Albert-Ludwigs-Universität Freiburg, Institut für Informatik
Lecture: Robot Mapping by Prof. Dr. Wolfram Burgard
Tutors: Dr. Daniel Büscher, Dr. Lukas Luft, Shengchao Yan, Kshitij Sirohi
Winter term 2020/21

Sheet 10

Topic: Graph-Based SLAM

Due: February 2, 2021

Exercise: Graph-Based SLAM

Implement a least-squares method to address SLAM in its graph-based formulation. To support this task, we provide a small Octave framework on the course website. The framework consists of the following folders:

data contains several datasets, each gives the measurements of one SLAM problem.

octave contains the Octave framework with stubs to complete.

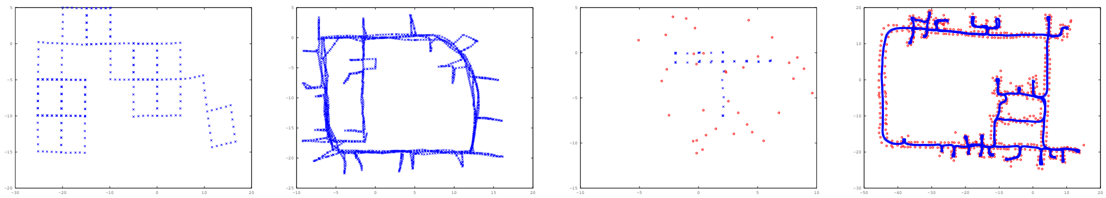
plots this stores the resulting images.

The tasks mentioned below should be implemented inside the framework in the directory `octave` by completing the stubs:

1.
   - Implement the function in `compute_global_error.m` for computing the current error value for a graph with constraints.
   - Implement the function in `linearize_pose_pose_constraint.m` for computing the error and the Jacobian of a pose-pose constraint. Test your implementation with `test_jacobian_pose_pose`.
   - Implement the function in `linearize_pose_landmark_constraint.m` for computing the error and the Jacobian of a pose-landmark constraint. Test your implementation with `test_jacobian_pose_landmark`.

2.
   - Implement the function in `linearize_and_solve.m` for constructing and solving the linear approximation.
   - Implement the update of the state vector and the stopping criterion in `lsSLAM.m`. A possible choice for the stopping criterion is $\|\Delta\mathbf{x}\|_\infty < \epsilon$, i.e., $\|\Delta\mathbf{x}\|_\infty = \max(|\Delta x_1|, \ldots, |\Delta x_n|) < \epsilon$.

After implementing the missing parts, you can run the framework. To do that, change into the directory `octave` and launch Octave. To start the main loop, type `lsSLAM`. The script will produce a plot showing the positions of the robot and (if available) the positions of the landmarks in each iteration. These plots will be saved in the `plots` directory.



simulation-pose-pose | intel | simulation-pose-landmark | dlr

Figure 1: Result for each dataset.

Figure 1 depicts the result that you should obtain after convergence for each dataset. Additionally, the initial and the final error for each dataset should be approximately:

dataset	initial error	final error
simulation-pose-pose.dat	138862234	8269
intel.dat	1795139	360
simulation-pose-landmark.dat	3030	474
dlr.dat	369655336	56860

The state vector contains the following entities:

- Pose of the robot: $\mathbf{x}_i = (x_i \; y_i \; \theta_i)^T$
  Hint: You may use the function $\mathrm{v2t}(\cdot)$ and $\mathrm{t2v}(\cdot)$:

$$\mathrm{v2t}(\mathbf{x}_i) = \begin{pmatrix} R_i & \mathbf{t}_i \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta_i) & -\sin(\theta_i) & x_i \\ \sin(\theta_i) & \cos(\theta_i) & y_i \\ 0 & 0 & 1 \end{pmatrix} = X_i$$

$$\mathrm{t2v}(X_i) = \mathbf{x}_i$$

- Position of a landmark: $\mathbf{x}_l = (x_l \; y_l)^T$

We consider the following error functions:

- Pose-pose constraint: $\mathbf{e}_{ij} = \mathrm{t2v}(Z_{ij}^{-1}(X_i^{-1}X_j))$, where $Z_{ij} = \mathrm{v2t}(\mathbf{z}_{ij})$ is the transformation matrix of the measurement $\mathbf{z}_{ij}^T = (\mathbf{t}_{ij}^T, \theta_{ij})$.
  Hint: For computing the Jacobian, write the error function with rotation matrices and translation vectors:

$$\mathbf{e}_{ij} = \begin{pmatrix} R_{ij}^T(R_i^T(\mathbf{t}_j - \mathbf{t}_i) - \mathbf{t}_{ij}) \\ \theta_j - \theta_i - \theta_{ij} \end{pmatrix}$$

- Pose-landmark constraint: $\mathbf{e}_{il} = R_i^T(\mathbf{x}_l - \mathbf{t}_i) - \mathbf{z}_{il}$