**2-Dimensional Incremental Trajectory Optimization**

[**Notation**: Variables in the following documentation are in *italics.* Underlined are the functions and conversion/calculations of essential to the isam algorithm.]

The goal is to optimize the trajectory using the batch method. This suggests that we construct the entire factor graph, and optimize it all at once. Thus, all the constraints (links between vertices, landmarks (however for this problem we don't have any landmarks), etc), would be optimized all at once. The main difference is that in this case, we have a dataset for 3-dimensional measurements. Thus, the noise model changes in its matrix length, and instead of .Pose2 function we now use .Pose3(). Most of the other dependencies and functions (like optimization and graph add remain similar as before).

We begin the batch optimization program by reading the sensor data from the “parking-garage.g2o” file using the predefined (or prebuilt) readG2o function from gtsam and loading the same in the variable *graph*. Thus in one line, using the readG2o function, we are able to construct the entire graph (using all the data in the .g2o file) comprising all the poses (vertices) and the links (edges).

The next step is to define the priorModel, comprising the “sigmas” in a diagonal form. Here have a 6-dimensional diagonal array storing the variances (in gtsam.noiseModel.Diagonal form) in the variable called *priorModel*. These values allow the “stochastic” nature of the graph poses. By including a variance (from the defined sigmas), the poses (vertices) are no longer deterministic. Thus we have successfully developed the noise for the prior factor. We then add this noise to the graph. The next step is the optimization of the developed graph. This step is identical to the previous batch problem.

For simplicity, the optimization parameters are chosen to be as of Gauss\_Newton parameters (from the gtsam library), and then set the *optimizer* as the GaussNewtonOptimizer. Then we execute the GaussNewton optimizer, optimizing over the “*graph*” we have made (using all the vertices and edges (constraints) from the .g2o data) and the mentioned parameters (saved as variable *params*)

We then optimize this developed optimizer and store the optimized result in the variable, *result*.

This result is then converted into a NumPy array (after processing through .atPose3() using the generate\_xy\_updated() function) and then stored in the variable *updated\_x\_y*. This allows us to plot the optimized trajectory of the robot. On the same graph, we can also plot the initial\_x\_y trajectory, computed by the sensor reading.

For a detailed understanding of the code and algorithm, please follow the linewise comments in the python code attached herewith.

**Graph:**

Chart

Description automatically generated