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 $\| \Delta x \|_\infty < \epsilon$, i.e., $\| \Delta x \|_\infty = \max (|\Delta x_1|, \ldots, |\Delta x_n|) < \epsilon$.

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.
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:

<table>
<thead>
<tr>
<th>dataset</th>
<th>initial error</th>
<th>final error</th>
</tr>
</thead>
<tbody>
<tr>
<td>simulation-pose-pose.dat</td>
<td>138862234</td>
<td>8269</td>
</tr>
<tr>
<td>intel.dat</td>
<td>1795139</td>
<td>360</td>
</tr>
<tr>
<td>simulation-pose-landmark.dat</td>
<td>3030</td>
<td>474</td>
</tr>
<tr>
<td>dlr.dat</td>
<td>369655336</td>
<td>56860</td>
</tr>
</tbody>
</table>

The state vector contains the following entities:

- **Pose of the robot**: \( x_i = (x_i, y_i, \theta_i)^T \)
  
  Hint: You may use the function \( v2t(\cdot) \) and \( t2v(\cdot) \):

  \[
  v2t(x_i) = \begin{pmatrix} R_i & t_i \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta_i) & -\sin(\theta_i) & x_i \\ \sin(\theta_i) & \cos(\theta_i) & y_i \\ 0 & 0 & 1 \end{pmatrix} = X_i
  \]

  \[
  t2v(X_i) = x_i
  \]

- **Position of a landmark**: \( x_l = (x_l, y_l)^T \)

We consider the following error functions:

- **Pose-pose constraint**: \( e_{ij} = t2v(Z_{ji}^{-1}(X_i^{-1}X_j)) \), where \( Z_{ij} = v2t(z_{ij}) \) is the transformation matrix of the measurement \( z_{ij}^T = (t_{ij}^T, \theta_{ij}) \).

  Hint: For computing the Jacobian, write the error function with rotation matrices and translation vectors:

  \[
  e_{ij} = \begin{pmatrix} R_{ij}^T(R_i^T(t_j - t_i) - t_{ij}) \\ \theta_j - \theta_i - \theta_{ij} \end{pmatrix}
  \]

- **Pose-landmark constraint**: \( e_{il} = R_i^T(x_l - t_i) - z_{il} \)