EE405A
Mapping & Localization

(TA) Daegyu Lee
School of Electrical Engineering
KAIST

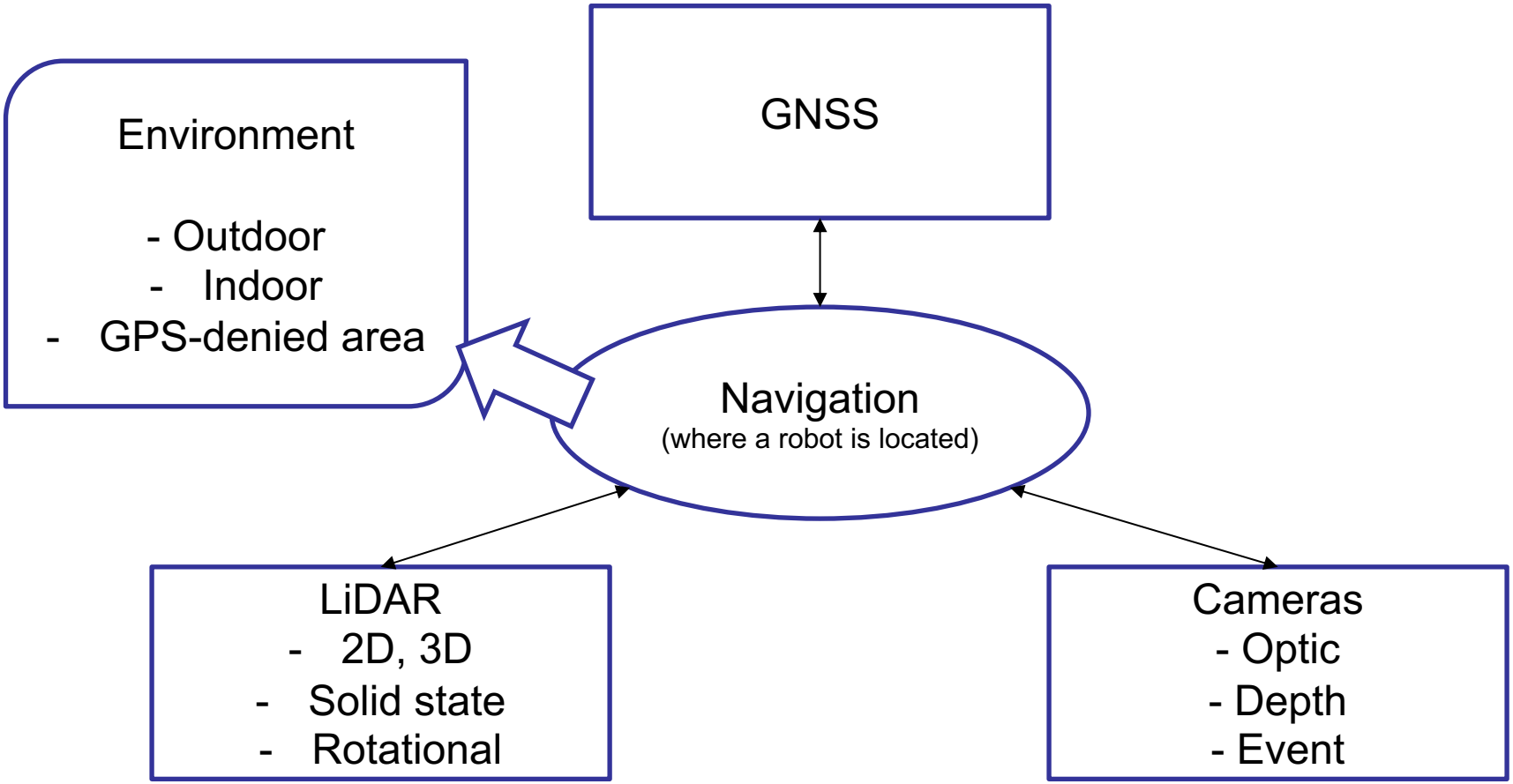
Oct 6, 2022

lee.dk@kaist.ac.kr

Introduction

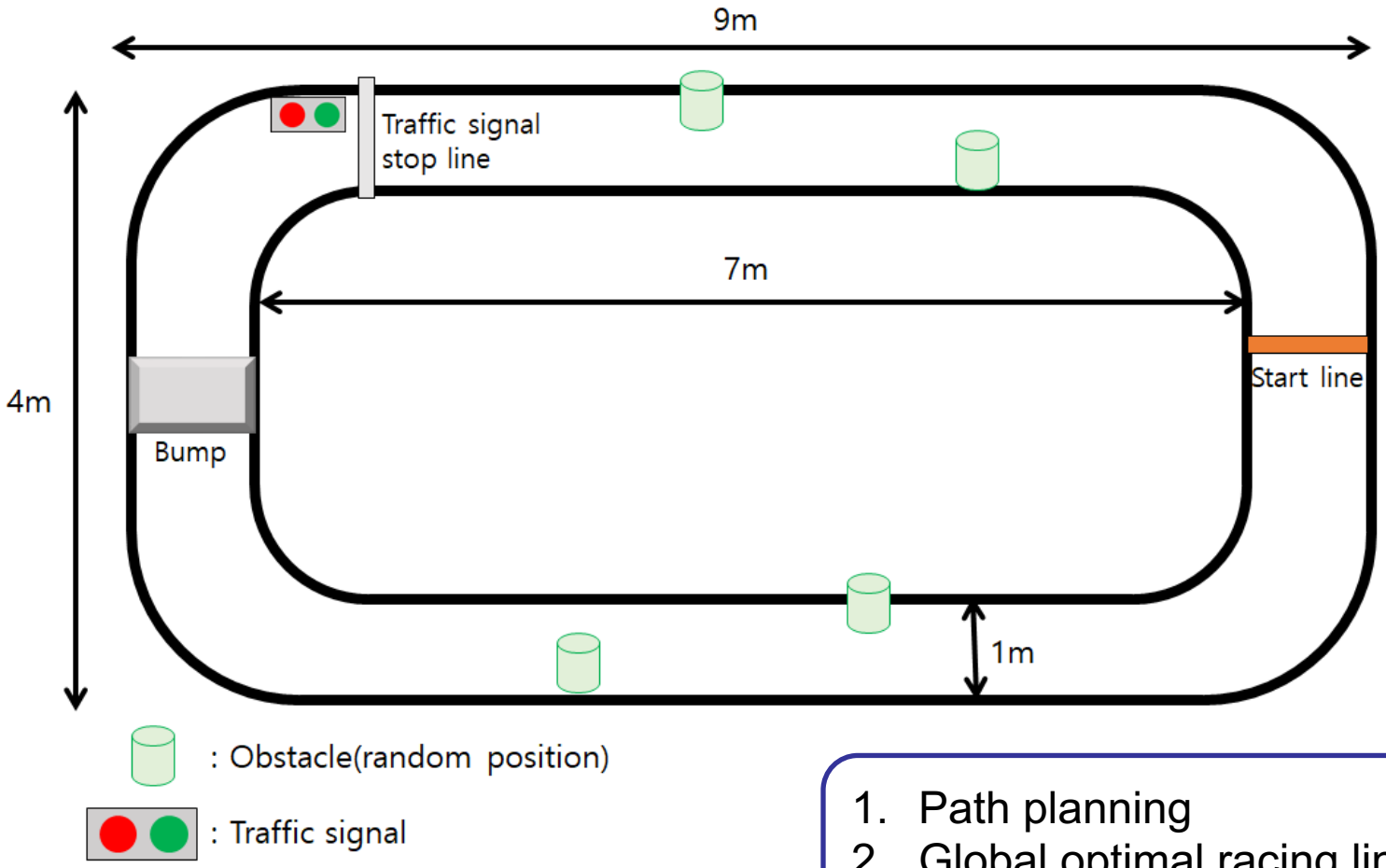
Introduction

- **Sensors for navigation**



Introduction

- **What is the purpose of the navigation? More precisely, localization?**

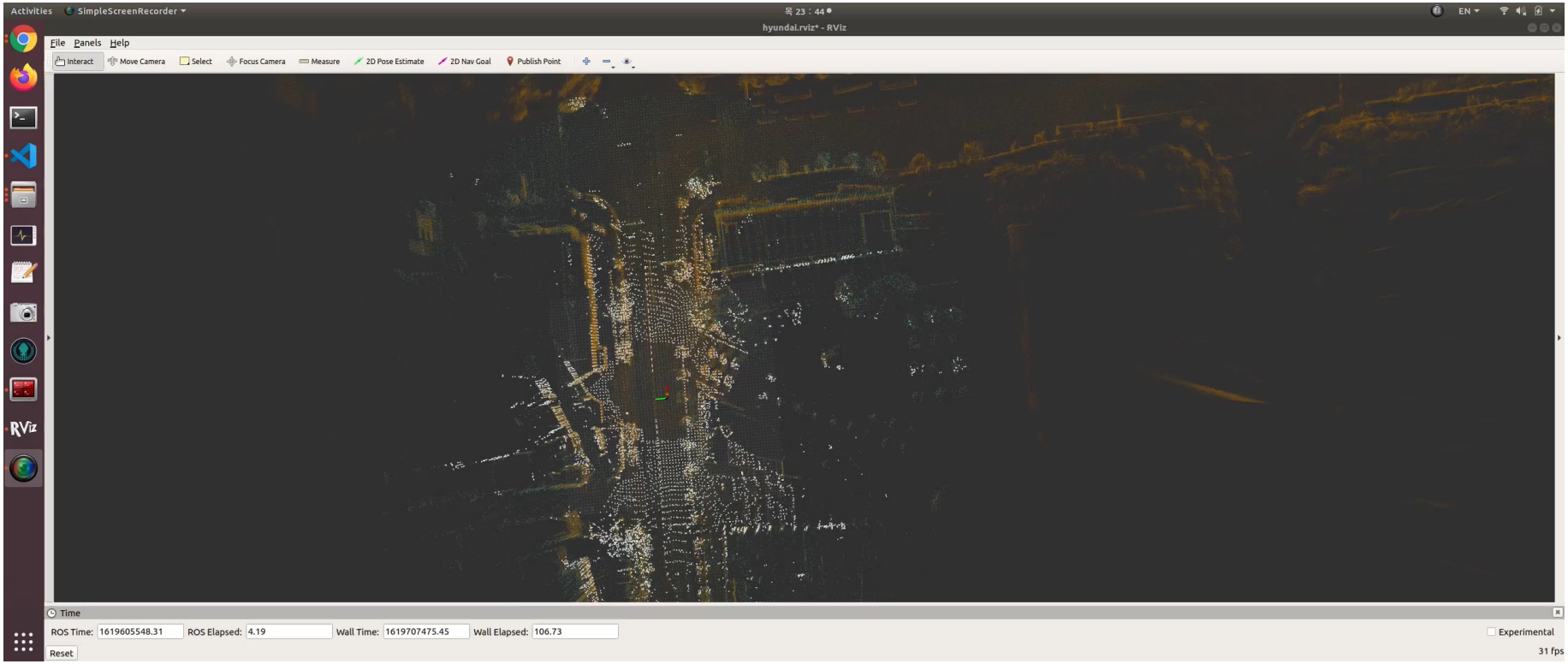


1. Path planning
2. Global optimal racing line
3. Behavior planning

Introduction

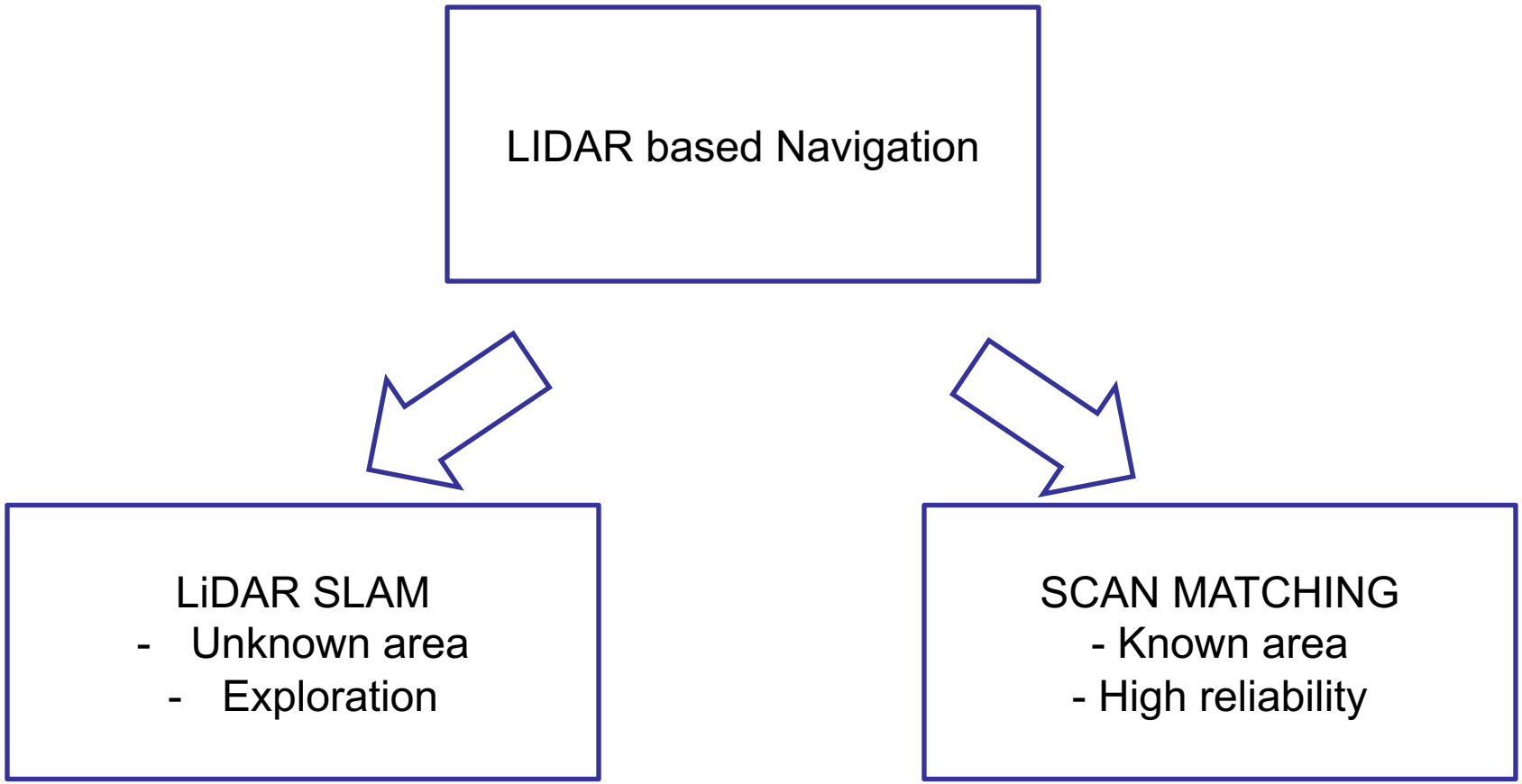
➢ Use cases

- ❑ **Urban environment(Metropolitan, High-building area …)**
- ❑ **GPS-denied environment**



Introduction

- **Use cases**



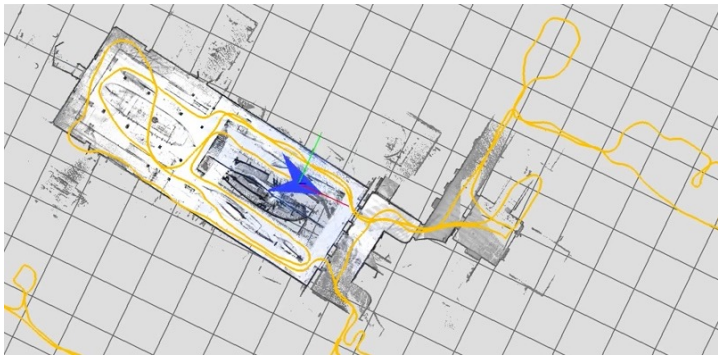
SLAM basics

SLAM basics

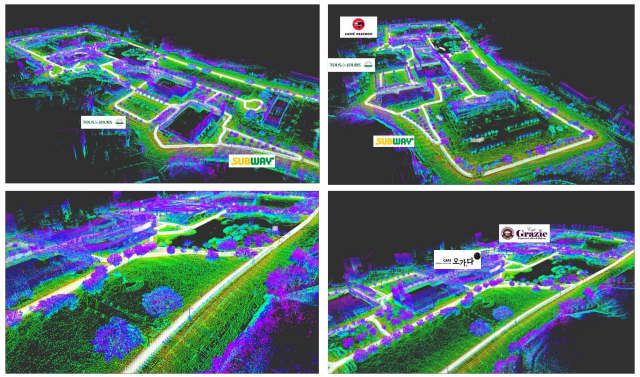
- **What is the SLAM algorithm?**
  - SLAM : Simultaneous Localization and Mapping
  - Localization

    : Robot calculates its pose in the map from initial point(Normally it starts from origin (0, 0).)
  - Mapping

    : Robot accumulates extracted features using a couple of sensors(Vision, LiDAR, Radar ..)
- **How to utilize the SLAM algorithm**
  - Generate a map and save it to implement scan matching algorithm
  - Estimate accurate robot odometry(exploration, laser-based odometry, visual odometry)



2D laser SLAM
(Google Cartographer)



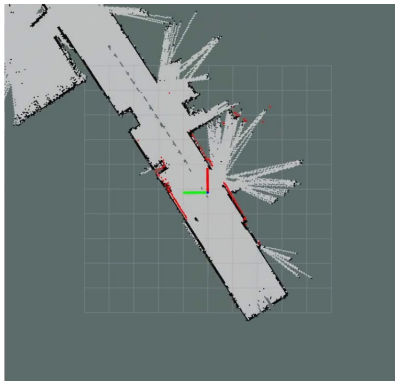
3D laser SLAM – KAIST east side
(LeGO-LOAM)

SLAM basics

➢ 2D LiDAR Mapping(ROS)

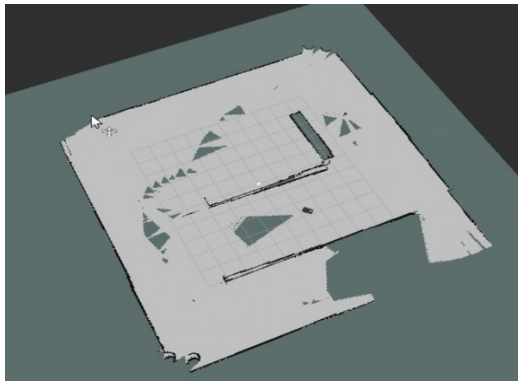
- ❑ A couple of 2D laser SLAM algorithm are provided in the ROS
- ❑ You should consider the input data to implement each algorithm

	Laser (sensor_msgs/LaserScan)	Odometry (nav_msgs/Odometry)	IMU (sensor_msgs/Imu)
hector SLAM	O	X	X
gmapping	O	O	X
Google cartographer	O	X	selective



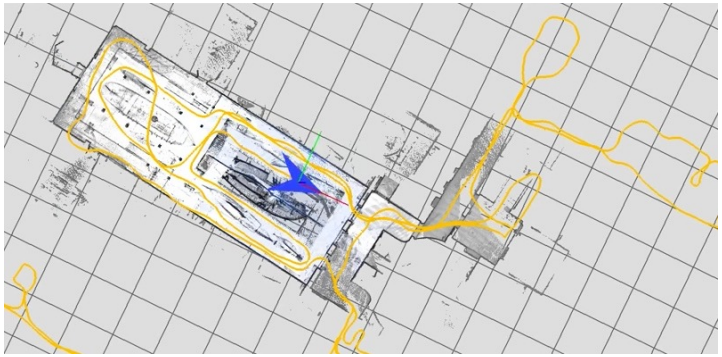
hector SLAM

http://wiki.ros.org/hector_slam



gmapping

http://wiki.ros.org/gmapping



2D laser SLAM
(Google Cartographer)

https://google-cartographer-ros.readthedocs.io/en/latest/

SLAM basics

➢ Occupancy Grid Map(ROS)

- ❑ A 2-D grid map, in which each cell represents the probability of occupancy
- ❑ The result of 2D LiDAR mapping is provided by occupancy grid map (nav_msgs/OccupancyGrid)
- ❑ You can save the result of mapping with map_saver command.

2.2 map_saver

map_saver saves a map to disk, e.g., from a SLAM mapping service.

2.2.1 Usage

```
rosrun map_server map_saver [--occ <threshold_occupied>] [--free <threshold_free>] [-f <mapname
>] map:=/your/costmap/topic
```

map_saver retrieves map data and writes it out to **map.pgm** and **map.yaml**. Use the **-f** option to provide a different base name for the output files. The **--occ** and **--free** options take values between 0 and 100. To save different map topic set **map** to your costmap topic.

2.2.2 Examples

```
rosrun map_server map_saver -f mymap
```

```
rosrun map_server map_saver --occ 90 --free 10 -f mymap map:=/move_base/global_costmap/costmap
```

2.2.3 Subscribed Topics

map (nav_msgs/OccupancyGrid)
Map will be retrieved via this latched topic.

http://wiki.ros.org/map_server

SLAM basics

➢ Occupancy Grid Map(Cont')

- ❑ You can read a map from disk with map_server node.

2.1 map_server

map_server is a ROS node that reads a map from disk and offers it via a ROS service.
The current implementation of the map_server converts color values in the map image data into ternary occupancy values: free (0), occupied (100), and unknown (-1). Future versions of this tool may use the values between 0 and 100 to communicate finer gradations of occupancy.

2.1.1 Usage

```
map_server <map.yaml>
```

2.1.2 Example

```
rosrun map_server map_server mymap.yaml
```

Note that the map data may be retrieved via either latched topic (meaning that it is sent once to each new subscriber), or via service. The service may eventually be phased out.

2.1.3 Published Topics

map_metadata (nav_msgs/MapMetaData)
Receive the map metadata via this latched topic.

map (nav_msgs/OccupancyGrid)
Receive the map via this latched topic.

2.1.4 Services

static_map (nav_msgs/GetMap)
Retrieve the map via this service.

2.1.5 Parameters

~frame_id (string, default: "map")
The frame to set in the header of the published map.

http://wiki.ros.org/map_server

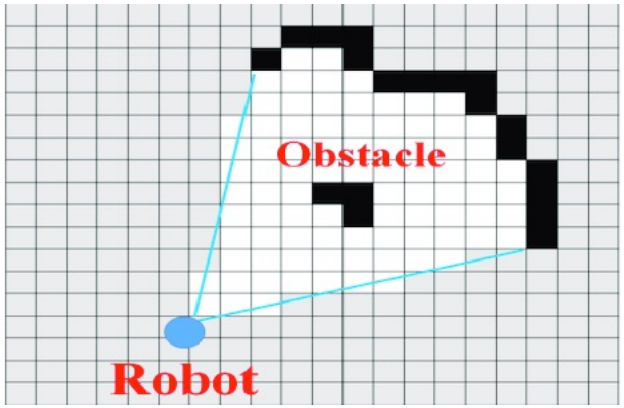
SLAM basics

➢ Occupancy Grid Map(Cont')

https://blog.bthere.ai/viewing-occupancy-grids-from-anywhere-in-the-world/

- The message of occupancy grid map is composed of these:

  - std_msgs/Header header
  - MapMetaData info
    - time map_load_time
    - float32 resolution
    - uint32 width
    - uint32 height
    - geometry_msgs/Pose
  - int8[] data



- Occupancy Grid Map is represented as image which contains resolution, width, height
- If you want to obtain the (x,y) values of the occupied grid such as obstacles, wall, barrier, you should convert gridmap using resolution, width, height and its pose.
- Example code link :

  https://www.dropbox.com/s/3rrkxdtpln6kvoa/hyundai_laser_collision_checker.zip?dl=0

  Please see the line 189 ~ 209 codes.

  If you see the for loop in the code, I changed occupancy grid data to the X,Y values.

SLAM basics

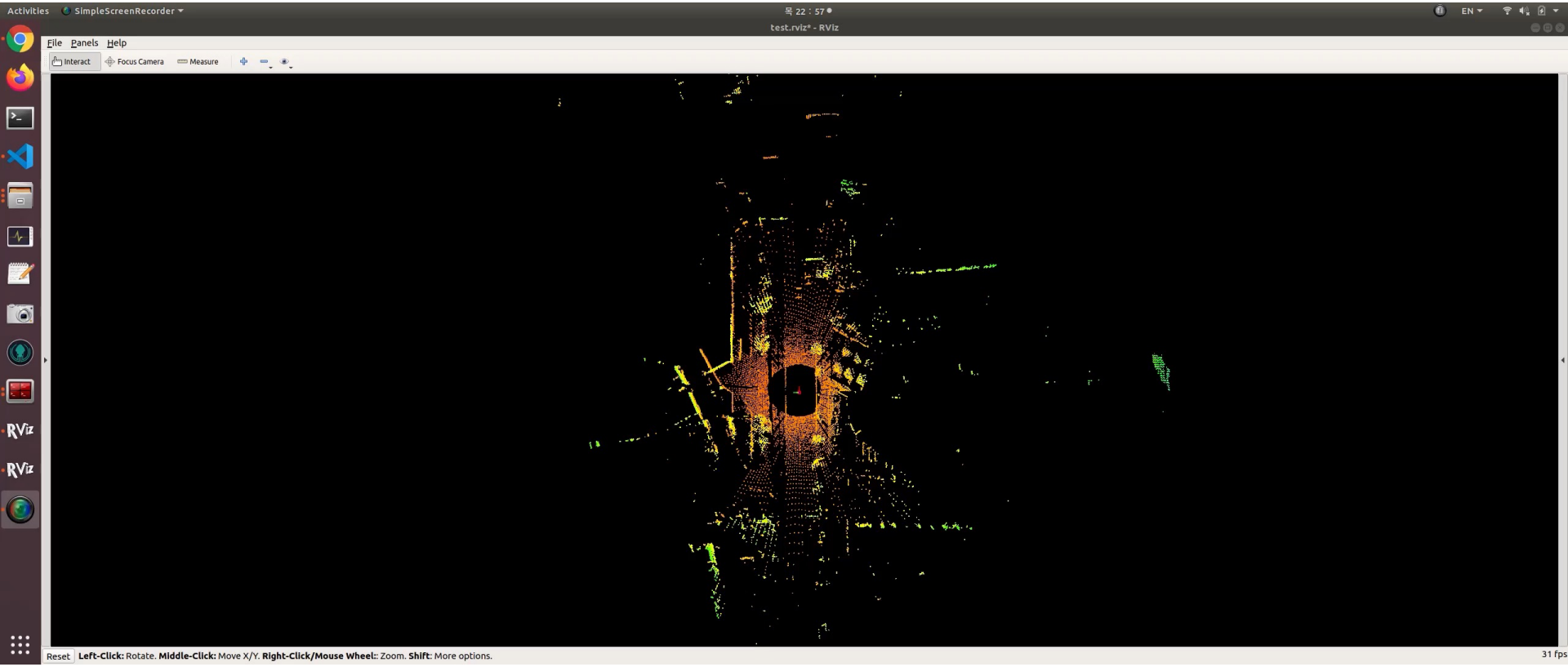
➢ 3D LiDAR Mapping

- ❑ A couple of 3D laser SLAM algorithm are provided in the ROS
  - ✓ LeGO-LOAM
    - https://github.com/RobustFieldAutonomyLab/LeGO-LOAM
    - Shan, Tixiao, and Brendan Englot. "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain." *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018.
  - ✓ LIO-SAM
    - https://github.com/TixiaoShan/LIO-SAM
    - Shan, Tixiao, et al. "Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping." *arXiv preprint arXiv:2007.00258* (2020).
- ❑ 3D laser scanner is required for all the 3D mapping algorithms.

  Lidar scanner : Velodyne, Ouster, Hesai, Sick ..

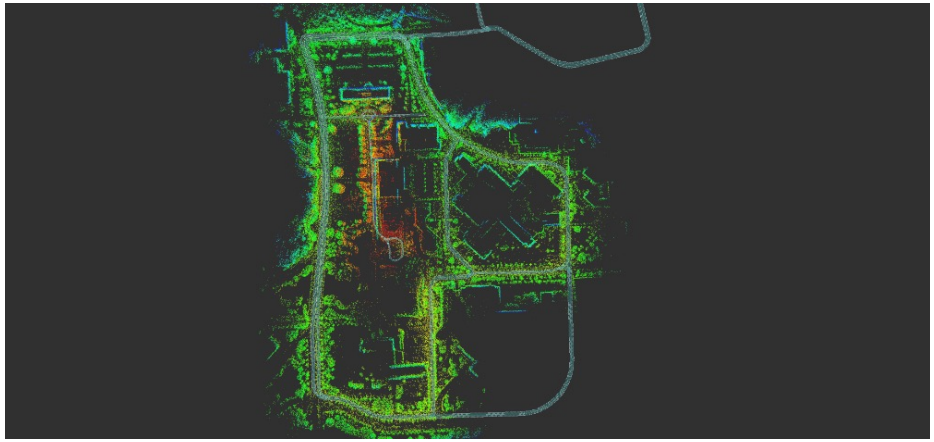
SLAM basics

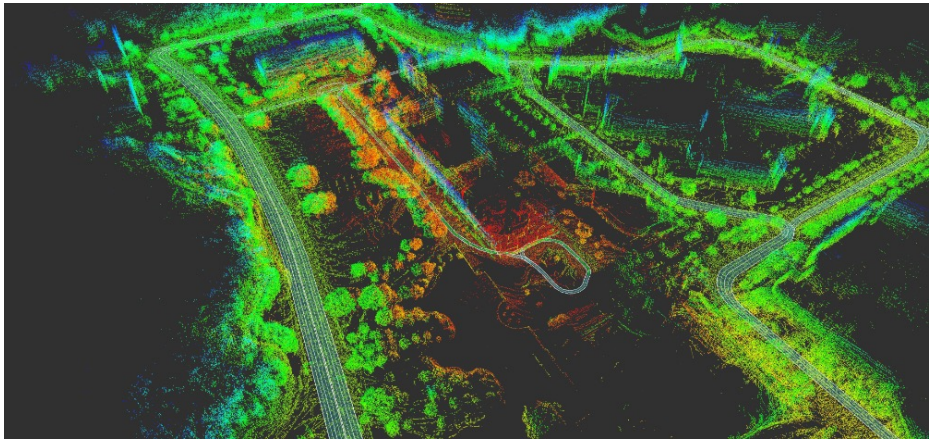
- Map building(lego-loam , LIO-SAM …)
  - Implement the open source map builder



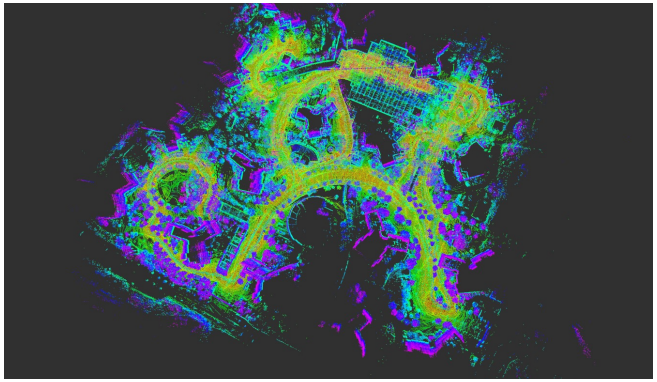
SLAM basics

- **3D LiDAR Mapping in the KAIST east area(with 2D road map)**



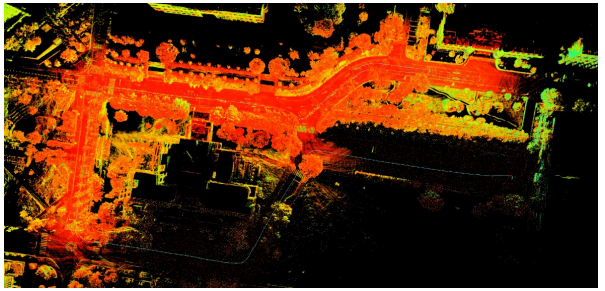


- **3D LiDAR Mapping for our lab project**



Ilsan apartment

Chunju apartment



Korea Univ.(Sejong)

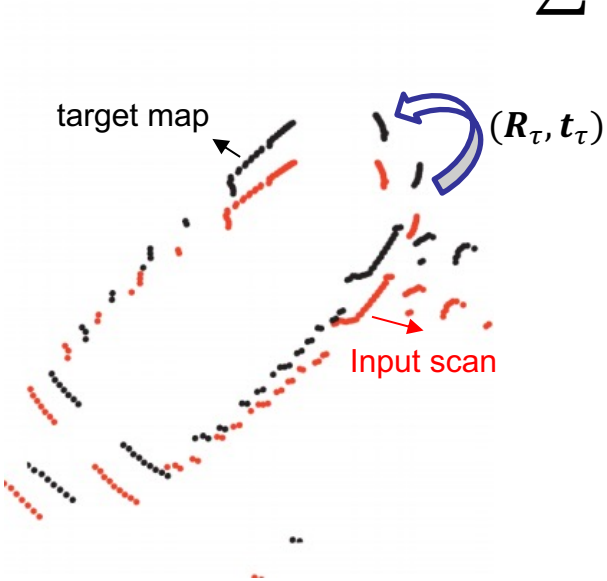
Scan Matching Algorithm

Scan Matching Algorithm

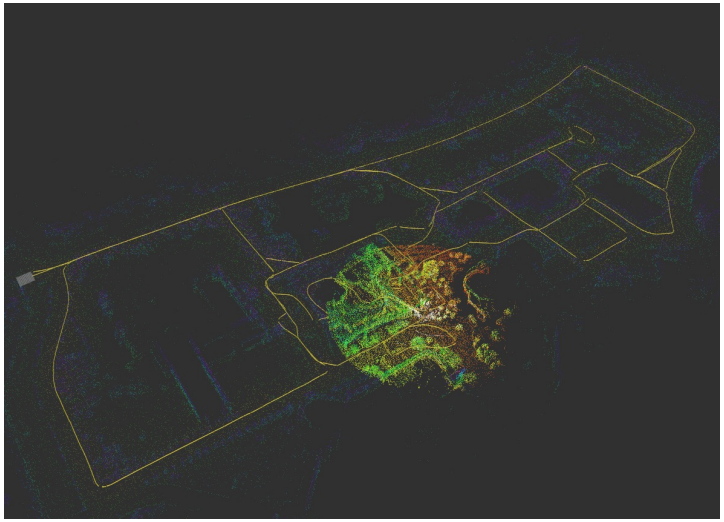
- **What is the scan matching algorithm?**
  - Scan : sequence of scan points representing the contour curve of the local environment

    Scan point(p_i): represented with polar coordinates
  - Search the translation($\boldsymbol{t}_\tau$) and rotation($\boldsymbol{R}_\tau$) (rigid transform) which minimize a distance error between the incoming scan points($\boldsymbol{p}_\tau = \sum p_i$) and pre-built map($\boldsymbol{M}$)

$$minimize \sum [(\boldsymbol{R}_\tau p_i + \boldsymbol{t}_\tau) - \boldsymbol{M}]$$



Source: Courtesy of Noel Welsh

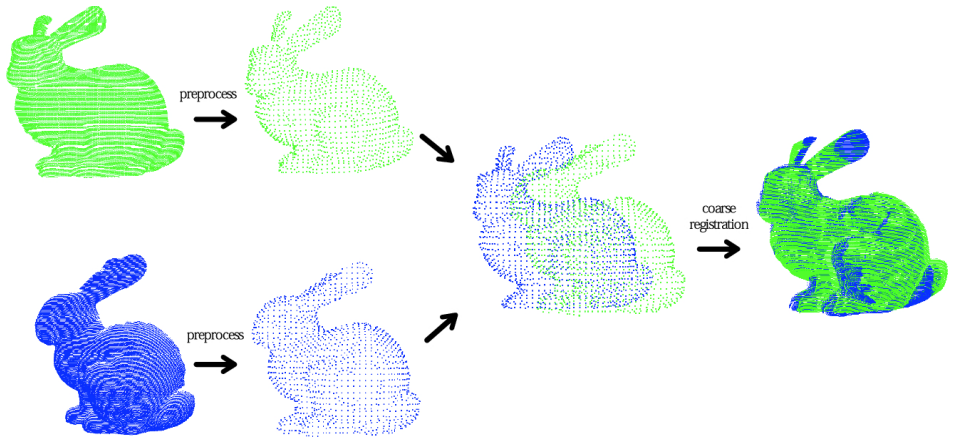


Result of scan matching algorithm at the KAIST east area

Scan Matching Algorithm

➢ Aligning Point Clouds

- ❑ There are a series of algorithms to align the point clouds with the translation($\boldsymbol{t}_\tau$) and rotation($\boldsymbol{R}_\tau$) between the incoming scan points($\boldsymbol{p}_\tau = \sum p_i$) and pre-built map($\boldsymbol{M}$)
- ❑ ICP : Iterative Closest Point algorithm
- ❑ NDT : Normal Distributions Transformation algorithm
- ❑ You can implement registration algorithm(aligning point cloud algorithm) using PCL(Point cloud library) : https://pcl.readthedocs.io/projects/tutorials/en/latest/index.html#registration



Feng X., Tan T., Yuan Y., Yin C. (2019) Aligning Point Clouds with an Effective Local Feature Descriptor. In: Ning H. (eds) Cyberspace Data and Intelligence, and Cyber-Living, Syndrome, and Health. CyberDI 2019, CyberLife 2019. Communications in Computer and Information Science, vol 1138. Springer, Singapore. https://doi.org/10.1007/978-981-15-1925-3_18

Construct a pre-built map

➢ Our Goal

- ❑ Search the translation($\boldsymbol{t}_\tau$) and rotation($\boldsymbol{R}_\tau$) (rigid transform) which minimize a distance error between the incoming scan points($\boldsymbol{p}_\tau = \sum p_i$) and pre-built map($\boldsymbol{M}$)

$$minimize\ \sum[(\boldsymbol{R}_\tau p_i + \boldsymbol{t}_\tau) - \boldsymbol{M}]$$

➢ Hector SLAM

- ❑ One of the proper SLAM algorithms using 2D LiDAR only without odometry as well as on platforms that exhibit roll/pitch motion (of the sensor, the platform or both).
- ❑ Hector SLAM leverages the **high update rate of modern LIDAR systems** (like the Hokuyo UTM-20LX, which we will use in this lecture).
- ❑ And it provides **2D pose estimates** at scan rate of the sensors (up to 40Hz).
- ❑ Installation: http://wiki.ros.org/hector_slam
- ❑ Github: https://github.com/tu-darmstadt-ros-pkg/hector_slam
- ❑ Tutorial: http://wiki.ros.org/hector_slam/Tutorials

Construct a pre-built map

➢ Hector SLAM

- ❑ Three packages are in Hector SLAM
  - ✓ **hector_mapping:** The SLAM node.
  - ✓ **hector_geotiff:** Saving of map and robot trajectory to geotiff images files.
  - ✓ **hector_trajectory_server:** Saving of tf based trajectories.
- ❑ hector_mapping
  - ✓ Subsrcibed Topics:
    - **scan (sensor_msgs/LaserScan)**

      : The laser scan used by the SLAM system.
    - **syscommand (std_msgs/String)**

      : System command (to save, reset map.
  - ✓ Published Topics:
    - **map (nav_msgs/OccupancyGrid)**

      : Get the map data from this topic, which is latched, and updated periodically.
    - **slam_output_pose (geometry_msgs/PoseStamped)**

      : The estimated robot pose without covariance**.**

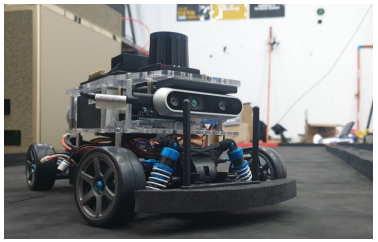
Reference: http://wiki.ros.org/hector_mapping

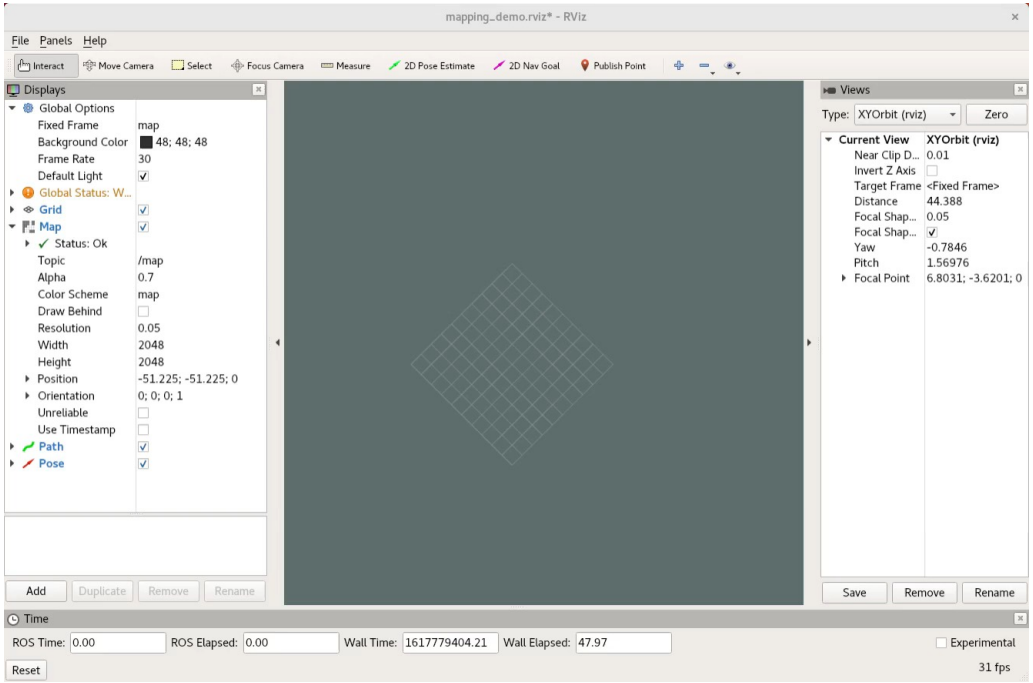
Construct a pre-built map

➢ Hector SLAM

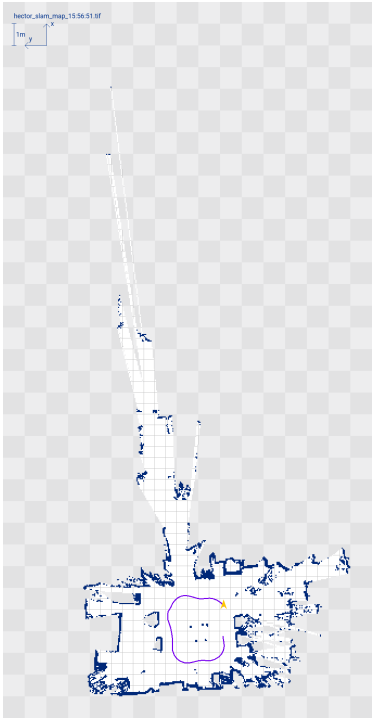
- ❑ Try to mapping a place in our lab using Hector SLAM (Not Homework!).
  - ✓ A rosbag file recorded from the RC platform which you will build later.
  - ✓ The generated map can be used to navigate your robot in the place.

example rosbag file : https://www.dropbox.com/s/zejwxifkaznihbe/2021-04-07-15-20-20.bag?dl=0





Building a map using Hector SLAM



Generated .tif map file

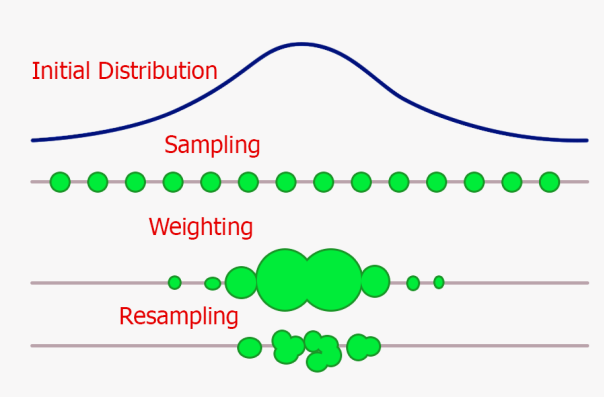
Implement 2D Localization based on Particle filter

Implement 2D Localization based on Particle filter

➢ Particle Filter-based Localization

❑ Particle filter algorithm [1]

- ✓ Particle filter is an algorithm to estimate a probabilistic distribution by sampling particles and importance weighting. (i.e., the probability that a robot is located in a specific point can be represented as the distribution.)
- ✓ It is the iteration of sampling the particles (Sampling), calculating each particle's score (Importance weighting), and resampling the particles considering the scores (Resampling).
- ✓ Since it requires expensive computation to calculate the score, the real-time localization with particle filter is hard to implement; however, the GPU can handle this problem with parallel processing.



Particle_filter$(\mathcal{X}_{t-1}, u_t, z_t)$:

1: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2: *for* $m = 1$ *to* M *do*
3: *sample* $x_t^{[m]} \sim \pi(x_t)$ — **Sampling**
4: $w_t^{[m]} = \frac{p(x_t^{[m]})}{\pi(x_t^{[m]})}$ — **Importance weighting**
5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6: *endfor*
7: *for* $m = 1$ *to* M *do*
8: *draw* i *with probability* $\propto w_t^{[i]}$
9: *add* $x_t^{[i]}$ *to* $\mathcal{X}_t$ — **Resampling**
10: *endfor*
11: *return* $\mathcal{X}_t$

[1] Robot mapping, University of Freiburg

Implement 2D Localization based on Particle filter

➢ Particle Filter-based Localization

❑ Odometry

- ✓ Odometry is an estimation of a robot position using motion sensor measurements (i.e., wheel odometer, rotary encoder).
- ✓ In the case of car-like model, odometry can be derived based on a vehicle model (learned in Week 5 Vehicle Control).
- ✓ In the absence of wheel odometry, the robot position (x, y, yaw) can be estimated with the vehicle speed (integral of the acceleration from IMU) and steering angle of the ego vehicle.

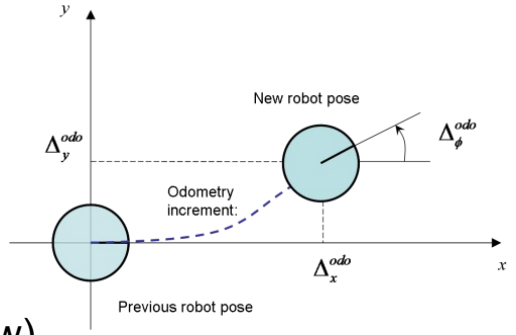
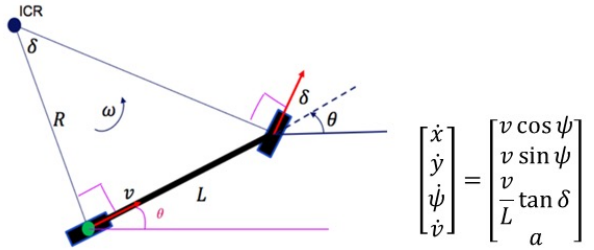


Figure for a robot motion

❑ References for odometry

- ✓ ROS odometry publisher: http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom
- ✓ Odometry for car-like model: https://medium.com/@waleedmansoor/how-i-built-ros-odometry-for-differential-drive-vehicle-without-encoder-c9f73fe63d87

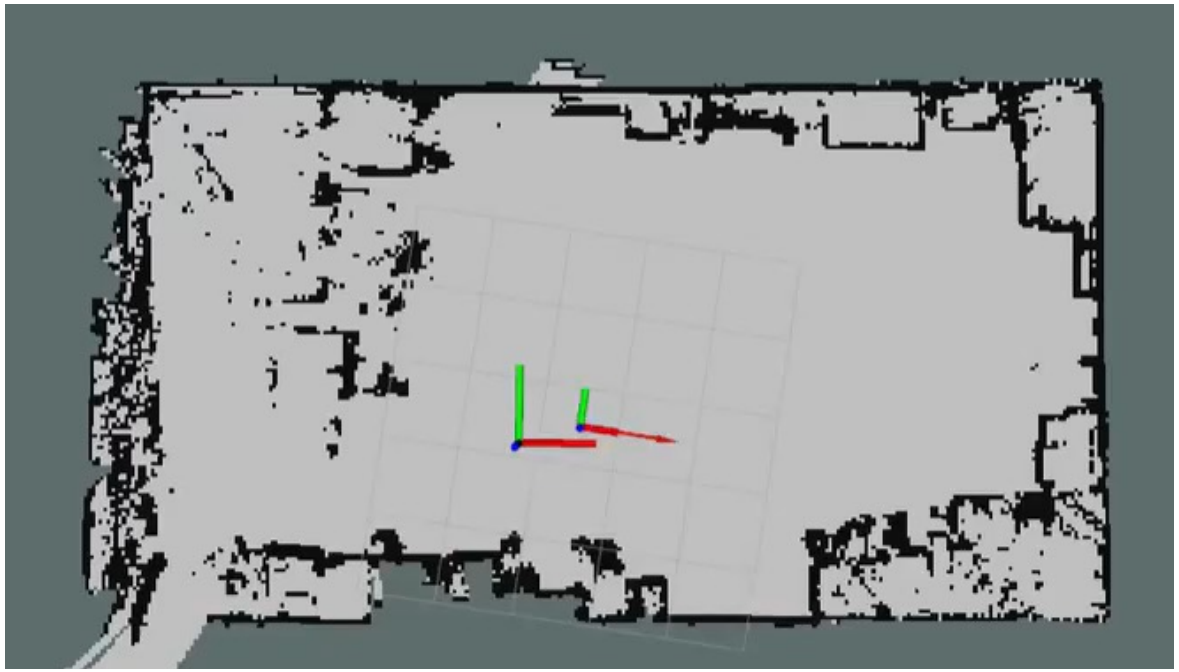


Robot odometry can be estimated by the model of the robot motion

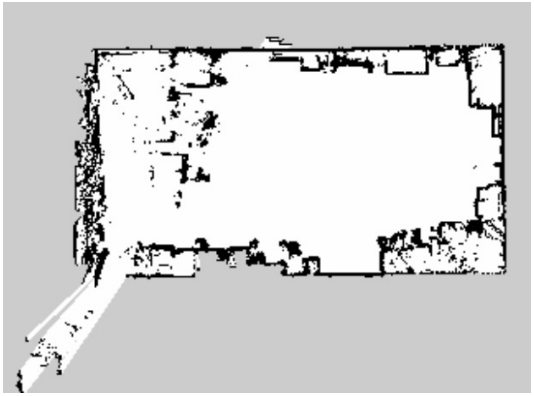
Implement 2D Localization based on Particle filter

➢ Particle Filter-based Localization

- ❑ A result of the particle filter-based localization on the track in C303.



Localization using Particle Filter



Generated map
using Hector SLAM

Implement Scan-matching Localization using PCL library

Implement Localization using PCL library

➢ Scan matching(Registration) algorithm using PCL library

- ❑ Search the translation($\boldsymbol{t}_\tau$) and rotation($\boldsymbol{R}_\tau$) (rigid transform) which minimize a distance error between the incoming scan points($\boldsymbol{p}_\tau = \sum p_i$) and pre-built map($\boldsymbol{M}$)
- ❑ i.e. ICP(Iterative Closest Point), NDT(Normal Distribution Transform) …
- ❑ It has the same function, but recently, an algorithm that can be processed with multi-thread is used.

*Parameters

Input : pointcloud(scan) → ($\boldsymbol{p}_\tau = \sum p_i$)
Target : pcd(pointcloud) format, occupancy grid map(2D) → pre-built map($\boldsymbol{M}$)

Common

- Leaf size(voxel size)[m]
- MSE(Mean squared error) score threshold(converged)
- Number of threads

NDT & (NDT-omp → multi-threads)

- Maximum Iteration
- Transformation epsilon[m]

ICP & (FAST-GICP → multi-threads)

- Maximum corresponding distance[m]

Implement Localization using PCL library

➢ Scan matching(Registration) algorithm using PCL library(cont')

***Parameters**

① Input : pointcloud(scan) → ($\boldsymbol{p}_\tau = \sum p_i$)
Target : pcd(pointcloud) format, occupancy grid map(2D) → pre-built map($\boldsymbol{M}$)

② **Common**
- Leaf size(voxel size)[m]
- MSE(Mean squared error) score threshold(converged)
- Number of threads

③ **NDT**
- Maximum Iteration
- Transformation epsilon[m]
- …

④ **FAST-GICP(ICP)**
- Maximum corresponding distance[m]
- …

① Make a input and target to the pcl::PointCloud<pointT>
PointT : pcl::PointXYZ, pcl::PointXYZI, pcl::PointXYZRGB

② voxelize filter, Mean squared error threshold, number of multi-threads

③ Transformation epsilon for steps, maximum iteration number

④ Maximum distance to find corresponding transformation matrix

Implement Localization using PCL library

- **Scan matching(Registration) algorithm using PCL library**
  - https://pcl.readthedocs.io/projects/tutorials/en/latest/iterative_closest_point.html#iterative-closest-point

```
#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/registration/icp.h>

int
 main (int argc, char** argv)
{
  pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_in (new pcl::PointCloud<pcl::PointXYZ>(5,1));
  pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_out (new pcl::PointCloud<pcl::PointXYZ>);

  // Fill in the CloudIn data
  for (auto& point : *cloud_in)
  {
    point.x = 1024 * rand() / (RAND_MAX + 1.0f);
    point.y = 1024 * rand() / (RAND_MAX + 1.0f);
    point.z = 1024 * rand() / (RAND_MAX + 1.0f);
  }

  std::cout << "Saved " << cloud_in->size () << " data points to input:" << std::endl;

  for (auto& point : *cloud_in)
    std::cout << point << std::endl;

  *cloud_out = *cloud_in;

  for (auto& point : *cloud_out)
    point.x += 0.7f;

  std::cout << "Transformed " << cloud_in->size () << " data points:" << std::endl;

  for (auto& point : *cloud_out)
    std::cout << point << std::endl;

  pcl::IterativeClosestPoint<pcl::PointXYZ, pcl::PointXYZ> icp;
  icp.setInputSource(cloud_in);
  icp.setInputTarget(cloud_out);

  pcl::PointCloud<pcl::PointXYZ> Final;
  icp.align(Final);

  std::cout << "has converged:" << icp.hasConverged() << " score: " <<
  icp.getFitnessScore() << std::endl;
  std::cout << icp.getFinalTransformation() << std::endl;

  return (0);
}
```

⟹ input, target

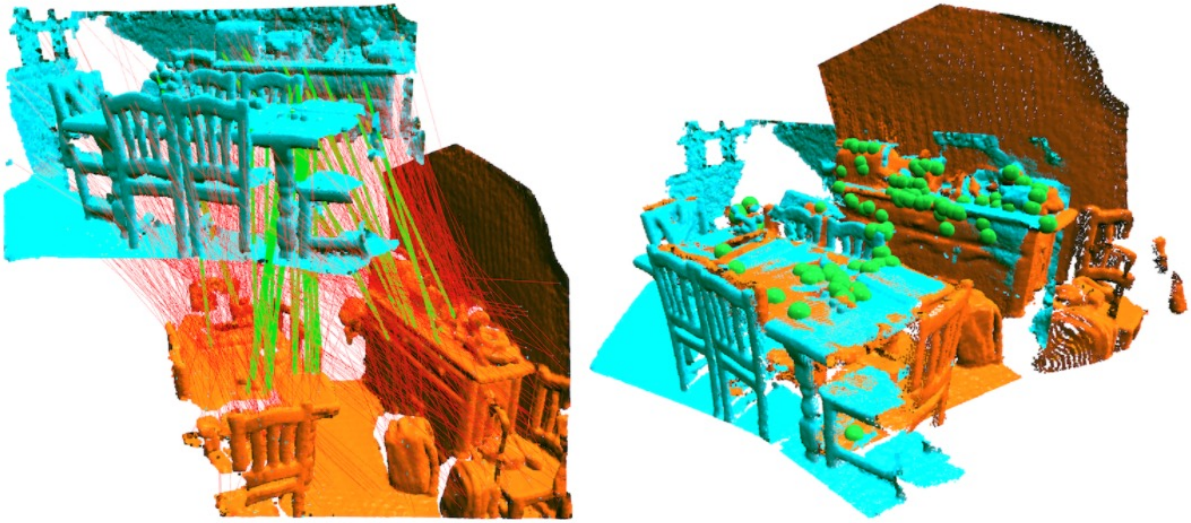
⟹ Input(put the random value)

⟹ To implement the algorithm, add arbitrary values.

⟹ Set the registration method. Set the parameters.

Implement Localization using PCL library

- **Scan matching(Registration) algorithm using PCL library**
  - It has the same function, but recently, an algorithm that can be processed with multi-thread is used.
  - i.e. NDT-omp, FAST-GICP, TEASER++, …
  - NDT-omp : https://github.com/koide3/ndt_omp
  - FAST-GICP: https://github.com/SMRT-AIST/fast_gicp
  - TEASER++: https://github.com/MIT-SPARK/TEASER-plusplus

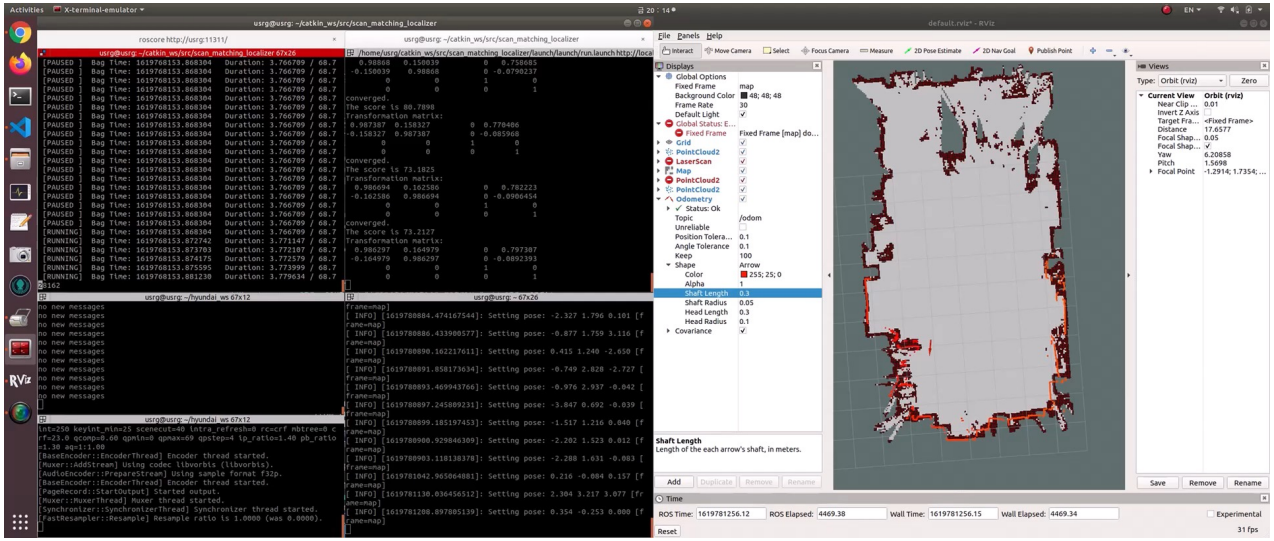


Yang, Heng, Jingnan Shi, and Luca Carlone. "Teaser: Fast and certifiable point cloud registration." *IEEE Transactions on Robotics* (2020).

Implement Localization using PCL library

- **Scan matching(Registration) algorithm using PCL library**

  *** Registration : NDT multi-thread**



1. Install NDT-omp. Please put the package in your catkin_ws/src directory :

https://github.com/koide3/ndt_omp

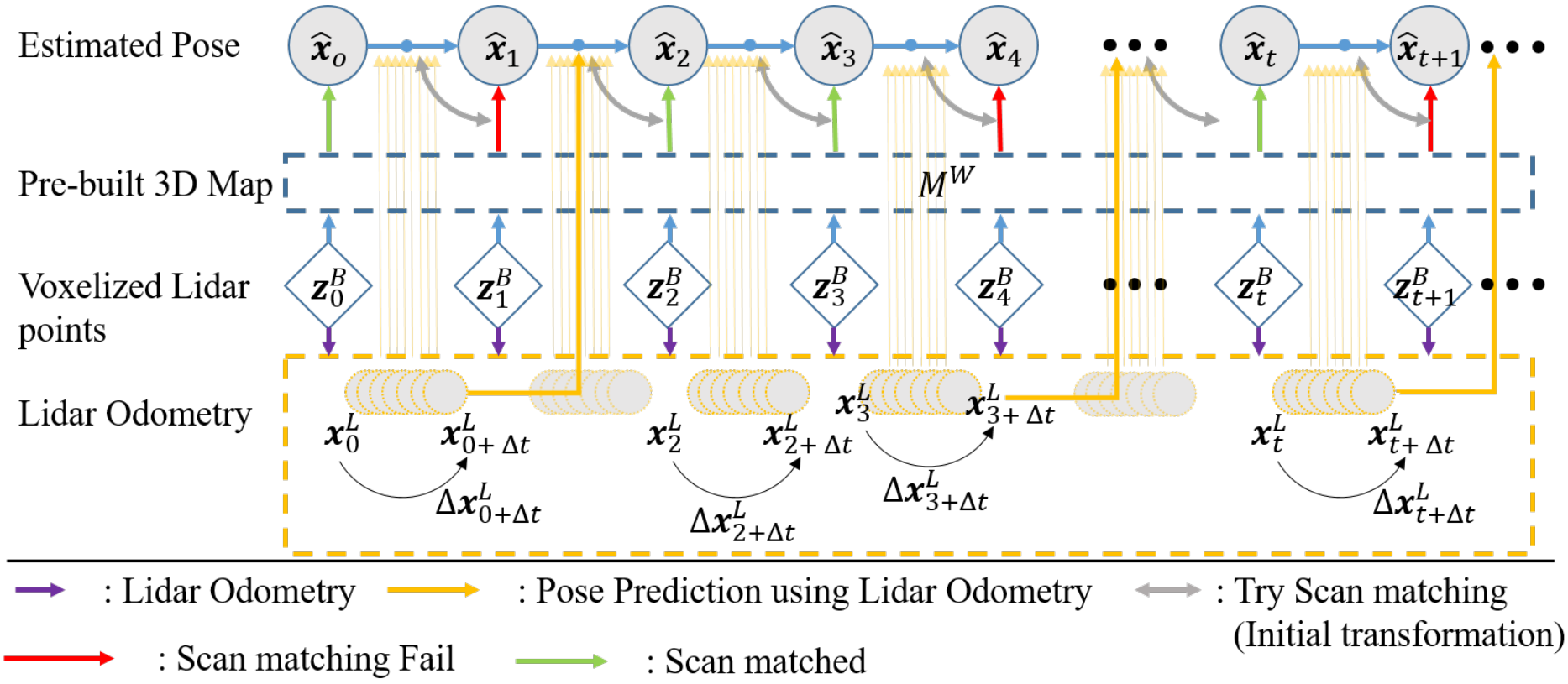
2. **You need to initialize the pose using RVIZ 2D Pose Estimate at first time**

Parametric study

LiDAR based Resilient Localization

Resilient Localization System

- Lidar Odometry + Scan matching + Extended Kalman Filter



Lee, D., Kang, G., Kim, B., & Shim, D. H. (2021). Assistive Delivery Robot Application for Real-World Postal Services. IEEE Access, 9, 141981-141998.

Parametric study

- Pointcloud Scan matching based Localization
  - ICP, NDT, FAST-GICP, TEASER++, …
  - If we use the **same parameters**, which are implemented in the mobile robot/Drones, scan matching algorithm fails when vehicle moves over the 30km/h

***Tuning Parameters**

Input pointcloud : pointcloud ring : upper ring 64 ~ 96 (origin → 128ch)

Target pointcloud : Sangam map sliding window (Origin map point size : 22,728,747)

Common

- Input cloud radius[m] : 70, 80, 90, 100
- Sliding window map size(radius)[m] : 70, 80, 90, 100
- Leaf size(voxel size)[m] : 0.5 ~ 1.0
- MSE(Mean squared error) score threshold : 0.2 ~ 0.5

NDT

- Maximum Iteration : 32, 64, 128
- Transformation epsilon : 0.01 ~ 0.2

FAST-GICP

- Number of threads : 2~10
- Maximum corresponding distance : 0.5 ~ 5.0

Parametric study

❖ Case study (NDT-omp registration)

Case 1 → FAIL

Common
- Input cloud channel : 128(full)
- Input cloud radius[m] : 70
- Sliding window map size(radius)[m] : 70
- Leaf size(voxel size)[m] : 0.5
- MSE(Mean squared error) score threshold : 0.2

NDT
- Maximum Iteration : 32
- Transformation epsilon : 0.01

⇨

- Very fast
- Too short range(70m) in the main street
- Too small iteration in the high speed
- Too small epsilon in the high speed
  → more accurate in the moderate speed

Case 2 → FAIL

Common
- Input cloud channel : 64(upper)
- Input cloud radius[m] : 70
- Sliding window map size(radius)[m] : 70
- Leaf size(voxel size)[m] : 0.5
- MSE(Mean squared error) score threshold : 0.2

NDT
- Maximum Iteration : 64
- Transformation epsilon : 0.01

⇨

- Half lidar channel(upper)
  → fast but less accuracy
- Too short range(70m) in the main street
- Enough iteration in the high speed
- Too small epsilon in the high speed
  → more accurate in the moderate speed

Parametric study

❖ Case study (NDT-omp registration)

Case 3 → FAIL

Common
- Input cloud channel : 64(upper)
- Input cloud radius[m] : 100
- Sliding window map size(radius)[m] : 100
- Leaf size(voxel size)[m] : 0.5
- MSE(Mean squared error) score threshold : 0.2

NDT
- Maximum Iteration : 128
- Transformation epsilon : 0.01

⇨

- Half lidar channel(upper)
  → fast but less accuracy
- Enough range(100m) in the main street
  → it can make slow output
- Too many iteration
  → scan matching frequency is too slow (~ 0.3 Hz)
- Too small epsilon in the high speed

Case 4 → SUCCESS!

Common
- Input cloud channel : 64(upper)
- Input cloud radius[m] : 80
- Sliding window map size(radius)[m] : 80
- Leaf size(voxel size)[m] : 0.5
- MSE(Mean squared error) score threshold : 0.2

NDT
- Maximum Iteration : 64
- Transformation epsilon : 0.03

⇨

- Half lidar channel(upper)
  → fast but less accuracy
- Enough range(80m) in the main street
- Enough iteration
  → scan matching frequency keeps 8~ 12 Hz
- Enough epsilon in the high speed

It succeeded in the ROS-bag data
HOWEVER, IT FAILED IN THE REAL-WORLD TEST the next day.
We determined to change the registration algorithm to the **FAST-GICP** algorithm.

Parametric study

❖ Case study (FAST-GICP algorithm)

Case 1 → FAIL

Common
- Input cloud channel : 96(upper)
- Input cloud radius[m] : 100
- Sliding window map size(radius)[m] : 100
- Leaf size(voxel size)[m] : 1.0
- MSE(Mean squared error) score threshold : 0.5

FAST-GICP
- Number of Thread : 2
- Maximum corresponding distance[m] : 1.0

⇨

- Half lidar channel(only upper) is less accuracy → add 32 below channel
  → still fast (like 64ch)
- Enough range(100m) in the main street
- Too small thread → slow
- Too small distance
  → Problem at the high speed

Case 2 → FAIL

Common
- Input cloud channel : 96(upper)
- Input cloud radius[m] : 100
- Sliding window map size(radius)[m] : 100
- Leaf size(voxel size)[m] : 1.0
- MSE(Mean squared error) score threshold : 0.5

FAST-GICP
- Number of Thread : 10
- Maximum corresponding distance[m] : 1.0

⇨

- Half lidar channel(only upper) is less accuracy → add 32 below channel
  → still fast (like 64ch)
- Enough range(100m) in the main street
- Too many thread
  → it makes other packages slow!
- Too small distance
  → Problem at the high speed

Parametric study

❖ Case study (FAST-GICP algorithm)

Case 3 → FAIL

Common
- Input cloud channel : 96(upper)
- Input cloud radius[m] : 100
- Sliding window map size(radius)[m] : 100
- Leaf size(voxel size)[m] : 1.0
- MSE(Mean squared error) score threshold : 0.5

FAST-GICP
- Number of Thread : 6
- Maximum corresponding distance[m] : 1.0

⇨

- Half lidar channel(only upper) is less accuracy → add 32 below channel → still fast (like 64ch)
- Enough range(100m) in the main street
- Moderate thread
- Too small corresponding distance → Problem at the high speed

Case 4 → SUCCESS!

Common
- Input cloud channel : 128
- Input cloud radius[m] : 100
- Sliding window map size(radius)[m] : 100
- Leaf size(voxel size)[m] : 1.0
- MSE(Mean squared error) score threshold : 0.5

FAST-GICP
- Number of Thread : 6
- Maximum corresponding distance[m] : 3.0

⇨

- Full Lidar channel → most accurate → still fast (like 64ch)
- Enough range(100m) in the main street
- Moderate thread
- Moderate corresponding distance

It succeeded in the ROS-bag data. Also, it succeeded in the REAL-WORLD test. →
Experiments are required for each environment that changes according to the season

Summary and Assignments

Experiment Summaries

- SLAM basics
  - 2-D, 3-D SLAMs supported by ROS.
  - Occupancy grid map messages to save the 2-D map
- Scan matching algorithm
  - How does the scan matching algorithm work?
- Implement 2D Localization based on Particle filter
- Implement 2D Localization based on PCL library
  - Example code
  - Real-world test
- Parametric study

Programming Assignment

- **Real-world navigation test(5pts):**
  - **Generate a 2-D map for N5 2354 room**
    - ✓ Publish generate map (1.5pts)
      - topic name is '/map_your_student_id'
      - message type is nav_msgs/OccupancyGrid
  - **Implement a PCL-based scan matching algorithm**
    - ✓ Clarify your ICP/NDT algorithm (0.5pts)
    - ✓ Record a localization result with video (less than 30s, max file size : 20MB)
    - ✓ Publish estimated vehicle pose type (2.0pts)
      - topic name is '/odom_your_student_id'
      - message type is nav_msgs/Odometry
  - **Generate a waypoint**
    - ✓ Save accumulated vehicle odometry to generate a reference waypoint
    - ✓ Pre-process or pose-process your waypoint to make a interpolated path with 0.1m step.
    - ✓ Publish your waypoint (1.0 pts)
      - topic name is "/path_your_student_id"
      - message type is nav_msgs/Path
  - **Screen capture your RVIZ tool & record**

Programming Assignment

- **Real-world navigation test(5pts):**
  - Screen capture your source code
  - Please submit your report in PDF format.
  - Submit RVIZ recording video and screen capture (less than 30s, max file size : 20MB)

Q & A

Email : lee.dk@kaist.ac.kr