance of the corresponding points, and the proportion of corresponding points in a cluster.

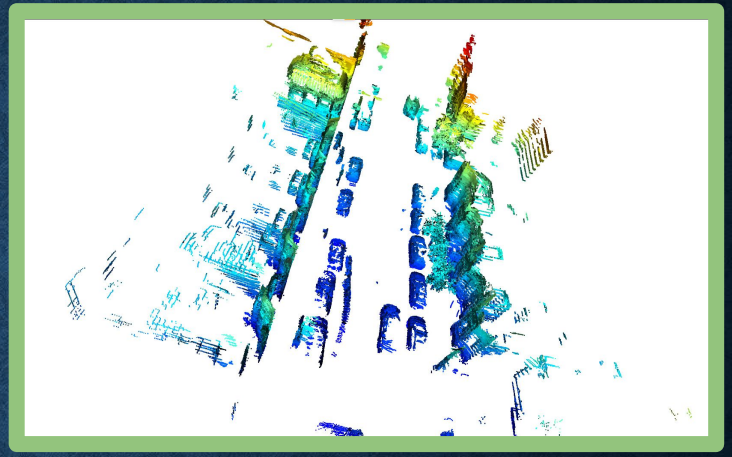
- The moving objects can be seen in green color in the figure.

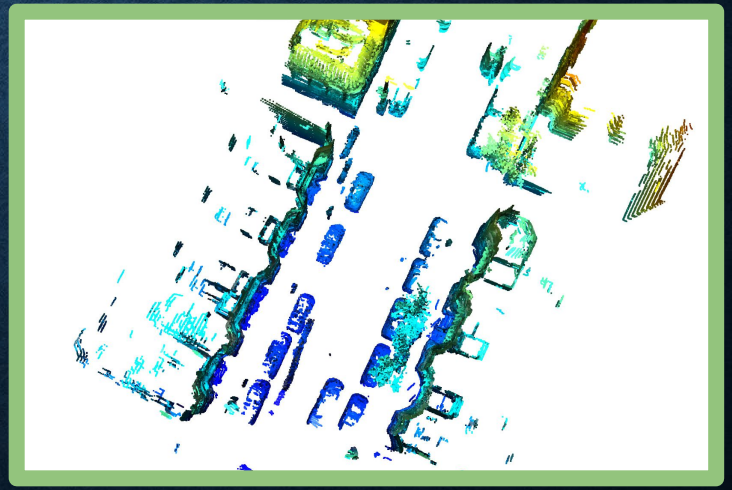


Mapping:

The transformation between each two consecutive frame is stored in a list, so that we can transform all the frames to the first frame coordination system. And the first frame is transformed to NED by imu pose estimation (quaternion to ypr) at that moment.

Put all the points together and downsample them to get final result.





Future Work:

If we can calibrate the IMU, yaw could be used for data fusion with point cloud registration.

Lidar has distortion, when car is driving, The first scanline does not match the last. If acceleration data is available, the distortion could be corrected.

We do not know which points belong to the same scanline. If this is available by reading raw data from lidar, we can have a much better ground points detection based on the angle difference of each scan line from bottom to top.

Open3d does not support viewpoint feature descriptor, which would be helpful for moving object detection. Maybe we can do ICP again after moving object detection?

Code is in Python right now, each frame take 1-2 seconds. Replace with C++ for real time.

Thank You!!!

Demo & Questions?