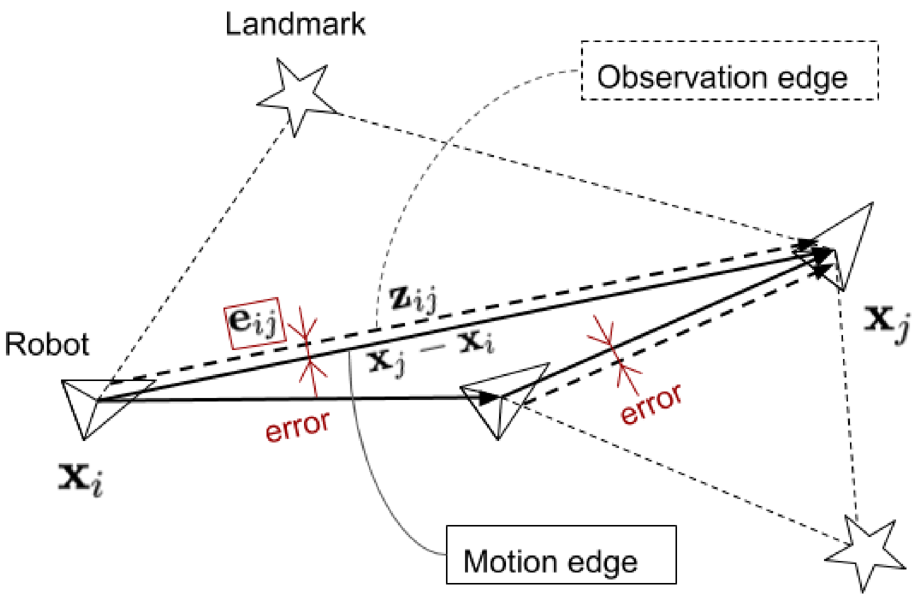
Graph Based SLAM

1 What is Graph Based SLAM?

Graph Based SALM is one of the offline SLAM method, which means correcting whole historical robot trajectory and landmarks position using all observation data. Considering a robot pose at time t_i as a node and a vector between 2 nodes (between 2 robot poses at time t_i and t_j) as an edge, Graph Based SLAM will try to find the best estimated robot and landmarks poses that minimize a cost function. The cost function contains all errors between a motion edge and a corresponding observation edge.



The error $\boldsymbol{e}_{ij}$ between a motion edge and an observation edge in different pose i and j is;

$$\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_j - \boldsymbol{x}_i) - \boldsymbol{z}_{ij}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are the robot poses in time step i and j respectively, and $\boldsymbol{z}_{ij}$ is the edge obtained by observing a same landmark in pose i and j. Thus, the cost function $F(\boldsymbol{x}_{0:t})$ can be expressed as;

$$F(\boldsymbol{x}_{0:t}) = \sum_{i,j} \boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)^{\mathrm{T}} \Omega_{ij} \boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

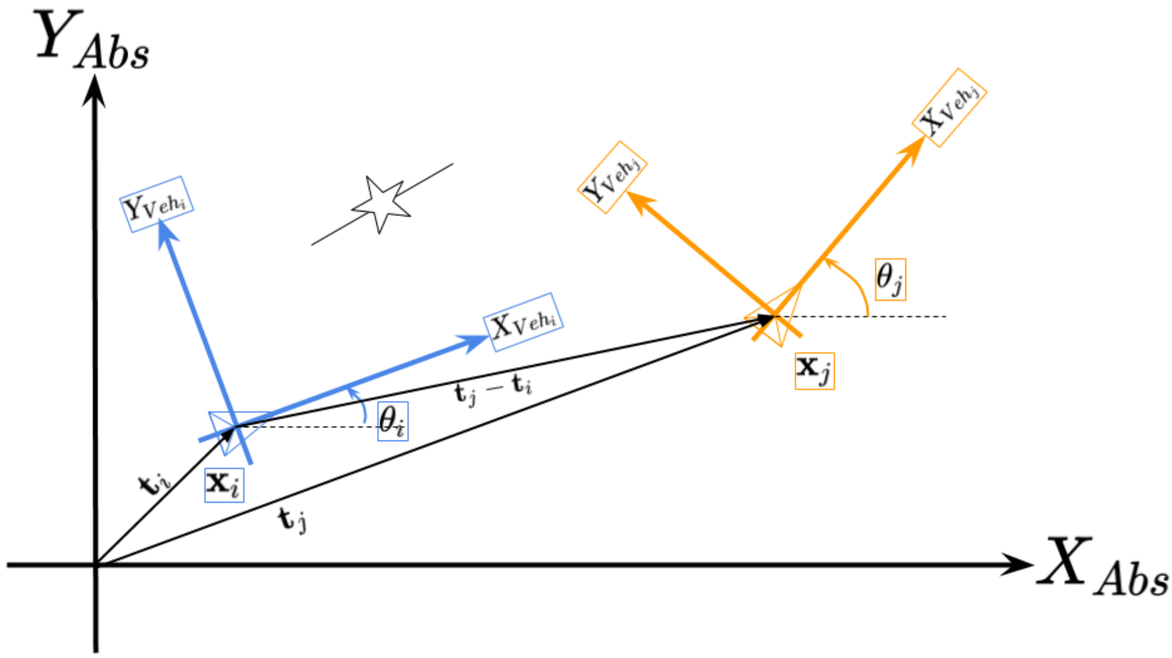
that is the sum of square errors $\boldsymbol{e}_{ij}^{\mathrm{T}} \boldsymbol{e}_{ij}$ weighted by information matrix Ω_{ij}, which expresses accuracy of the edge (Mahalanobis distance). The aim of Graph Based SLAM is to find the robot and landmarks poses that minimize this cost function.

2 Definitions

Defines some coordinates and variables used in this document.

2.1 Coordinate

There are 2 robot poses and 1 landmark on *Absolute* coordinate. Each robot pose has own *Vehicle* coordinate, the X axis is the heading direction and Y axis is the left hand of the robot. Assumes that every landmark has own angle, although the absolute value of the landmark's angle is not so important (to be discussed later).



2.2 Robot position vector: $\boldsymbol{t}_i$

It contains robot position x_i and y_i on *Absolute* coordinate.

$$\boldsymbol{t}_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

2.3 Robot state vector: $\boldsymbol{x}_i$

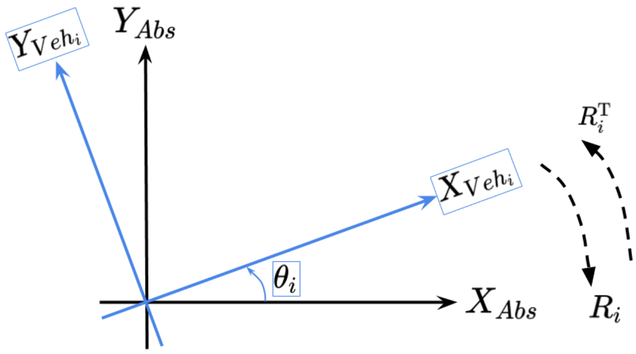
However, the robot pose has another parameter, which is yaw angle θ_i on *Absolute* coordinate.

$$\boldsymbol{x}_i = \begin{pmatrix} \boldsymbol{t}_i \\ \theta_i \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \\ \theta_i \end{pmatrix}$$

2.4 Rotation matrix: R_i

This matrix rotates the coordinate clockwise. In particular, R_i converts $Vehicle_i$ coordinate into the $Absolute$ coordinate.

$$R_i = \begin{pmatrix} cos\theta_i & -sin\theta_i \\ sin\theta_i & cos\theta_i \end{pmatrix}$$



2.5 Pose representation matrix: X_i

One of the way to represent the robot pose is transformation matrix, but there are a number of alternatives. In this document, the pose representation matrix consists of rotation matrix R_i and robot position vector t_i.

$$X_i = \begin{pmatrix} R_i & \boldsymbol{t}_i \\ 00 & 1 \end{pmatrix} = \begin{pmatrix} cos\theta_i & -sin\theta_i & x_i \\ sin\theta_i & cos\theta_i & y_i \\ 0 & 0 & 1 \end{pmatrix}$$

$$X_i = \begin{pmatrix} \overset{Abs \leftarrow Veh_i}{R_i} & \overset{Abs}{\mathbf{t}_i} \\ 00 & 1 \end{pmatrix}$$

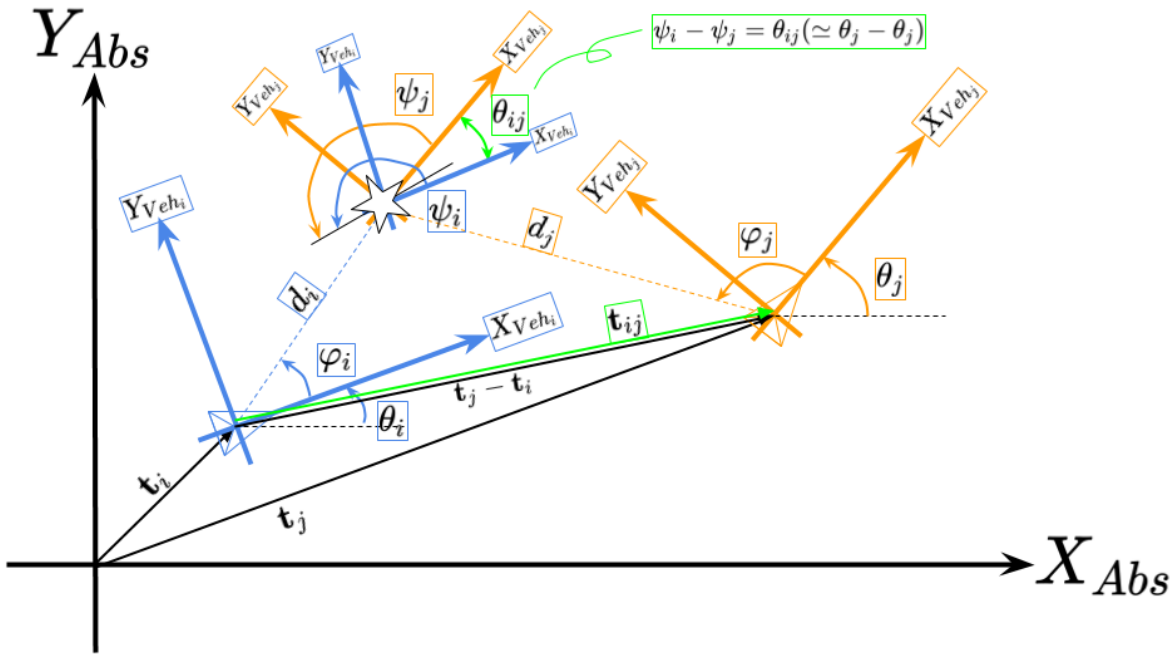
The inverse of the pose representation matrix has power to convert the origin from $Absolute$ coordinate into own coordinate. In particular, X_i^{T} converts the origin from $Absolute$ coordinate into $Vehicle_i$ coordinate.

$$X_i^{-1} = \begin{pmatrix} R_i^{\mathrm{T}} & -R_i^{\mathrm{T}}\boldsymbol{t}_i \\ 00 & 1 \end{pmatrix}$$

$$X_i^{-1} = \begin{pmatrix} \overset{Veh_i \leftarrow Abs}{R_i^{\mathrm{T}}} & -\overset{Veh_i \leftarrow Abs}{R_i^{\mathrm{T}}}\overset{Abs}{\mathbf{t}_i} \\ 00 & 1 \end{pmatrix}$$

2.6 Sensor model

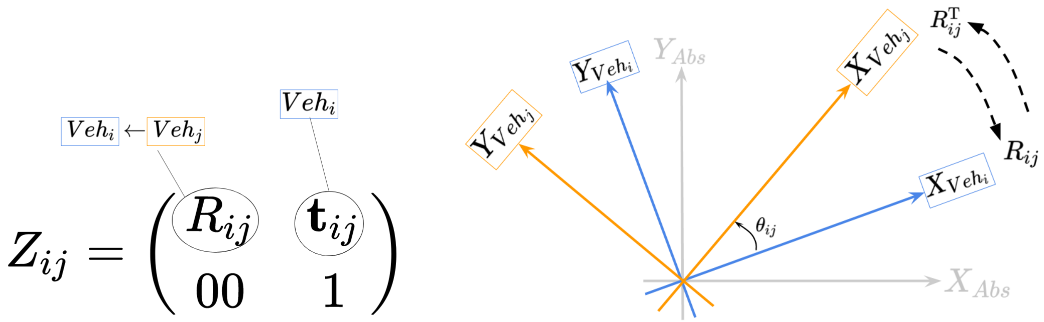
The sensor on the robot observes two information, the distance d and the angle φ from the robot to a landmark. When the robot in a pose i and j observe a same landmark, the robot can calculate the difference of the landmark angle $\psi_i - \psi_j$ from each view by using computer vision or something, although the robot doesn't know the absolute values of ψ_i and ψ_j (the robot only can know the relative value). And θ_{ij} in the observation representation Z_{ij} denotes this relative value of the landmark angle $\psi_i - \psi_j$ from each view.



2.7 Observation representation matrix: Z_{ij}

If the robot in different poses observe a same landmark, then the relative pose of the robot between these poses can be calculated from the view of the landmark. Assumes that the robot in a pose i and j observe a same landmark, the observation representation Z_{ij} will be,

$$Z_{ij} = \begin{pmatrix} R_{ij} & \boldsymbol{t}_{ij} \\ 00 & 1 \end{pmatrix} = \begin{pmatrix} cos\theta_{ij} & -sin\theta_{ij} & x_{ij} \\ sin\theta_{ij} & cos\theta_{ij} & y_{ij} \\ 0 & 0 & 1 \end{pmatrix}$$



The origin of the observation representation Z_{ij} is on $Vehicle_i$ coordinate, thus $\boldsymbol{t}_{ij}$ means the distance and θ_{ij} means the angle from $Vehicle_i$ coordinate to $Vehicle_j$ coordinate.

3 Optimization

The cost function calculated by errors between edges can be optimized using a least square method such as the Gauss-Newton algorithm.

3.1 Error and Cost function

The $\boldsymbol{t}_{ij}$ and $\theta_{ij} (= \psi_i - \psi_j)$ in the observation representation Z_{ij} should be equal to the difference of the position $\boldsymbol{t}_j - \boldsymbol{t}_i$ and the angle $\theta_j - \theta_i$ respectively in an ideal environment. However in the real world, these values are different due to sensor error.

$$\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_j - \boldsymbol{x}_i) - \boldsymbol{z}_{ij}$$

The error function $\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ can be expressed by using the pose representation matrix X_i, X_j and the observation representation matrix Z_{ij} introduced in the previous section. First, the motion edge between poses in time step i and j is calculated by multiplying the inverse of i th pose representation X_i^{-1} to j th pose representation X_j;

$$\begin{aligned} X_i^{-1} X_j &= \begin{pmatrix} R_i^{\mathrm{T}} & -R_i^{\mathrm{T}} \boldsymbol{t}_i \\ 00 & 1 \end{pmatrix} \begin{pmatrix} R_j & \boldsymbol{t}_j \\ 00 & 1 \end{pmatrix} \\ &= \begin{pmatrix} R_i^{\mathrm{T}} R_j & R_i^{\mathrm{T}} (\mathbf{t}_j - \mathbf{t}_i) \\ 00 & 1 \end{pmatrix} \end{aligned}$$

$Abs \leftarrow \boxed{Veh_j}$ $\quad$ Abs

$$X_i^{-1} X_j = \begin{pmatrix} R_i^{\mathrm{T}} R_j & R_i^{\mathrm{T}} (\mathbf{t}_j - \mathbf{t}_i) \\ 00 & 1 \end{pmatrix}$$

$\boxed{Veh_i} \leftarrow Abs$

Then, the error $\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ between the motion edge and the corresponding observation edge is obtained by multiplying the inverse of the observation representation Z_{ij}^{-1};

$$\begin{aligned} Z_{ij}^{-1}(X_i^{-1} X_j) &= \begin{pmatrix} R_{ij}^{\mathrm{T}} & -R_{ij}^{\mathrm{T}} \boldsymbol{t}_{ij} \\ 00 & 1 \end{pmatrix} \begin{pmatrix} R_i^{\mathrm{T}} R_j & R_i^{\mathrm{T}} (\boldsymbol{t}_j - \boldsymbol{t}_i) \\ 00 & 1 \end{pmatrix} \\ &= \begin{pmatrix} R_{ij}^{\mathrm{T}} R_i^{\mathrm{T}} R_j & R_{ij}^{\mathrm{T}} \{ R_i^{\mathrm{T}} (\boldsymbol{t}_j - \boldsymbol{t}_i) - \boldsymbol{t}_{ij} \} \\ 00 & 1 \end{pmatrix} \end{aligned}$$

$\boxed{Veh_i} \leftarrow Abs \leftarrow \boxed{Veh_j}$ $\quad$ $\boxed{Veh_i}$

$$Z_{ij}^{-1}(X_i^{-1} X_j) = \begin{pmatrix} R_{ij}^{\mathrm{T}} R_i^{\mathrm{T}} R_j & R_{ij}^{\mathrm{T}} \{ R_i^{\mathrm{T}} (\mathbf{t}_j - \mathbf{t}_i) - \mathbf{t}_{ij} \} \\ 00 & 1 \end{pmatrix}$$

$\boxed{Veh_j} \leftarrow \boxed{Veh_i}$

The error function $\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ consists of the translation elements $R_{ij}^{\mathrm{T}}\{R_i^{\mathrm{T}}(\boldsymbol{t}_j - \boldsymbol{t}_i) - \boldsymbol{t}_{ij}\}$ and the (angle of the) rotation elements $R_{ij}^{\mathrm{T}} R_i^{\mathrm{T}} R_j$ of this matrix $Z_{ij}^{-1}(X_i^{-1} X_j)$;

$$\begin{aligned}\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \begin{pmatrix} R_{ij}^{\mathrm{T}}\{R_i^{\mathrm{T}}(\boldsymbol{t}_j - \boldsymbol{t}_i) - \boldsymbol{t}_{ij}\} \\ angle(R_{ij}^{\mathrm{T}} R_i^{\mathrm{T}} R_j) \end{pmatrix} \\ &= \begin{pmatrix} R_{ij}^{\mathrm{T}}\{R_i^{\mathrm{T}}(\boldsymbol{t}_j - \boldsymbol{t}_i) - \boldsymbol{t}_{ij}\} \\ (\theta_j - \theta_i) - \theta_{ij} \end{pmatrix}\end{aligned}$$

The objective of Graph Based SLAM is to reduce these errors — between *Motion model* and *Observation model* — by using weighted square errors (Mahalanobis distance) and a least squares method (the Gauss-Newton algorithm) with a sparse graph structure. The cost function is the sum of the weighted square errors $\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ across all observation data;

$$\begin{aligned}F(\boldsymbol{x}_{0:t}) &= \sum_{i,j} \boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)^{\mathrm{T}} \Omega_{ij} \boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &= \boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t})^{\mathrm{T}} \Omega_{0:t} \boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t})\end{aligned}$$

3.2 Linearization

The Gauss-Newton algorithm is used to minimize this const function. The idea is to approximate the error function by its 1st order Taylor expansion around the current initial guess $\boldsymbol{x}_{0:t}$.

$$\begin{aligned}F(\boldsymbol{x}_{0:t} + \Delta\boldsymbol{x}_{0:t}) &= \boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t} + \Delta\boldsymbol{x}_{0:t})^{\mathrm{T}} \Omega_{0:t} \boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t} + \Delta\boldsymbol{x}_{0:t}) \\ &\simeq (\boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t}) + J_{0:t}\Delta\boldsymbol{x}_{0:t})^{\mathrm{T}} \Omega_{0:t} (\boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t}) + J_{0:t}\Delta\boldsymbol{x}_{0:t}) \\ &= \{\boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t})^{\mathrm{T}} + (J_{0:t}\Delta\boldsymbol{x}_{0:t})^{\mathrm{T}}\} \Omega_{0:t} (\boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t}) + J_{0:t}\Delta\boldsymbol{x}_{0:t}) \\ &= \boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t})^{\mathrm{T}} \Omega_{0:t} \boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t}) + (J_{0:t}\Delta\boldsymbol{x}_{0:t})^{\mathrm{T}} \Omega_{0:t} \boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t}) \\ &\quad + \boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t})^{\mathrm{T}} \Omega_{0:t} (J_{0:t}\Delta\boldsymbol{x}_{0:t}) + (J_{0:t}\Delta\boldsymbol{x}_{0:t})^{\mathrm{T}} \Omega_{0:t} (J_{0:t}\Delta\boldsymbol{x}_{0:t}) \\ &= F(\boldsymbol{x}_{0:t}) + \{(\Omega_{0:t}\boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t}))^{\mathrm{T}} (J_{0:t}\Delta\boldsymbol{x}_{0:t})\}^{\mathrm{T}} + \boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t})^{\mathrm{T}} \Omega_{0:t} J_{0:t}\Delta\boldsymbol{x}_{0:t} + \Delta\boldsymbol{x}_{0:t}^{\mathrm{T}} J_{0:t}^{\mathrm{T}} \Omega_{0:t} J_{0:t} \Delta\boldsymbol{x}_{0:t}\end{aligned}$$

Since $\Omega_{0:t}$ is a symmetric matrix, and $\boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t})^{\mathrm{T}} \Omega_{0:t} J_{0:t}\Delta\boldsymbol{x}_{0:t}$ is a scalar,

$$\begin{aligned}F(\boldsymbol{x}_{0:t} + \Delta\boldsymbol{x}_{0:t}) &\simeq F(\boldsymbol{x}_{0:t}) + (\boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t})^{\mathrm{T}} \Omega_{0:t} J_{0:t}\Delta\boldsymbol{x}_{0:t})^{\mathrm{T}} + \boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t})^{\mathrm{T}} \Omega_{0:t} J_{0:t}\Delta\boldsymbol{x}_{0:t} + \Delta\boldsymbol{x}_{0:t}^{\mathrm{T}} J_{0:t}^{\mathrm{T}} \Omega_{0:t} J_{0:t} \Delta\boldsymbol{x}_{0:t} \\ &= F(\boldsymbol{x}_{0:t}) + 2\boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t})^{\mathrm{T}} \Omega_{0:t} J_{0:t}\Delta\boldsymbol{x}_{0:t} + \Delta\boldsymbol{x}_{0:t}^{\mathrm{T}} J_{0:t}^{\mathrm{T}} \Omega_{0:t} J_{0:t} \Delta\boldsymbol{x}_{0:t} \\ &= F(\boldsymbol{x}_{0:t}) + 2\boldsymbol{b}_{0:t}^{\mathrm{T}} \Delta\boldsymbol{x}_{0:t} + \Delta\boldsymbol{x}_{0:t}^{\mathrm{T}} H_{0:t} \Delta\boldsymbol{x}_{0:t}\end{aligned}$$

where

$$\boldsymbol{b}_{0:t} = J_{0:t}^{\mathrm{T}} \Omega_{0:t} \boldsymbol{e}_{0:t}(\boldsymbol{x}_{0:t}), \quad H_{0:t} = J_{0:t}^{\mathrm{T}} \Omega_{0:t} J_{0:t}$$

3.3 Solve and Update

Regards $\boldsymbol{x}_{0:t}$ as a constant and $\Delta\boldsymbol{x}_{0:t}$ is a variable, the $\Delta\boldsymbol{x}_{0:t}$ which decreases the cost function $F(\boldsymbol{x}_{0:t} + \Delta\boldsymbol{x}_{0:t})$ most can be derived by differentiating $F(\boldsymbol{x}_{0:t} + \Delta\boldsymbol{x}_{0:t})$ with respect to $\Delta\boldsymbol{x}_{0:t}$ and set the differential as zero.

$$\frac{\partial F(\boldsymbol{x}_{0:t} + \Delta\boldsymbol{x}_{0:t})}{\partial \Delta\boldsymbol{x}_{0:t}} \simeq 2\boldsymbol{b}_{0:t} + (H_{0:t} + H_{0:t}^{\mathrm{T}})\Delta\boldsymbol{x}_{0:t} = 0$$

Since $H_{0:t}$ is a symmetric matrix as well (because $\Omega_{0:t}$ is symmetric),

$$\begin{aligned}2\boldsymbol{b}_{0:t} + 2H_{0:t}\Delta\boldsymbol{x}_{0:t} &= 0 \\ \Delta\boldsymbol{x}_{0:t} &= -H_{0:t}^{-1}\boldsymbol{b}_{0:t}\end{aligned}$$

The estimated robot poses can be obtained by adding this increments $\Delta\boldsymbol{x}_{0:t}$ to the initial guess $\boldsymbol{x}_{0:t}$.

$$\boldsymbol{x}'_{0:t} = \boldsymbol{x}_{0:t} + \Delta\boldsymbol{x}_{0:t}$$

Finally, recalculates $H_{0:t}$ and $\boldsymbol{b}_{0:t}$, and **iterates** with the previous result until it is converged.

3.4 Information matrix of the system

Every edge contributes to the system with an addend term. To calculate the addend term in each edge, H_{ij} and $\boldsymbol{b}_{ij}$, separates the Jacobian matrix J_{ij} of the error function $\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ into 2 parts, one is about a robot pose $\boldsymbol{x}_i$ in pose i and the other one is about a robot pose $\boldsymbol{x}_j$ in pose j since the error function depends only on these 2 nodes.

$$J_{ij} = \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\partial(\boldsymbol{x}_i, \boldsymbol{x}_j)} = \begin{pmatrix} \frac{\partial e_{ij_x}}{\partial x_i} & \frac{\partial e_{ij_x}}{\partial y_i} & \frac{\partial e_{ij_x}}{\partial \theta_i} & \frac{\partial e_{ij_x}}{\partial x_j} & \frac{\partial e_{ij_x}}{\partial y_j} & \frac{\partial e_{ij_x}}{\partial \theta_j} \\ \frac{\partial e_{ij_y}}{\partial x_i} & \frac{\partial e_{ij_y}}{\partial y_i} & \frac{\partial e_{ij_y}}{\partial \theta_i} & \frac{\partial e_{ij_y}}{\partial x_j} & \frac{\partial e_{ij_y}}{\partial y_j} & \frac{\partial e_{ij_y}}{\partial \theta_j} \\ \frac{\partial e_{ij_\theta}}{\partial x_i} & \frac{\partial e_{ij_\theta}}{\partial y_i} & \frac{\partial e_{ij_\theta}}{\partial \theta_i} & \frac{\partial e_{ij_\theta}}{\partial x_j} & \frac{\partial e_{ij_\theta}}{\partial y_j} & \frac{\partial e_{ij_\theta}}{\partial \theta_j} \end{pmatrix} = \begin{pmatrix} A_{ij} & B_{ij} \end{pmatrix}$$

where A_{ij} and B_{ij} are the derivatives of the error function $\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ with respect to $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

$$\begin{aligned} A_{ij} &= \begin{pmatrix} \frac{\partial e_{ij_x}}{\partial x_i} & \frac{\partial e_{ij_x}}{\partial y_i} & \frac{\partial e_{ij_x}}{\partial \theta_i} \\ \frac{\partial e_{ij_y}}{\partial x_i} & \frac{\partial e_{ij_y}}{\partial y_i} & \frac{\partial e_{ij_y}}{\partial \theta_i} \\ \frac{\partial e_{ij_\theta}}{\partial x_i} & \frac{\partial e_{ij_\theta}}{\partial y_i} & \frac{\partial e_{ij_\theta}}{\partial \theta_i} \end{pmatrix} \\ &= \begin{pmatrix} \frac{\partial}{\partial \boldsymbol{t}_i} R_{ij}^{\mathrm{T}}\{R_i^{\mathrm{T}}(\boldsymbol{t}_j - \boldsymbol{t}_i) - \boldsymbol{t}_{ij}\} & \frac{\partial}{\partial \theta_i} R_{ij}^{\mathrm{T}}\{R_i^{\mathrm{T}}(\boldsymbol{t}_j - \boldsymbol{t}_i) - \boldsymbol{t}_{ij}\} \\ \frac{\partial}{\partial \boldsymbol{t}_i}\{(\theta_j - \theta_i) - \theta_{ij}\} & \frac{\partial}{\partial \theta_i}\{(\theta_j - \theta_i) - \theta_{ij}\} \end{pmatrix} \\ &= \begin{pmatrix} -R_{ij}^{\mathrm{T}} R_i^{\mathrm{T}} & R_{ij}^{\mathrm{T}} \frac{\partial R_i^{\mathrm{T}}}{\partial \theta_i}(\boldsymbol{t}_j - \boldsymbol{t}_i) \\ 00 & -1 \end{pmatrix} \end{aligned}$$

$$\begin{aligned} B_{ij} &= \begin{pmatrix} \frac{\partial e_{ij_x}}{\partial x_j} & \frac{\partial e_{ij_x}}{\partial y_j} & \frac{\partial e_{ij_x}}{\partial \theta_j} \\ \frac{\partial e_{ij_y}}{\partial x_j} & \frac{\partial e_{ij_y}}{\partial y_j} & \frac{\partial e_{ij_y}}{\partial \theta_j} \\ \frac{\partial e_{ij_\theta}}{\partial x_j} & \frac{\partial e_{ij_\theta}}{\partial y_j} & \frac{\partial e_{ij_\theta}}{\partial \theta_j} \end{pmatrix} \\ &= \begin{pmatrix} \frac{\partial}{\partial \boldsymbol{t}_j} R_{ij}^{\mathrm{T}}\{R_i^{\mathrm{T}}(\boldsymbol{t}_j - \boldsymbol{t}_i) - \boldsymbol{t}_{ij}\} & \frac{\partial}{\partial \theta_j} R_{ij}^{\mathrm{T}}\{R_i^{\mathrm{T}}(\boldsymbol{t}_j - \boldsymbol{t}_i) - \boldsymbol{t}_{ij}\} \\ \frac{\partial}{\partial \boldsymbol{t}_j}\{(\theta_j - \theta_i) - \theta_{ij}\} & \frac{\partial}{\partial \theta_j}\{(\theta_j - \theta_i) - \theta_{ij}\} \end{pmatrix} \\ &= \begin{pmatrix} R_{ij}^{\mathrm{T}} R_i^{\mathrm{T}} & \boldsymbol{0} \\ 00 & 1 \end{pmatrix} \end{aligned}$$

Then, H_{ij} and $\boldsymbol{b}_{ij}$ can be calculated with A_{ij} and B_{ij};

$$H_{ij} = \begin{pmatrix} A_{ij}^{\mathrm{T}} \\ B_{ij}^{\mathrm{T}} \end{pmatrix} \Omega_{ij} \begin{pmatrix} A_{ij} & B_{ij} \end{pmatrix} = \begin{pmatrix} A_{ij}^{\mathrm{T}} \Omega_{ij} A_{ij} & A_{ij}^{\mathrm{T}} \Omega_{ij} B_{ij} \\ B_{ij}^{\mathrm{T}} \Omega_{ij} A_{ij} & B_{ij}^{\mathrm{T}} \Omega_{ij} B_{ij} \end{pmatrix}$$

$$\boldsymbol{b}_{ij} = \begin{pmatrix} A_{ij}^{\mathrm{T}} \\ B_{ij}^{\mathrm{T}} \end{pmatrix} \Omega_{ij} \boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \begin{pmatrix} A_{ij}^{\mathrm{T}} \Omega_{ij} \boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ B_{ij}^{\mathrm{T}} \Omega_{ij} \boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) \end{pmatrix}$$

Thus, the information matrix of the system $H_{0:t}$ and the information vector of the system $\boldsymbol{b}_{0:t}$ can be obtained by adding these sub-matrix H_{ij} and sub-vector $\boldsymbol{b}_{ij}$ respectively in corresponding elements.

$$\begin{aligned} H_{0:t_{[ii]}} &+= A_{ij}^{\mathrm{T}} \Omega_{ij} A_{ij} & H_{0:t_{[ij]}} &+= A_{ij}^{\mathrm{T}} \Omega_{ij} B_{ij} \\ H_{0:t_{[ji]}} &+= B_{ij}^{\mathrm{T}} \Omega_{ij} A_{ij} & H_{0:t_{[jj]}} &+= B_{ij}^{\mathrm{T}} \Omega_{ij} B_{ij} \end{aligned}$$

$$\begin{aligned} \boldsymbol{b}_{0:t_{[i]}} &+= A_{ij}^{\mathrm{T}} \Omega_{ij} \boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \boldsymbol{b}_{0:t_{[j]}} &+= B_{ij}^{\mathrm{T}} \Omega_{ij} \boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) \end{aligned}$$

4 Source code

Belwo are source code snipets of 3D (x, y, θ) and 2D (x, y) ver.

4.1 3D (x, y, θ) ver.

Since $\boldsymbol{x} = (x, y, \theta)^{\mathrm{T}}$, all of the equations in the program is same as discussed in the previous section **3**.

4.1.1 Error and Cost function

```
# Local information matrix `Omega` (from a dataset file)
Omega = edge_ij.info_matrix # 3x3 matrix

# Pose representation matrix `X_i` and
# Rotation matrix `R_i` on Vehicle_i coordinate
X_i = vec2mat(node_i) # 3x3 matrix
R_i = X_i[0:2, 0:2]    # 2x2 matrix

# Pose representation matrix `X_j` on Vehicle_j coordinate
X_j = vec2mat(node_j) # 3x3 matrix

# Observation representation matrix `Z_ij` and
# Rotation matrix `R_ij` on Vehicle_j coordinate
Z_ij = vec2mat(edge_ij.mean) # 3x3 matrix
R_ij = Z_ij[0:2, 0:2]        # 2x2 matrix

# Error between edges `e`
e = mat2vec(Z_ij.I * X_i.I * X_j) # 3x1 matrix
```

4.1.2 Linearization

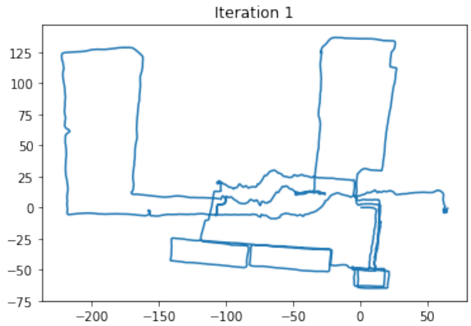
```
# Differentail of `R_i` ... d(R_i)/d(yaw_i)
dR_dyaw_i = np.mat([
    [-s_i, -c_i], # [-sin(yaw_i), -cos(yaw_i)],
    [c_i,  -s_i]  # [ cos(yaw_i), -sin(yaw_i)]
])
# Robot position vector `t_i`, `t_j`
t_i = node_i[0:2, 0] # 2*1 matrix, [x_i, y_i]
t_j = node_j[0:2, 0] # 2*1 matrix, [x_j, y_j]

# Separated Jacobian matrix `A_ij` which is regarding to `x_i`
A = np.mat(np.zeros((3, 3))) # 3x3 matrix with all zeros
A[0:2, 0:2] = -R_ij.T * R_i.T                        # Top left 2x2 elements
A[0:2, 2:3] = R_ij.T * dR_dyaw_i.T * (t_j - t_i) # Top right 2x1 elements
A[2:3, 0:3] = np.mat([0, 0, -1])                     # Bottom 1x3 elements

# Separated Jacobian matrix `B_ij` which is regarding to `x_j`
B = np.mat(np.zeros((3, 3))) # 3x3 matrix with all zeros
B[0:2, 0:2] = R_ij.T * R_i.T     # Top left 2x2 elements
B[0:2, 2:3] = np.mat([0, 0]).T  # Top right 2x1 elements
B[2:3, 0:3] = np.mat([0, 0, 1]) # Bottom 1x3 elements

# Information sub-matrix of the system `H_ii`, `H_ij`, `H_ji`, `H_jj`
H_ii = A.T * Omega * A;     H_ij = A.T * Omega * B
H_ji = B.T * Omega * A;     H_jj = B.T * Omega * B

# Adding the sub-matrix into the information matrix of the system `H`
self.H[i_idx[0]:i_idx[1], i_idx[0]:i_idx[1]]+=H_ii;
self.H[i_idx[0]:i_idx[1], j_idx[0]:j_idx[1]]+=H_ij
self.H[j_idx[0]:j_idx[1], i_idx[0]:i_idx[1]]+=H_ji;
self.H[j_idx[0]:j_idx[1], j_idx[0]:j_idx[1]]+=H_jj

# Information sub-vector of the system `b_i`,`b_j`
b_i = A.T * Omega * e
b_j = B.T * Omega * e

# Adding the sub-vector into the information vector of the system `b`
self.b[i_idx[0]:i_idx[1]] += b_i
self.b[j_idx[0]:j_idx[1]] += b_j
```

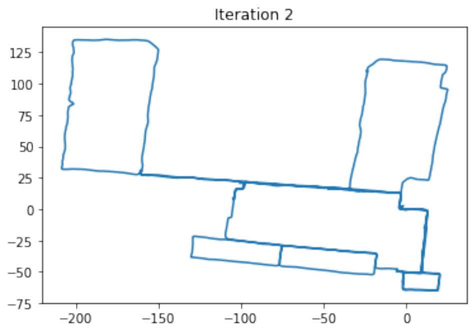
4.1.3 Solve and Update

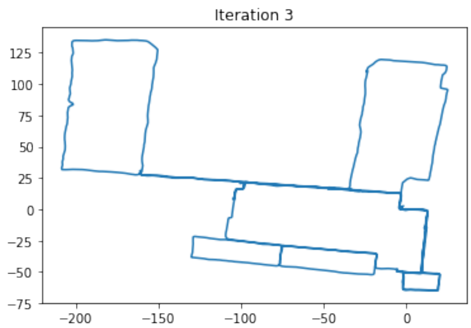
```
# Add an Identity matrix to fix the first pose, `x0` and `y0`, as the origin
H[0:3, 0:3] += np.eye(3)

# Make sparse matrix of `H`
H_sparse = scipy.sparse.csc_matrix(H) # 3`n_node`x3`n_node` matrix

# `H`^-1
H_sparse_inv = scipy.sparse.linalg.splu(H_sparse)

# `dx` = -`H`^-1 * `b`
dx = -H_sparse_inv.solve(self.b) # 3`n_node`x1 matrix

# Reshape
dx = dx.reshape([3, self.n_node], order='F') # 3x`n_node` matrix

# Update
for i in range(self.n_node):
  self.node[i].pose += dx[:, i]
```

4.1.4 Result

Below are the results for 3 iterations of the previous section **3.3**.







One can see that the trajectories which regarded as different paths are restored.

4.2 2D (x, y) ver.

Since $\boldsymbol{x} = (x, y)^{\mathrm{T}}$, the equations in the program are slightly different. The pose representation matrix X is not used because there is no rotation elements. Thus, the error function $\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is just the simple difference between each vector, and the separeted Jacobian matrix A_{ij} and B_{ij} also become simpler. The calculations of the information matrix $H_{0:t}$, information vector $\boldsymbol{b}_{0:t}$, and the solving equation $\Delta\boldsymbol{x}_{0:t} = -H_{0:t}^{\mathrm{T}}\boldsymbol{b}_{0:t}$ are not changed.

4.2.1 Error and Cost function

```
# Coveriance matrix
Sigma = np.mat([[sigma_xx, sigma_xy],
                [sigma_xy, sigma_yy]])

# Local information matrix `Omega`
Omega = Sigma.I

# Error between edges `e`
e = np.mat([[(nodes[Id_j][1] - nodes[Id_i][1]) - dx_ij],  # x
            [(nodes[Id_j][2] - nodes[Id_i][2]) - dy_ij]]) # y
```

4.2.2 Linearization

```
# Separated Jacobian matrix `A_ij` which is regarding to `x_i`
A = np.mat([[-1, 0],
            [0, -1]])

# Separated Jacobian matrix `B_ij` which is regarding to `x_j`
B = np.mat([[1, 0],
            [0, 1]])

# Information sub-matrix of the system `H_ii`, `H_ij`, `H_ji`, `H_jj`
H_ii = A.T * Omega * A;     H_ij = A.T * Omega * B
H_ji = B.T * Omega * A;     H_jj = B.T * Omega * B

# Adding the sub-matrix into the information matrix of the system `H`
H[Id_i*2:(Id_i+1)*2, Id_i*2:(Id_i+1)*2] += H_ii
H[Id_i*2:(Id_i+1)*2, Id_j*2:(Id_j+1)*2] += H_ij
H[Id_j*2:(Id_j+1)*2, Id_i*2:(Id_i+1)*2] += H_ji
H[Id_j*2:(Id_j+1)*2, Id_j*2:(Id_j+1)*2] += H_jj

# Information sub-vector of the system `b_i`,`b_j`
b_i = A.T * Omega * e
b_j = B.T * Omega * e

# Adding the sub-vector into the information vector of the system `b`
b[Id_i*2:(Id_i+1)*2] += b_i
b[Id_j*2:(Id_j+1)*2] += b_j
```

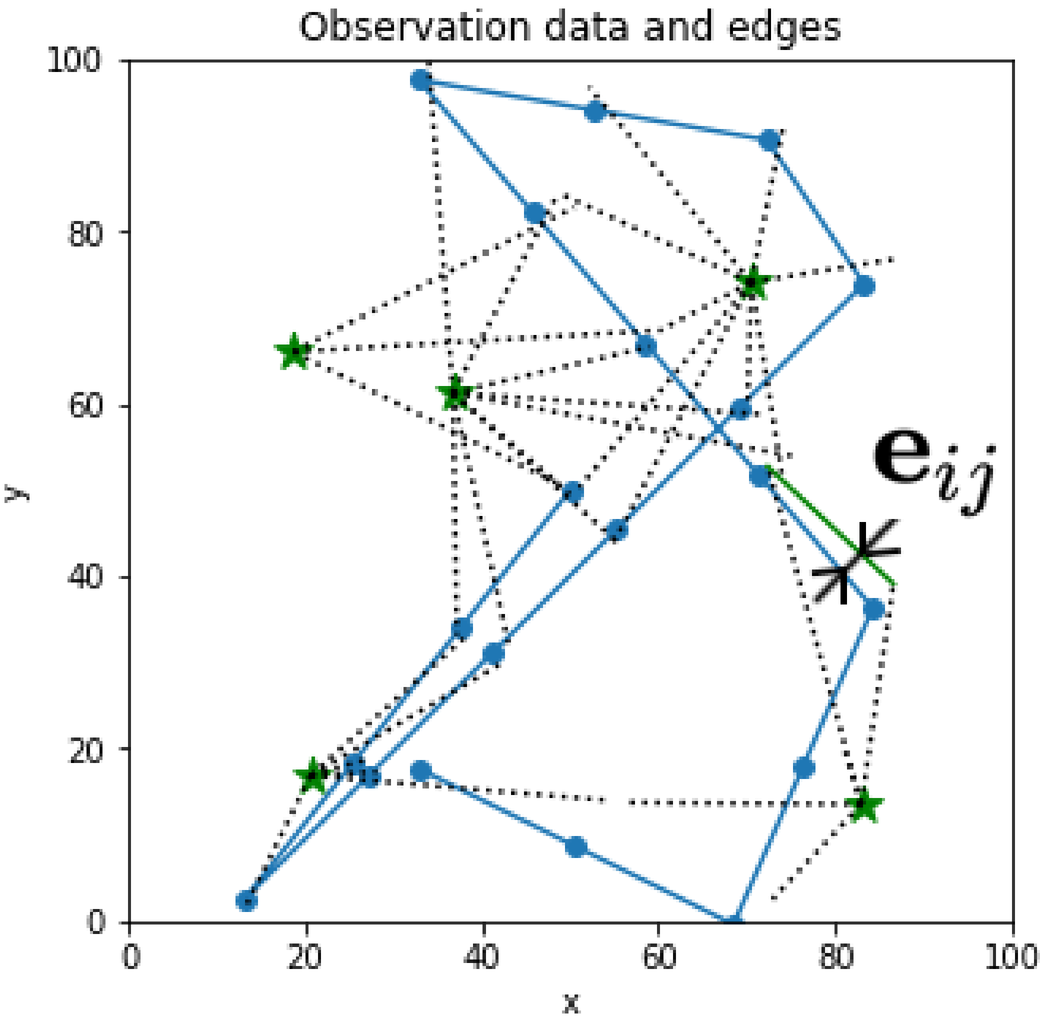
4.2.3 Solve and Update

```
# Add an Identity matrix to fix the first pose, `x0` and `y0`, as the origin
H[0:2, 0:2] += np.eye(2)

# Make sparse matrix of H
H_sparse = scipy.sparse.csc_matrix(H)

# `H`^-1
H_sparse_inv = scipy.sparse.linalg.splu(H_sparse)

# `dx` = -`H`^-1 * `b`
dx = -H_sparse_inv.solve(b)

# Update
for i in range(len(dx)/2):
    nodes[i][1] += dx.item((i*2, 0))    # x
    nodes[i][2] += dx.item((i*2+1, 0)) # y
```

4.2.4 Result

First, below are the trajectory before the optimization (just piling up the motion edges = Dead Reckoning), the observation data, and the observation edges. In this case, an observation edge $\boldsymbol{z}_{ij}$ between pose $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ can be calculated by subtracting each observation data when the robot in $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ see a same landmark.

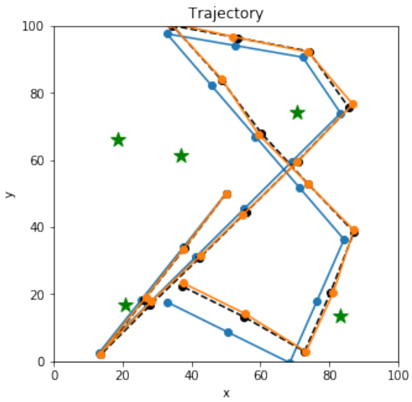
- Blue line: Initial robot trajectory (Dead Reckoning)
- **Black** dashed line: Observation data (not observation edges)
- Green line: Observation edges calculated by observation data
- Star mark: Ground truth position of landmark

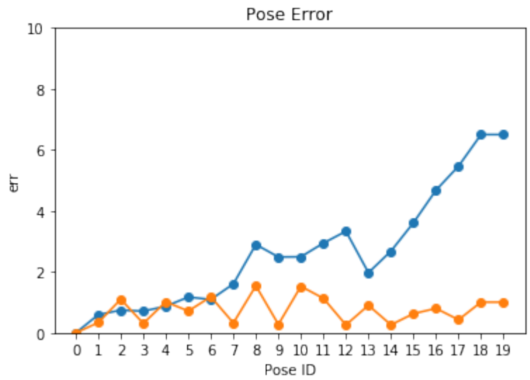


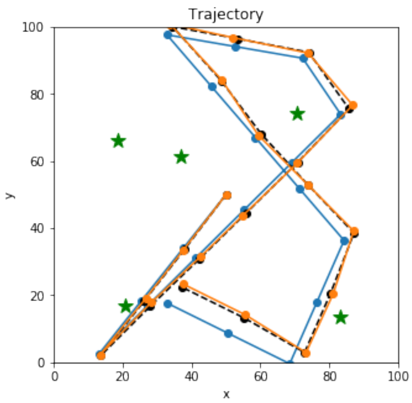
Due to the odometry noise, there are difference between the robot pose inferred from the Dead Reckoning and the observation data (the robot pose from the view of landmarks). The goal of the Graph Based SLAM is to find the trajectory that reduces this difference (the error fucntion $\boldsymbol{e}_{ij}$).

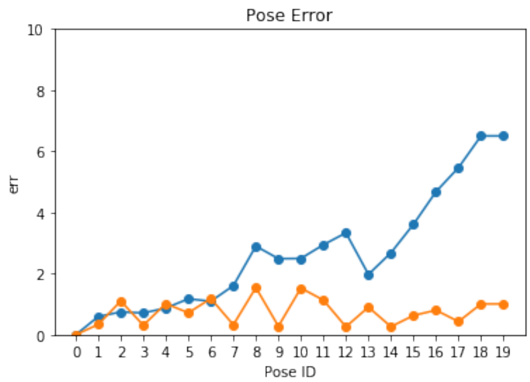
Next, below are the results for 3 iterations of the previous section **3.3** and the pose error against the ground truth in each pose.

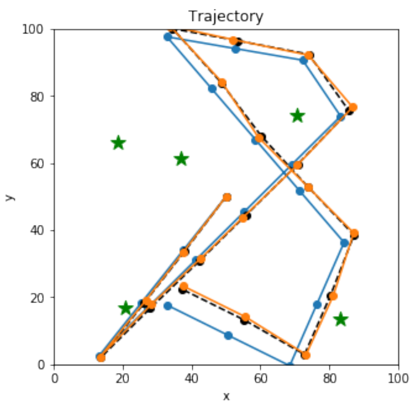
- Blue line: Initial robot trajectory (Dead Reckoning)
- Orange line: Optimized robot trajectory
- **Black** dashed line: Ground truth of the robot trajectory
- Star mark: Ground truth position of landmark

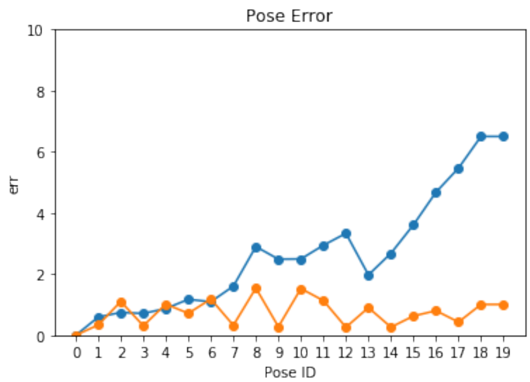






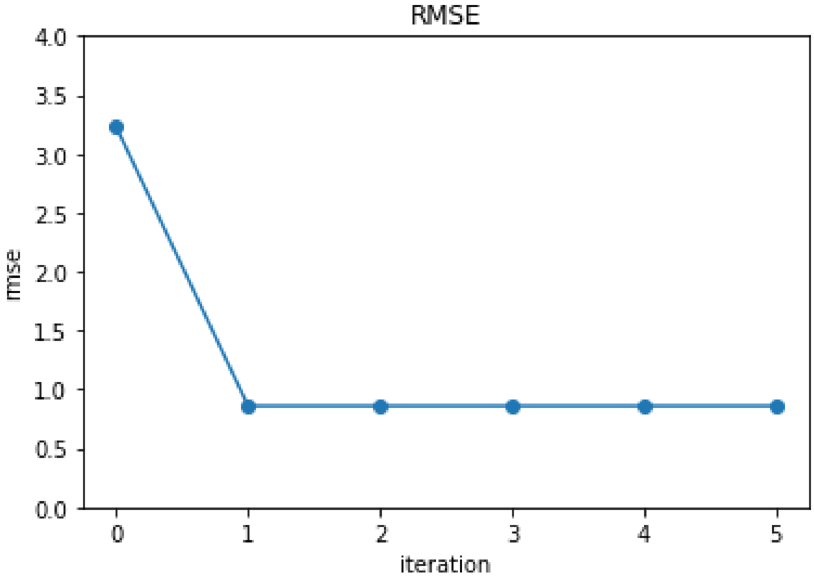






One can see that the error between the optimized robot trajectory (Orange line) and the ground truth (**Black** dashed line) is smaller than the initail one (Blue line). Also, the pose error of the initial robot trajectory by Dead Reckoning (Blue line) increases as the robot moves.

Finally, below is the RMSE for 5 iterations of the previous section **3.3**



One can see that it is converged at the 1st iteration because the system is linear due to omitting θ.

References

[1] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM", IEEE Intelligent Transportation Systems Magazine, 2010.

[2] GitHub - deleji/graph-slam: 一个二□平面的激光□□化例子, `https://github.com/deleji/graph-slam`

[3] Udacity - Artificial Intelligence for Robotics: Implementing SLAM, `https://www.udacity.com/course/artificial-intelligence-for-robotics--cs373`