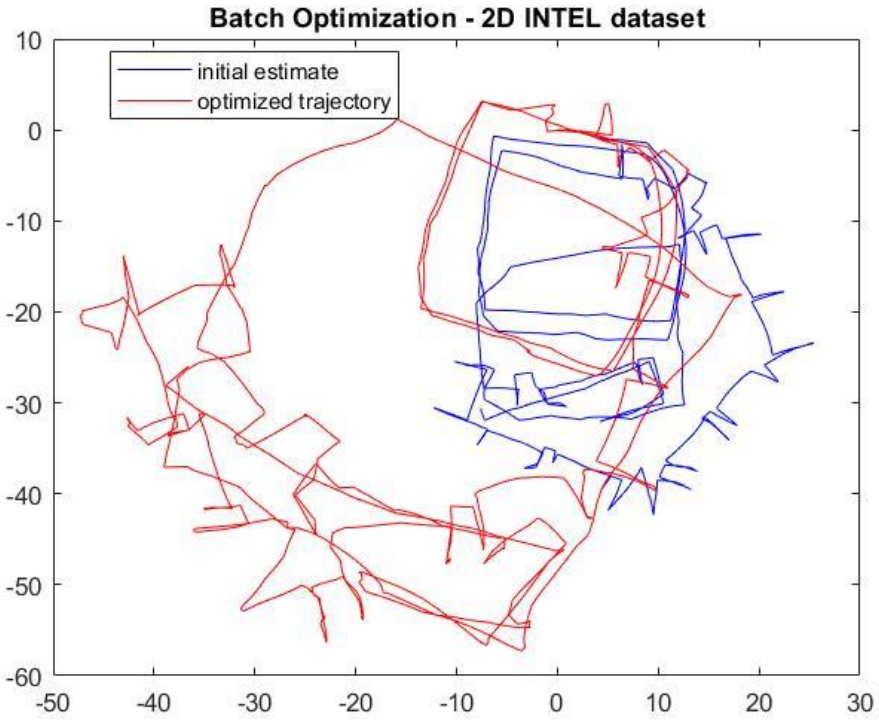
Mobile Robotics: Problem Set 4

Note: The terms 'edges' and 'factors' have been used interchangeable in this document, since the edges of the graph represent the factors/constraints. Similarly, terms 'vertices' and 'nodes' have been used interchangeably. These vertices represent the pose estimates.

Problem 1B: Batch Solution in 2D

Graph Construction Process and Parameters:

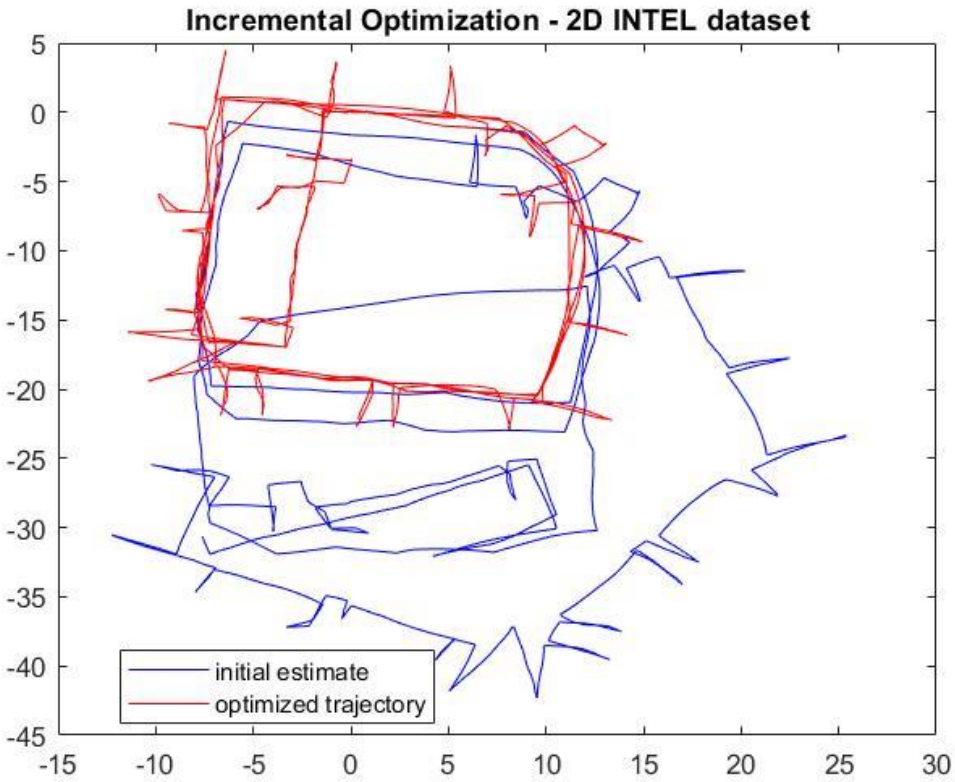
1. Read the data from g2o file and store it separately in `edgeMatrix` and `vertexMatrix`.
2. Initialize objects of classes `NonlinearFactorGraph` and `Values` to store the edges (constraints) and vertices (initial pose estimates).
3. Add a prior factor to the `graph` to constrain node 0 to the origin. Add a small noise to this edge.
4. Iterate through all the edges in `vertexMatrix` and add them to the `initialEstimate` (object of class `Values`). The vertices simply contain mean of the estimate. Mean in 2D is stored in a `Pose2` object which contains `[x, y, theta]`.
5. Iterate through all the edges in `edgeMatrix` and add them to the `graph` (object of class `NonlinearFactorGraph`). The edges contain mean and covariance information. The mean is provided directly in the g2o file but covariance matrix has to be generated using the `info2Cov` function (which constructs the information matrix using the upper triangle elements, inverts it to form the covariance matrix, and returns the lower Cholesky of the covariance matrix).
6. Use `GaussNewtonOptimizer` to smoothen the map and store it in `result`.
7. Plot the results and initial estimate.



Problem 1C: Incremental Solution in 2D

Graph Construction Process and Parameters:

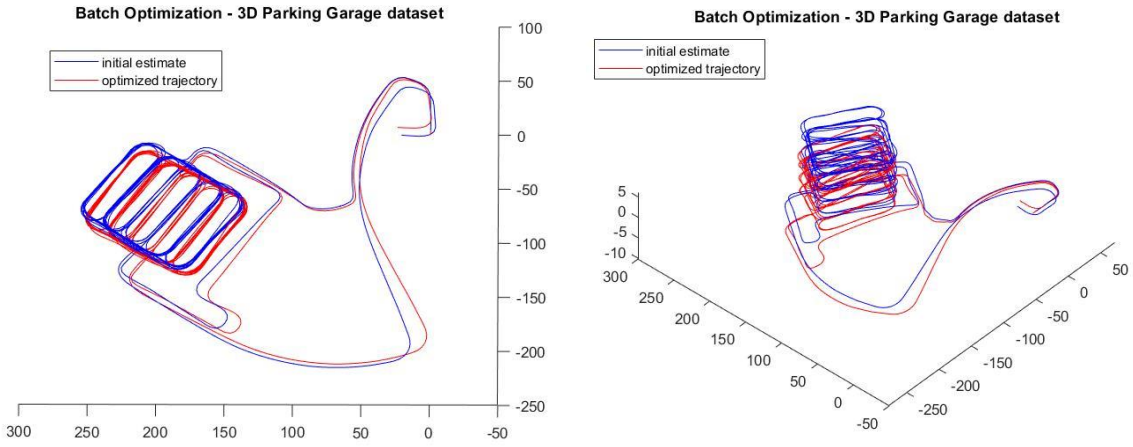
1. Read the data from g2o file and store it separately in `edgeMatrix` and `vertexMatrix`.
2. Initialize objects of classes `NonlinearFactorGraph` and `Values` to store the edges (constraints) and vertices (initial pose estimates).
3. Initialize an object of the `ISAM2` class. Call this object `isam`.
4. Add a prior factor to the `graph` to constrain node 0 to the origin. Add a small noise to this edge.
5. Add Node 0 to the `initialEstimate`.
6. Update the `isam` object and calculate the estimate using `update` and `calculateEstimate` functions of the `ISAM2` class. Store it in `result`.
7. Now iterate through the rest of the vertices in do the following for each vertex:
   7.1. Clear `initialEstimate` and re-initialize `graph`.
   7.2. Add this vertex to `initialEstimate`. Instead of getting the mean information of the vertex from the g2o file, calculate it by adding the corresponding odometry edge mean to the mean of the last vertex in `result`.
   7.3. Now find all edges that have this vertex as their second vertex (i.e. the other vertex of this edge should already be addressed in a previous iteration) and add them to the `graph`.
   7.4. Update the `isam` object and calculate the new estimate and store it in `result`.
8. Plot the results and the initial estimate.



Problem 2B: Batch Solution in 3D

Graph Construction Process and Parameters:

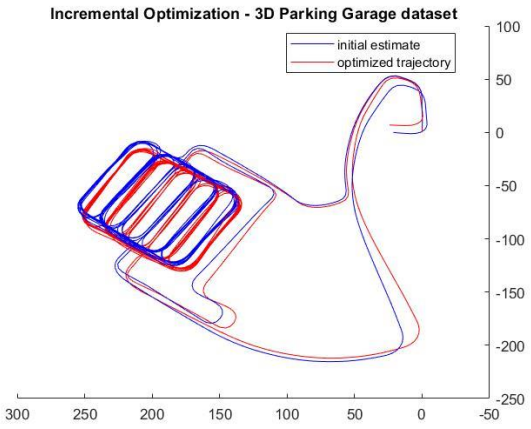
1. Read the data from g2o file and store it separately in `edgeMatrix` and `vertexMatrix`.
2. Initialize objects of classes `NonlinearFactorGraph` and `Values` to store the edges (constraints) and vertices (initial pose estimates).
3. Add a prior factor to the `graph` to constrain node 0 to the origin. Add a small noise to this edge.
4. Iterate through all the edges in `vertexMatrix` and add them to the `initialEstimate` (object of class `Values`). The vertices simply contain mean of the estimate. Mean in 3D is stored in a `Pose3` object which contains `(t,R)` where `t` is the 3D translation vector (an object of the `Point3` class) which can be directly obtained from the g2o file, and `R` is the rotation matrix (an object of the `Rot3` class). The g2o file provides rotation information in quaternion format which can be converted into a rotation matrix using MATLAB's `quat2rotm` function.
5. Iterate through all the edges in `edgeMatrix` and add them to the `graph` (object of class `NonlinearFactorGraph`). The edges contain mean and covariance information. The covariance matrix is a 6X6 matrix in 3D and can be calculated using `info2Cov` function as before.
6. Use `GaussNewtonOptimizer` to smoothen the map and store it in `result`.
7. Plot the results and initial estimate.

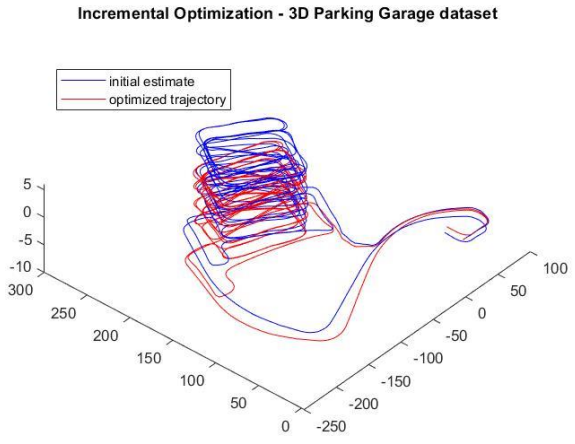


Problem 2C: Incremental Solution in 3D

Graph Construction Process and Parameters:

1. Read the data from g2o file and store it separately in `edgeMatrix` and `vertexMatrix`.
2. Initialize objects of classes `NonlinearFactorGraph` and `Values` to store the edges (constraints) and vertices (initial pose estimates).
3. Initialize an object of the `ISAM2` class. Call this object `isam`.
4. Add a prior factor to the `graph` to constrain node 0 to the origin. Add a small noise to this edge.
5. Add Node 0 to the `initialEstimate`.
6. Update the `isam` object and calculate the estimate using `update` and `calculateEstimate` functions of the `ISAM2` class. Store it in `result`.
7. Now iterate through the rest of the vertices in do the following for each vertex:
    a. Clear `initialEstimate` and re-initialize `graph`.
    b. Add this vertex to `initialEstimate`. Instead of getting the mean information of the vertex from the g2o file, calculate it by adding the corresponding odometry edge mean to the mean of the last vertex in `result`. In 3D case, the `[x y z]` can be added directly but in order to combine the rotation matrices, use the `compose` function belonging to the `Rot3` class.
    c. Now find all edges that have this vertex as their second vertex (i.e. the other vertex of this edge should already be addressed in a previous iteration) and add them to the `graph`.
    d. Update the `isam` object and calculate the new estimate and store it in `result`.
8. Plot the results and the initial estimate.





Problem 1A and 2A – Reading g2o files

2D g2o file:

- In 2D g2o file, each line contains 12 space separated objects. The first object in each line is a string containing either `VERTEX_SE2` and `EDGE_SE2`, and rest of the 11 objects can be interpreted as floating point numbers (although they might be integers).
- We can read the lines using `textscan` function of MATLAB.
- We classify these lines according to the string object and store the corresponding floating-point numbers in separate matrices called `vertexMatrix` and `edgeMatrix`.

2D g2o file:

- In 2D g2o file, each line contains 31 space separated objects. The first object in each line is a string containing either `VERTEX_SE3:QUAT` and `EDGE_SE3:QUAT`, and rest of the 30 objects can be interpreted as floating point numbers (although they might be integers).
- We can read the lines using `textscan` function of MATLAB.
- We classify these lines according to the string object and store the corresponding floating-point numbers in separate matrices called `vertexMatrix` and `edgeMatrix`.

Discussion

Incremental optimization gives a better estimate of the trajectory, as seen in the 2D case. This is because of two reasons. Firstly, in the incremental optimization, we optimize in each iteration giving the solver the chance to correct previous pose estimates using the new data each time. Secondly, we provide the solver a better initial estimate by adding the odometry information to the result from the previous iteration. The ISAM2 solver internally optimizes the incremental optimization process.

The difference between batch and incremental optimization is not prominent in the 3D case because the initial estimate in the 3D case was very good.

README

Instructions to Run my code:

- Download and unzip bnalin_ps4.zip file
- Download the GTSAM toolbox and store it at <folder_path>
- Open MATLAB command window and type:

  ```
  >> addpath(genpath('<folder_path>'));
  ```

- Go to the folder myCode
- Run the scripts hw4_1b, hw4_1c, hw4_2b, hw4_2c
- hw4_1a and hw4_2a are helper functions to read G2O files
- info2cov is a helper function to convert information vector as given in G2o file to corresponding lower cholesky factorization of covariance matrix
- data folder contains the required G2O files.