RE510 Lab 2
Graph-Based SLAM

Seungwon Song
Sswan55@kaist.ac.kr

Lab 2

- Learn about Graph based SLAM (Simultaneous Localization And Mapping)

- In this lab, we use odometry and LiDAR data.

- Brief introduction for SLAM..
  - Odometry always have error (drift, wheel diameter changing, etc.)
  - Local odometry might be accurate, but after accumulation, it's not.
  - With SLAM, using external sensor(LiDAR, image, ...), can correct the robot position.

SLAM Approaches

- Filtering
  - Current robot pose
  - All landmark positions

  - Matrix inverse (EKF-SLAM)
  - Fixed-linearize point

  - Particle filter SLAM
  - Fast SLAM

- Graph-based (Smoothing)
  - All robot pose
  - All landmark positions

  - Least square approach
  - Sparse matrix factorization
  - Re-linearize

EKF-SLAM (Dissanayake, IEEE TRO, 2001)	Sparse Extended Information Filter (S. Thrun, IJRR, 2004)	SAM (F. Dellaert, M. Kaess, IJRR, 2006)	iSAM (M. Kaess, IEEE T.RO., 2008)

Least-Square Error

- Error function
  - Error $\mathbf{e}_i$ is typically the **difference** between the **predicted and actual** measurement

$$\mathbf{e}_i(\mathbf{x}) \quad = \quad \mathbf{z}_i - f_i(\mathbf{x})$$

  - We assume that the error has **zero mean** and is **normally distributed**
  - Gaussian error with information matrix $\boldsymbol{\Omega}_i$
  - The squared error of a measurement depends only on the state and is a scalar

$$e_i(\mathbf{x}) \quad = \quad \mathbf{e}_i(\mathbf{x})^T \boldsymbol{\Omega}_i \mathbf{e}_i(\mathbf{x})$$

Inverse of covariance matrix of measurement

Reference: University Freiburg 'Robot Mapping – WS 2017/18 Lecture Notes', Wolfram Burgard

Least-Square Error

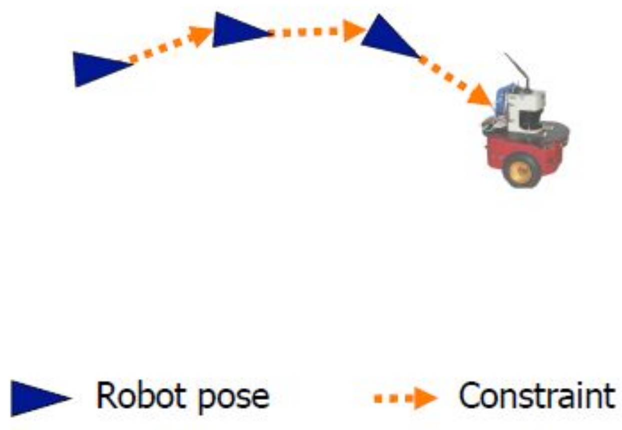
- Find minimum of error function
  - Find the state $\mathbf{x}^*$ which minimizes the error given all measurements

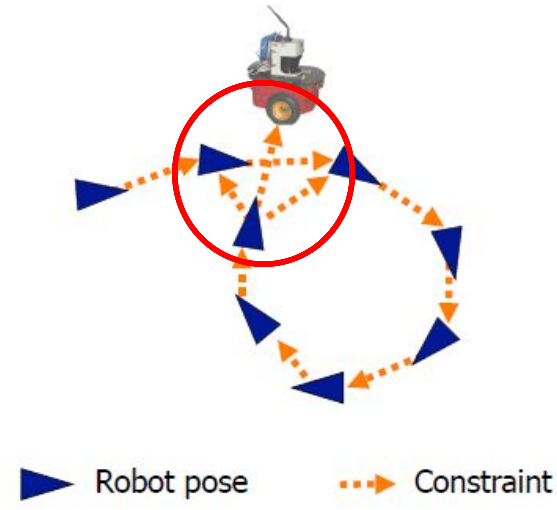
$$
\begin{aligned}
\mathbf{x}^* &= \underset{\mathbf{x}}{\operatorname{argmin}}\, F(\mathbf{x}) \longleftarrow \boxed{\text{global error (scalar)}} \\
&= \underset{\mathbf{x}}{\operatorname{argmin}} \sum_i e_i(\mathbf{x}) \longleftarrow \boxed{\text{squared error terms (scalar)}} \\
&= \boxed{\underset{\mathbf{x}}{\operatorname{argmin}} \sum_i \mathbf{e}_i^T(\mathbf{x}) \Omega_i \underbrace{\mathbf{e}_i(\mathbf{x})}_{\boxed{\text{error terms (vector)}}}}
\end{aligned}
$$

  - When measurement's dimension is 'n',
  - Complex and no closed form (미지수의 해가 수식으로 정리되는 형태가 아님)
  - So, numerical approaches is needed

Least-Square Approach to SLAM

- **Graph SLAM**: use a graph to represent the problem
- Node(Vertex) : pose of robot
- Edge : constraint between nodes (odometry, LiDAR, vision..)
- Build the graph and find a node configuration that minimize the error introduced by the constraints



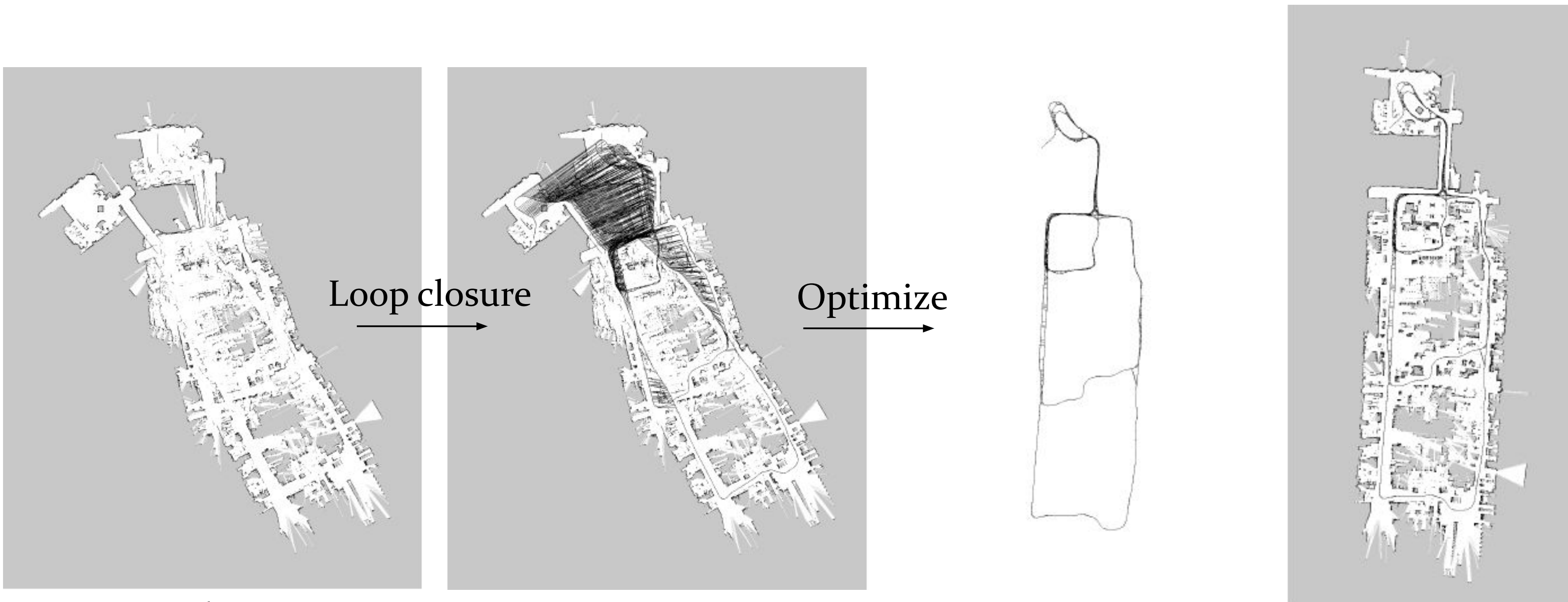


- When revisiting a previously visited place, optimizes the graph by recognizing that it is the same location using information about the surrounding environment (sensor data) and adding constraints between non-successive nodes.

KAIST

URL URBAN ROBOTICS LAB

Example



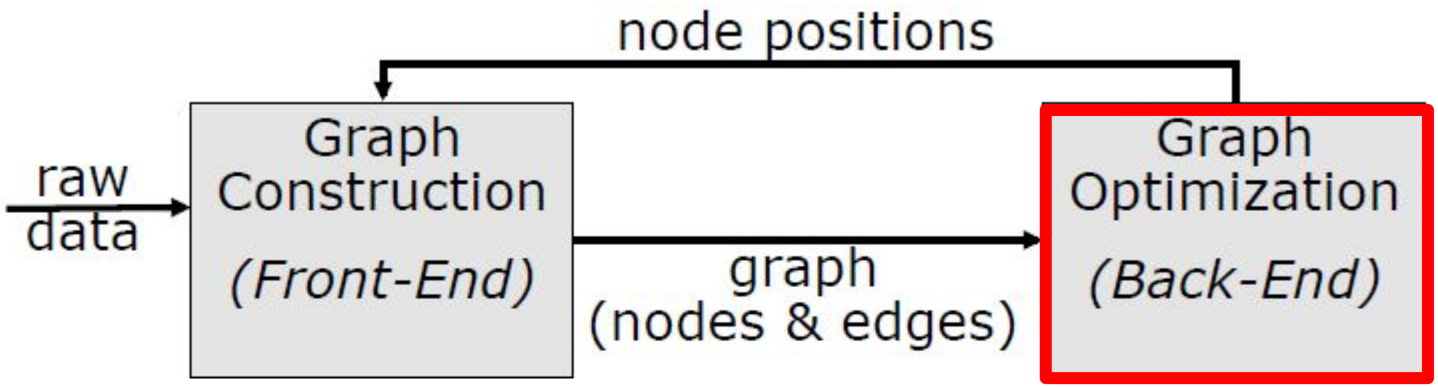
Raw odometry
Grid Map

Loop closure matched

Optimized Path

Optimized Map

Overall Graph SLAM system



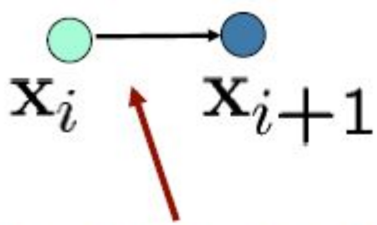
- Front-End: generates constraints between nodes
- Back-End: optimizing nodes(pose) using constraints and information (ex. iSAM, g2o, ceres)

The Graph

- It consists of n nodes $\mathbf{x} = \mathbf{x}_{1:n}$
- Each $\mathbf{x}_i$ is a 2D or 3D transformation (the pose of the robot at time t_i)
- A constraint/edge exists between the nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ if…

• Create an Edge if.. (1)

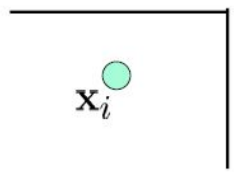
- …the robot moves from $\mathbf{x}_i$ to $\mathbf{x}_{i+1}$
- Edge corresponds to odometry



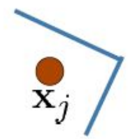
The edge represents the **odometry** measurement

The Graph

- **Create an Edge if.. (2)**
  - ...the robot observes the same part of the environment from $\mathbf{x}_i$ and from $\mathbf{x}_j$

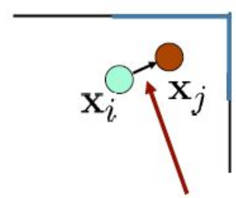


Measurement from $\mathbf{x}_i$



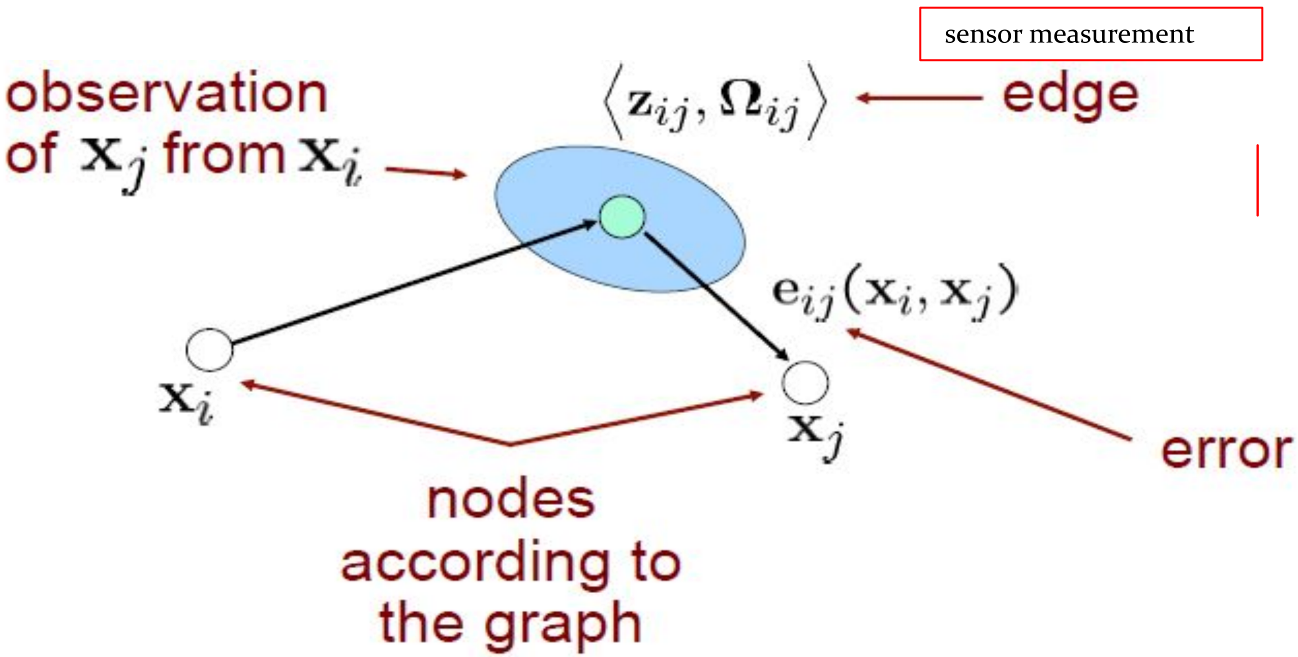
Measurement from $\mathbf{x}_j$

  - Construct a **virtual measurement** about the position of $\mathbf{x}_j$ seen from $\mathbf{x}_i$



Edge represents the position of $\mathbf{x}_j$ seen from $\mathbf{x}_i$ based on the **observation**

Pose Graph



- **Goal:** $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} \sum_{ij} \mathbf{e}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}$

- Information matrix is inverse of covariance
- The "bigger" , the more the edge "matters" in the optimization.

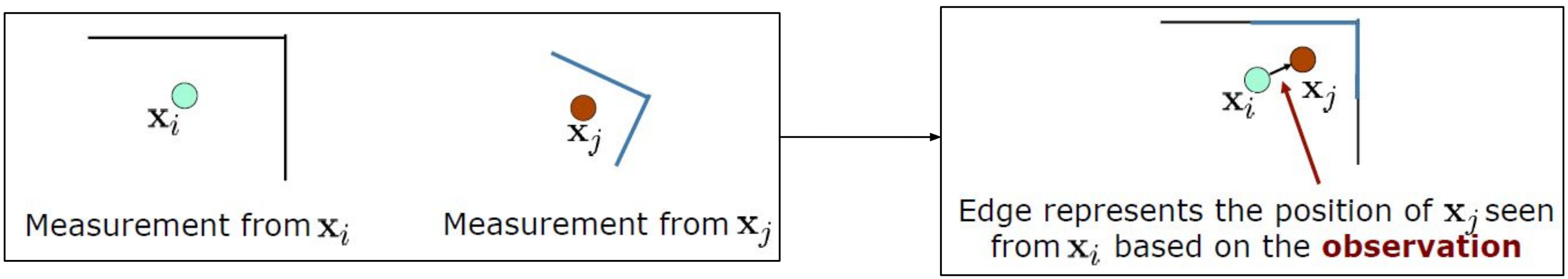
Optimize algorithm

- In this experiment, we use G2O ported to python.

- If you're interested in optimization equations and algorithm, read the paper :
  - Kümmerle, Rainer & Grisetti, Giorgio & Strasdat, Hauke & Konolige, Kurt & Burgard, Wolfram. (2011). G2o: A general framework for graph optimization. Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA). 3607 - 3613. 10.1109/ICRA.2011.5979949.

ICP

- In page 8, there is situation of creating observation edge.
- How can we get those type of edge?
- For matching two LiDAR pointclouds, we use Iterative-Closest-Point algorithm.



ICP

- Given two corresponding point sets:

  $X = \{x_1, ..., x_{N_x}\}$

  $P = \{p_1, ..., p_{N_p}\}$

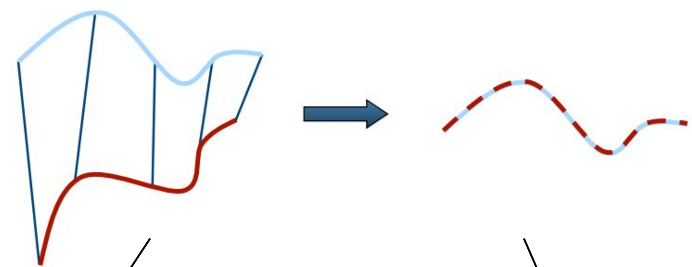
- Wanted: Translation t and rotation R that minimize the sum of the squared error:

$$E(R,t) = \frac{1}{N_p} \sum_{i=1}^{N_p} ||x_i - Rp_i - t||^2$$

Where x_i and p_i are corresponding points

- If the correct correspondences are known, the correct relative rotation/translation can be calculated in **closed form**

Idea



Translation

Rotation

Center of Mass

$$\mu_x = \frac{1}{N_x} \sum_{i=1}^{N_x} x_i \quad \text{and} \quad \mu_p = \frac{1}{N_p} \sum_{i=1}^{N_p} p_i$$

are the centers of mass of the two point sets

Idea:

- Subtract the corresponding center of mass from every point in the two point sets before calculating the transformation
- The resulting point sets are:

$$X' = \{x_i - \mu_x\} = \{x'_i\} \quad \text{and}$$
$$P' = \{p_i - \mu_p\} = \{p'_i\}$$

5

Singular Value Decomposition

Let $W = \sum_{i=1}^{N_p} x'_i p'^T_i$

denote the singular value decomposition (SVD) of W by:

$$W = U \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} V^T$$

where $U, V \in \mathbb{R}^{3\times3}$ are unitary, and $\sigma_1 \geq \sigma_2 \geq \sigma_3$ are the singular values of W

Reference : http://ais.informatik.uni-freiburg.de/teaching/ss12/robotics/slides/17-icp.pdf

KAIST

URBAN ROBOTICS LAB

Basic ICP Algorithm

- Determine corresponding points → Find nearest neighbor of each points
- Compute rotation R, translation t via SVD → T by centroid, R by SVD
- Apply R and t to the points of the set to be registered
- Compute the error $E(R,t)$ → Mean distance between all corresponding points
- If error decreased and error > threshold
  - Repeat these steps
  - Stop and output final alignment, otherwise

Reference : http://ais.informatik.uni-freiburg.de/teaching/ss12/robotics/slides/17-icp.pdf

Experiment

- Environment
  - Ubuntu (18.04 tested)
  - Python (2.7 prefered)
    - numpy
    - sklearn
    - scipy
    - matplotlib
  - G2opy (Ported g2o)
    - Included on prereq/g2opy
    - Have to build.. (instruction : prereq/Readme.txt)

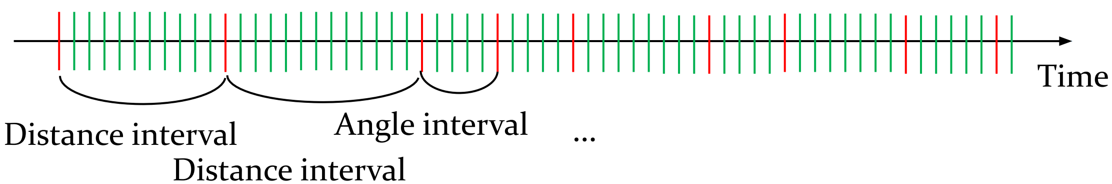
Experiment

- Assignment
  - Fill the skeleton code in slam.py
  - Run the code, save the result of matplotlib
  - Write the report..
    - Explain slam.py code (with principle of graph SLAM)
    - Explain icp.py code (with theory of ICP)
    - Contain image result (at page 21)

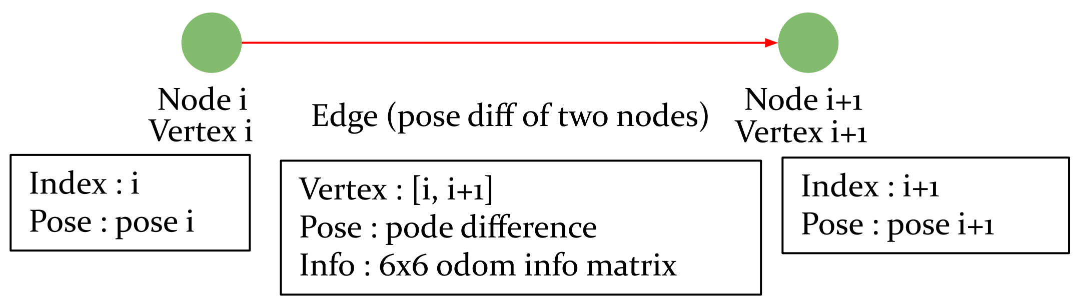
- Score Criteria
  - Make runnable code. (35%)
  - Write report. (50%)
  - Upgrade loop closing (10%)
  - Define new information matrix (5%)

Experiment

- Assignments
  - Assignment 1 : Node generation
    - Calculate pose difference using odometry.
    - In certain interval, generate node (odometry+LiDAR)

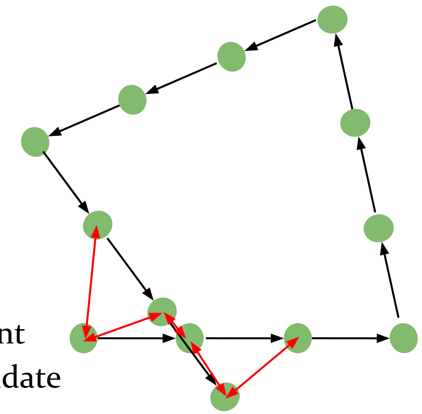


  - Assignment 2 : Add odometry vertex & edge
    - Add vertex and edge of odometry information

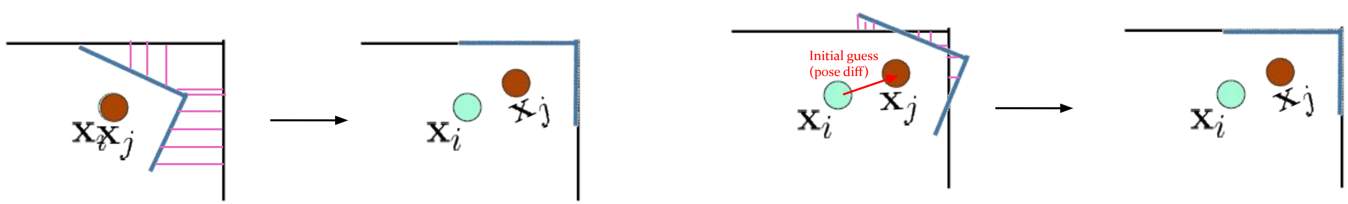


Experiment

- Assignments
  - Assignment 3 : Find loop closure
    - Find loop closure (matching pair)
    - Pair can be near nodes. (in certain distance threshold)
    - Or just simply, try to match all pairs ([1,2], [1,3], [1,4], [1,5], [2,3], [2,4], ...)



  - Assignment 4 : ICP matching, Add edge and Optimize
    - With no initial guess / With initial guess



Hard to find correspondences between points
Hard to match, or even cannot match

Can find correct correspondences
Easy to match within few steps

    - Add edge using ICP constraint
      - If ICP matching is correct (can determine using distances between corresponding points.)
      - Add edge to graph structure for optimization.
      - How can you put constraint pose? (calculate using initial guess and ICP result)

KAIST

URBAN ROBOTICS LAB

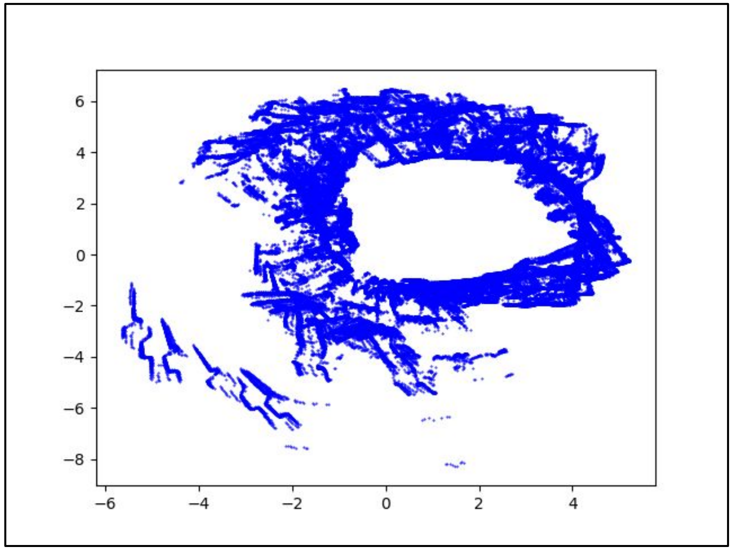
Experiment

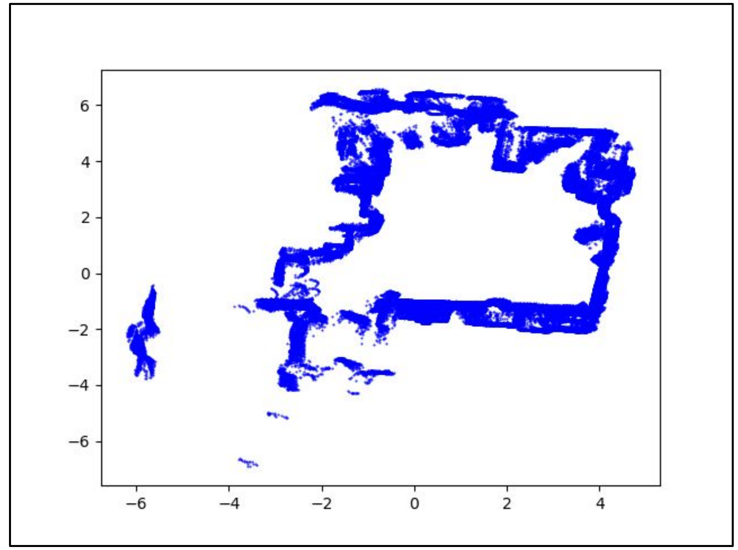
- Additional notes
  - Actually, assignment 3 and assignment 4 have to done simultaneously (find loop closure -> try matching -> optimize -> find another using optimized pose)
    But for simplicity, I split assignment 3 and assignment 4.
    If you want to do strictly, you can merge them.
  - Loop closure score
    - Brute force(all pair) : 5 / 10
    - Find near node in odom : 8 / 10
    - Merge assign 3,4 : 10 / 10
  - Information matrix score
    - Identity matrix * constant : 3/5
    - Related to odometry difference : 5/5
      (ex: If robot rotate, large yaw covariance, robot go forward, large x covariance and etc.)

x-x	x-y	x-z	x-roll	x-pitch	x-yaw
	y-y	y-z	y-roll	y-pitch	y-yaw
		z-z	z-roll	z-pitch	z-yaw
			roll-roll	roll-pitch	roll-yaw
				pitch-pitch	pitch-yaw
					yaw-yaw

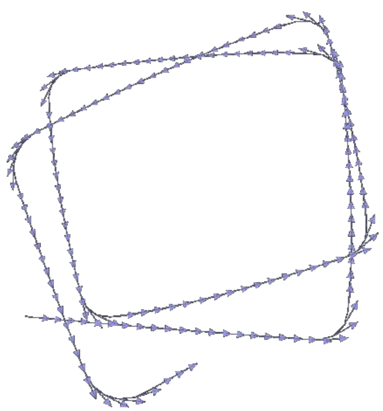
Experiment

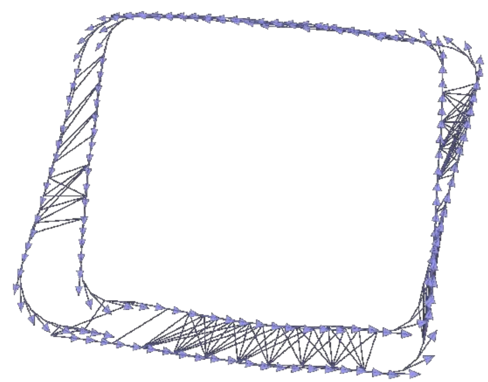
- Example result





Matplotlib





G2o viewer
(option)