Trabajo Práctico Final: Graph Based SLAM
Inteligencia Artificial para Robots - CEIA

6 de octubre de 2021

1. Descripción

Este TP ha sido adaptado de lo publicado en las clases de Cyrill Stachniss en la universidad de Bonn (link). El mismo, por el momento, se encuetra completamente para realizar en Octave. En este documento se describirán los pasos necesarios para guiar a quien quiera resolverlo.

2. Problema a resolver

Implementar el método de SLAM basado en grafos utilizando lo visto en clase. Para realizar esta tarea se provee un framework en *Octave* (ver carpeta **slam**). Ese framework contiene:

- **data** donde hay varios datasets (elergir uno), cada uno de estos da las mediciones de un problema de SLAM.
- **octave** contiene el framework en *Octave* con partes para completar.
- **plots** en esta carpeta se guardarán los gráficos.

Para poder realizar el trabajo, se deben completar los siguientes archivos en el directorio **octave**:

1.
   - Implementar una función en `compute_global_error.m` para calcular el valor de error actual para un grafo con restricciones.
   - Implementar la función en `linearize_pose_pose_constraint.m` para calcular el error y el Jacobiano de una restricción posición-posición. Probar tu implementación con `test_jacobian_pose_pose`.
   - Implementar la función en `linearize_pose_landmark_constraint.m` para calcular el error y el Jacobiano de una restricción posición-referencia. Probar tu implementación con `test_jacobian_pose_landmark`.
2.
   - Implementar la función `linearize_and_solve.m` para construir y resolver la aproximación lineal.
   - Implementar la actualización del vector de estados y el criterio de detención en `lsSLAM.m`. Una posible elección para el criterio de detención es $||\Delta x||_\infty < \epsilon$, i.e., $||\Delta x||_\infty = max(|\Delta x_1|, ..., |\Delta x_n|) < \epsilon$.

Luego de implementar las partes faltantes, se puede correr el framework. Para hacerlo, situar el directorio de trabajo de *Octave* en la carpeta octave y correr `lsSLAM`. El script mostrará un gráfico con las posiciones del robot y (si están disponibles) las posiciones de las referencias en cada iteración. Esos gráficos serán guardados en la carpeta plot.

Los errores iniciales y finales de cada dataset deberían ser aproximadamente:

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

El vector de estado contiene lo siguiente:

- posición del robot: $\mathbf{x}_i = (x_i, y_i, \theta_i)^T$

  Hint: podrías estar interesado en utilizar las funciones $v2t(.)$ y $t2v(.)$:

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

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

- posición de una referencia: $\mathbf{x}_l = (x_l, y_l)$

Se consideran las siguientes funciones de error:

- restricción posición - posición: $\mathbf{e}_{ij} = t2v(Z_{ij}^{-1}(X_i^{-1}X_j))$, donde $Z_{ij} = v2t(\mathbf{z}_{ij})$ es la matriz de transformación de la medición $\mathbf{z}_{ij}^T = (\mathbf{t}_{ij}^T, \theta_{ij})$.

  Hint: para calcular el Jacobiano, escribir la función de error con matrices de rotación y vectores de traslación:

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

- restriccón posición-referencia: $\mathbf{e}_{il} = R_i^T(\mathbf{x}_l - \mathbf{t}_i) - z_{il}$