Exercise 3.0 — Pose Graph Optimization and Iterative Closest Point

Artificial Intelligence For Robotics

Timo Hinzmann

Spring Semester 2017

1 Pose Graph Optimization (PGO)

In this exercise you will implement a pose graph optimization algorithm to estimate the position of a robot based on wheel odometry and loop closure constraints. The repetition in Sec. 1.1 is based on [1] and [2].

1.1 Repetition: PGO

Recall that the edges of a pose graph encode the constraints between two nodes i and j given an observation z_{ij}. Based on the current pose estimates stored in node i and j we can formulate the error as:

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij} \tag{1}$$

The goal of maximum likelihood pose graph optimization is to minimize the negative log-likelihood given all constraints

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{<i,j>\in\mathcal{C}} \mathbf{e}_{ij}^\top \Omega_{ij} \mathbf{e}_{ij} \tag{2}$$

where $\mathbf{x}^*$ are the optimized poses, $\mathcal{C}$ is the set of pairwise constraints from node i to node j and Ω_{ij} is the corresponding measurement information matrix. The error function can be approximated with a first order Taylor expansion given an initial guess $\breve{\mathbf{x}}$ and a perturbation $\Delta\mathbf{x}$:

$$\mathbf{e}_{ij}(\breve{\mathbf{x}} + \Delta\mathbf{x})^\top \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}(\breve{\mathbf{x}} + \Delta\mathbf{x}) \tag{3}$$

$$\approx (\mathbf{e}_{ij} + \mathbf{J}_{ij}\Delta\mathbf{x})^\top \boldsymbol{\Omega}_{ij} (\mathbf{e}_{ij} + \mathbf{J}_{ij}\Delta\mathbf{x}) \tag{4}$$

$$= \underbrace{\mathbf{e}_{ij}^\top \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}}_{c_{ij}} + 2 \underbrace{\mathbf{e}_{ij}^\top \boldsymbol{\Omega}_{ij} \mathbf{J}_{ij}}_{\mathbf{b}_{ij}} \Delta\mathbf{x} + \Delta\mathbf{x}^\top \underbrace{\mathbf{J}_{ij}^\top \boldsymbol{\Omega}_{ij} \mathbf{J}_{ij}}_{\mathbf{H}_{ij}} \Delta\mathbf{x} \tag{5}$$

where we defined $\mathbf{e_{ij}} := \mathbf{e_{ij}}(\breve{\mathbf{x}})$. Minimization of Eq. 5 translates into solving the linear system $\mathbf{H}\Delta\mathbf{x}^* = -\mathbf{b}$ using sparse Cholesky factorization. The linearized solution is then given as $\mathbf{x}^* = \breve{\mathbf{x}} + \Delta\mathbf{x}^*$.

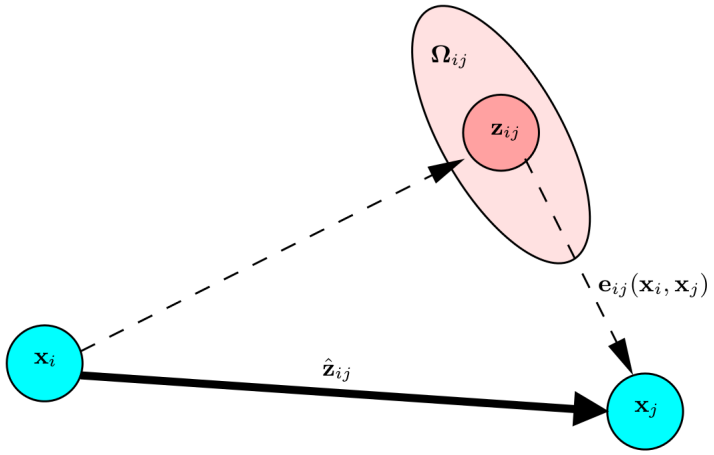

Figure 1: The measurement z_{ij} and entry of the information matrix Ω_{ij} results in a constraint between node i and j. The expected observation $\hat{z}_{ij}$ is computed from the current estimate x_i and x_j. [Figure taken from [1]]

1.2 Assignments

1.2.1 Synthetic Dataset: Robot with odometry and loop closure constraints in 1D

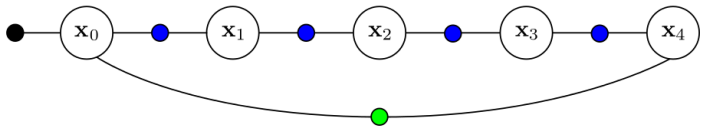

Figure 2: Pose graph in factor graph formulation. The constraints consist of wheel odometry (blue), loop closure (green) and fixed first node (black). [Figure taken from [2]]

For simplicity, we consider the problem in one dimension and estimate only the position in x-direction, i.e. the state is $\mathbf{x} := \begin{bmatrix} x \end{bmatrix}$.

Tasks: Follow the steps described in [1], Algorithm 1 and fill in the sections marked with `#TODO` in `pgo_1D.py`. In particular:

1. Compute the residual $\mathbf{e}_{ij}$ as well as Jacobians $\mathbf{A}_{ij}$ and $\mathbf{B}_{ij}$ for
   (a) Wheel odometry: Returns the measured distance u_{ij} from node i to node j.
   (b) Loop closure: The loop closure constraint can be interpreted as an odometry constraint from node i to node j with $u_{ij} = 0$.
   (c) Fixation of first node: We fix the first pose to the origin which is (arbitrarily) selected as starting point.

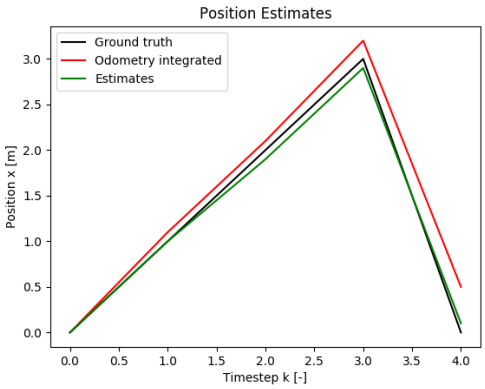

Figure 3: Sample position output: Ground truth, integrated odometry, fused integrated odometry and loop closure constraints.

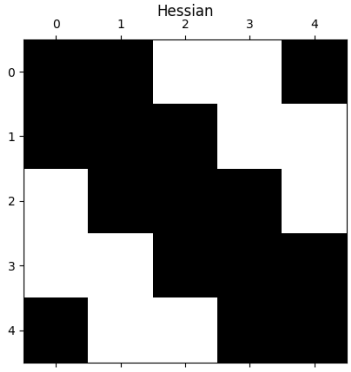

Figure 4: Hessian showing the odometry and loop closure constraints. Black squares are non-zero matrix entries.

2. Set the corresponding values of the measurement information matrix:
   - Wheel odometry: $\Omega_{ij} = 100$
   - Loop closure: $\Omega_{ij} = 100$
   - Fixation of first node: $\Omega_{ij} = 1000$
3. Implement the Gauss-Newton based pose graph optimization which computes $\mathbf{H}$, $\mathbf{b}$ and solves for $\Delta\mathbf{x}$ using sparse Cholesky factorization. The positions estimated by your pose graph optimization algorithm are automatically stored to `results_1D.txt`.

1.2.2 Synthetic Dataset: Robot with odometry and loop closure constraints in 2D

In this assignment we augment the problem from the 1D to the 2D case. The state of the robot is now $\mathbf{x} := \begin{bmatrix} x & y & \theta \end{bmatrix}$.

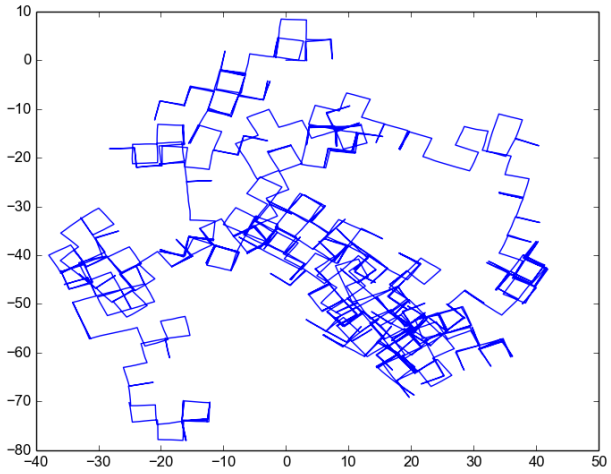

Figure 5: Initial guess for x and y computed from odometry.

The input data is given as follows:

- `vertices.dat`:
  `vertex_id x y theta`
- `edges.dat`:
  `vertex_id_from vertex_id_to dx dy dth O11 O12 O22 O33 O13 O23`
- `loop_closures.dat`:
  `vertex_id_from vertex_id_to dx dy dth O11 O12 O22 O33 O13 O23`

where dx, dy, dth is the odometry measurement between two vertices and Ω_{ij} is the entry of the information matrix defined as

$$\mathbf{\Omega} = \begin{bmatrix} \Omega_{11} & \Omega_{12} & \Omega_{13} \\ \Omega_{12} & \Omega_{22} & \Omega_{23} \\ \Omega_{13} & \Omega_{23} & \Omega_{33} \end{bmatrix} \tag{6}$$

The edge and loop closure constraints are obtained from wheel and laser odometry as well as laser scan matching respectively. Fig. 5 shows the initial guess for x and y computed from the odometry.

Tasks: Follow the steps described in [1], Algorithm 1 and fill in the sections marked with `#TODO` in `pgo_2D.py`. The optimized states $\hat{\mathbf{x}} := \begin{bmatrix} \hat{x} & \hat{y} & \hat{\theta} \end{bmatrix}$ are automatically saved to `results_2D.txt`. Feel free to reuse the code from Sec. 1.2.1.

2 Pointcloud Registration

In the first assignment you will implement the point-to-point iterative closest point (ICP) algorithm to register a synthetic source pointcloud to a target pointcloud. In the second assignment you will implement a pointcloud registration algorithm of your choice to register a real-world vision pointcloud to a laser pointcloud. The repetition about ICP in Sec. 2.1 is based on [3].

2.1 Repetition: ICP

The goal of ICP is to find the transformation matrix $\mathbf{T}$ that best transforms all points from source to target pointcloud. Several ICP variants exist - such as point-to-point, point-to-plane and generalized ICP. In the first assignment, we focus on point-to-point ICP and follow the algorithm as described in Algorithm 2.

Algorithm 2 Point-to-point ICP

input: Two pointclouds: source $\mathbf{p}_s$, target $\mathbf{p}_t$; an initial transformation $\hat{\mathbf{T}}$
output: The optimized transformation $\hat{\mathbf{T}}$ which aligns source and target pointcloud.
$\mathbf{p}_t^c$: Nearest neighbors in target cloud for every point in source cloud.

1: **function** COMPUTEBESTTRANSFORMATION($\mathbf{p}_s$,$\mathbf{p}_t$)
2: $\bar{\mathbf{p}}_s \leftarrow$computeCentroid($\mathbf{p}_s$)
3: $\bar{\mathbf{p}}_t \leftarrow$computeCentroid($\mathbf{p}_t$)
4: $\begin{bmatrix}\mathbf{U} & \mathbf{D} & \mathbf{V}^\top\end{bmatrix} \leftarrow$SVD($\sum_i (\mathbf{p}_{s,i} - \bar{\mathbf{p}}_s)(\mathbf{p}_{t,i} - \bar{\mathbf{p}}_t)^\top$)
5: $\hat{\mathbf{R}} = \mathbf{U}^\top \mathbf{V}$
6: $\hat{\mathbf{t}} = \bar{\mathbf{p}}_t - \hat{\mathbf{R}}\bar{\mathbf{p}}_s$
7: $\hat{\mathbf{T}} \leftarrow$ createTransformationMatrix($\hat{\mathbf{t}}, \hat{\mathbf{R}}$)
8: **end function**

9: **function** ICP($\mathbf{p}_t$,$\mathbf{p}_s$, $\hat{\mathbf{T}}$)
10: $\mathbf{p}_s^{orig} \leftarrow \mathbf{p}_s$ ▷ Save original source pointcloud for line 16.
11: $\mathbf{p}_s \leftarrow$ transformCloud($\mathbf{p}_s, \hat{\mathbf{T}}$)
12: **while** not converged **do**
13: rejectPointsTooFarAway() ▷ [Optional]
14: $\mathbf{p}_t^c \leftarrow$ getNearestNeighbors($\mathbf{p}_s, \mathbf{p}_t$)
15: $\hat{\mathbf{T}} \leftarrow$ computeBestTransformation($\mathbf{p}_s$,$\mathbf{p}_t^c$)
16: $\mathbf{p}_s \leftarrow$ transformCloud($\mathbf{p}_s, \hat{\mathbf{T}}$)
17: **end while**
18: $\hat{\mathbf{T}} \leftarrow$ computeBestTransformation($\mathbf{p}_s^{orig}$,$\mathbf{p}_s$)
19: **end function**

2.2 Assignments

2.2.1 Synthetic Dataset

In Fig. 6 you can see the (misaligned) source pointcloud (red) and the target pointcloud (green). The goal is to estimate the transformation $\mathbf{T}$ that best aligns the source to the target pointcloud as shown in Fig. 7 . Note that these are the same pointclouds, generated by applying a transformation $\mathbf{T}$, i.e. there exist unique point-to-point correspondences which is an underlying assumption of ICP.

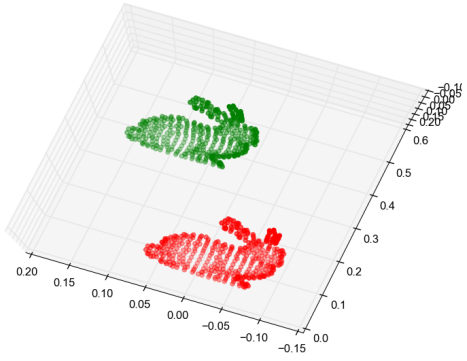

Figure 6: Misaligned pointclouds: source (red) and target (green).

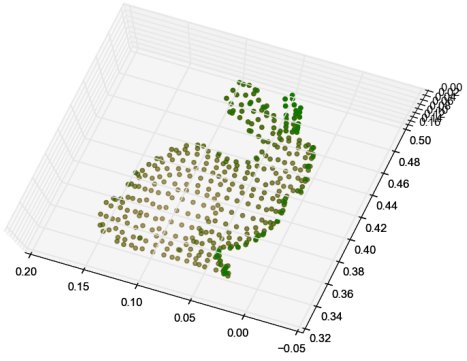

Figure 7: Aligned pointclouds: source (red) and target (green).

Task: Follow the steps described in Algorithm 2 and modify the sections marked with `#TODO` in `results_alignment_synthetic.py`. The program will automatically save the final transformation $\mathbf{T}$ which will be used for grading.

2.2.2 Real-World Dataset: Vision-laser alignment

In this assignment you will have to register a pointcloud obtained from a camera to a laser pointcloud (cf. Fig. 8 and 9). Notice that the difference to Sec. 2.2.1 is that, in general, unique point-to-point matches do not exist. Your task is to implement a robust pointcloud registration algorithm of your choice that aligns the vision to the laser pointcloud by modifying the

Figure 8: Platforms used for point-cloud generation: Fixed-wing (Vision) and helicopter (Laser) UAV. [Figure taken from [4]]

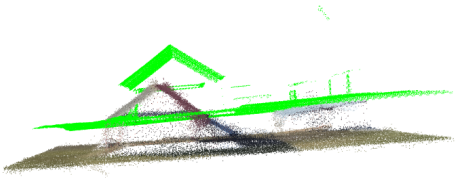

Figure 9: Misaligned source (vision, colored) and target (laser, green) pointcloud. [Figure taken from [4]]

`#TODO`s in `alignment_real_world.py`. The final transformation $\mathbf{T}$ is stored in `results_alignment_real_world.txt`. Feel free to reuse the code from Sec. 2.2.1.

3 Submission

To hand in the exercise you have to zip the whole folder `(3_0_pgo_icp)` with your code and result files and upload it on moodle. The zip file you upload should have the name *aifr_lastname_ex3* (replace *lastname* with your last name). The python scripts are written in a way, such that it always creates a `results*.txt` or `results*.pkl` file with the data that is used for evaluation. **Please make sure that your python scripts are executable**.

References

[1] G. Grisetti, R. Kuemmerle, C. Stachniss, *et al.*, "A tutorial on graph-based SLAM," *Intelligent Transportation Systems Magazine, IEEE*, vol. 2, no. 4, pp. 31–43, 2010.

[2] N. Sünderhauf, *Robust Optimization for Simultaneous Localization and Mapping*. PhD thesis, 2012.

[3] C. Cadena, I. Gilitschenski, and R. Siegwart, "Artifical intelligence for robotics: Pose graph optimization & iterative closest points." Lecture Notes, Lecture 7.

[4] T. Hinzmann, T. Stastny, G. Conte, *et al.*, "Collaborative 3D Reconstruction using Heterogeneous UAVs: System and Experiments," in *Experimental Robotics - The 15th International Symposium on Experimental Robotics, ISER 2016, October 3-6, 2016, Tokyo, Japan*, 2016.