**3-Dimensional Batch 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 incremental method. This suggests that we incrementally create vertices (poses) in the graph and optimize them in a sequential (incremental) order.

The first step is to load the poses, edges, and info (information) from the edges\_vertices\_and\_info() function defined in *Problem1a*. This function returns the poses (vertices), edges (constraints), and the information from the input\_INTEL\_g2o.g2o file. This type of data arrangement is going to allow us to pass data (from each time step during the data collection process) in a sequential way (using for and nested for loops).

Once we have the data in this format, we now initialize the gtsam.ISAM2 algorithm to incrementally optimize a nonlinear factor graph. Now, to generate, update and optimize the trajectory sequentially, we start with a for loop, for each *pose*.

In this “for” loop, we first initialize the graph as a *NonLinearFactorGraph*. This initialization will be repeated each time the “for” loop is executed (that is for all *poses*). We then set the *initialEstimates* as gtsam.Values(). This is followed by loading the index (as *idp*) and the x-y-theta (*x,y, theta*) values corresponding to each *pose*.

Now, for the first *pose*, when the index==0, we set the point from the .g2o file with some *priorNoise*. This *priorNoise* is simply a diagonal matrix and easily called gtsam.noiseModel.Diagonal.Sigmas function. Once this point is defined, we insert the *pose* of it (along with its index) to the *initialEstimate*.

This method will be repeated every time the index becomes zero, that is when the robot completes one circle.

For the other indices (*poses*), we write the code in the else condition. The first step is to calculate the *prevPose*, which comes from the *result* variable. This *result* variable corresponds to the calculated Estimate of the graph.  
For the index at hand (from the first “for” loop), we insert this *prevPose* in the *initialEstimate*. Once this is done, we begin another “for” loop, for *edges*. For each *edge*, we first load all its values. The next important step is to generate the information matrix. The information matrix is the 3x3 symmetric matrix. This is stored as variable named *info*.

The next step is to check if the current index (from the first for loop) is equal to the *edge* index we have at hand (from the second “for” loop). If so, this suggests that the robot has been near to the index (vertex) it has already traveled through. This helps us to imply constraints on this *edge*, on which the gtsam optimizer can update and optimize the graph.

We invert the information matrix and develop the covariance matrix Model (stored as *Model*) using this inverse. We add this constraint using the gtsam.BetweenFactorPose2 function between, the first and second index of the *edge* (at hand, from the second “for” loop). We also pass the *pose* received from the *edge* as well as the *Model*. This constraint is added to the *graph*.

In a nutshell, this process is repeated for all *edges*, and then this is repeated for all *poses* (vertices). This generates the graph in sequential order. At the end of each pose, we update the graph, using isam.update (and also pass the *initialEstimate*) as a variable.

In the end, we update the graph using the isam.calculateEstimate and store the result as, *result*.

This result is then converted into a NumPy array (after processing through .atPose2() 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.

[**Note**: For a detailed understanding of the code and algorithm, please follow the line-wise comments in the python code attached herewith.]

**Graph:**

Chart

Description automatically generated