NAVARCH 568: Mobile Robotics
Problem Set 4

Harrison Chen

March 29, 2020

Task 1: 2D Graph SLAM

Part A

The function I wrote to read into the Intel G2O dataset is called `readg2o_2D` and is located at the bottom of the `ROB530_PS4_Task1.m` file.
The function takes in the name of the G2O files as an input, which it opens with `fopen`. Knowing that the G2O file start with vertices and then ends with edges, it takes the file in line by line as a character array. It then checks the first element to see if it's a "V" for VERTEX or an "E" for EDGE. If it sees a "V", it increments a counter called `vertex_end` and proceeds to the next line. When an "E" is detected, it breaks out of the while loop.
Once we have `vertex_end`, which represents the number of vertices in the file, we separate the `fscanf` operations for vertices and edges. This is necessary because the `formatSpec` for the two different types are different: a vertex line has 1 string, 1 integer, and 3 doubles, while an edge line has 1 string, 2 integers, and 9 doubles. Ignoring the strings, we set the dimension of the vertex `fscanf` operation at $4 \times$ `vertex_end` and the edge `fscanf` operation at $11 \times Inf$. We end up with the outputs `vertices` and `edges`, which have columns that correspond to each vertex/edge.

Part B

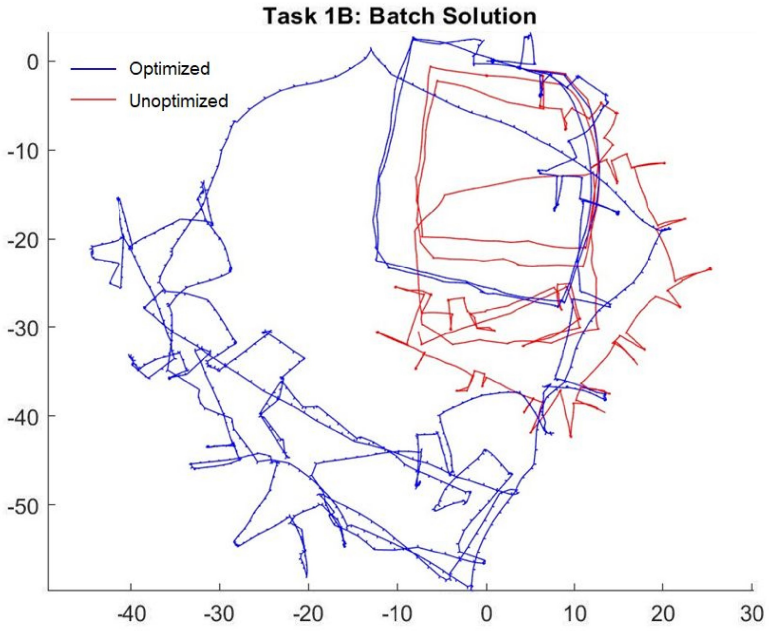


Figure 1: Comparison between optimized trajectory using a Gauss-Newton batch solver and unoptimized trajectory of the Intel dataset

$$priorMean = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, priorNoise = \begin{bmatrix} 0.3^2 & 0 & 0 \\ 0 & 0.3^2 & 0 \\ 0 & 0 & 0.1^2 \end{bmatrix} = \begin{bmatrix} 0.09 & 0 & 0 \\ 0 & 0.09 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}$$

The prior mean that we initialize and use for the Gauss-Newton batch solver is just the (x, y, θ) from the first vertex in the G2O file. The prior variance is referenced from the example code `Pose2SLAMExample.m`. The prior mean and noise are used to build a prior factor, which we add to the NonlinearFactorGraph variable `fg_batch`.

Next, we loop through the columns of the `edges` matrix and add all of them to `fg_batch` as well. We feed in the keys of the corresponding vertices, the (x, y, θ) transformation between them, and the noise model, which is calculated by building out the information matrix using the given upper-triangular terms, then inverting it to get the covariance matrix.

We then initialize the Values variable `initial_batch` to store all the vertices, and loop through columns of the `vertices` matrix to retrieve the key and pose. Finally, we feed the `fg_batch` and `initial_batch` to the `GaussNewtonOptimizer`, run `optimizeSafely`, then plot the results against `initial_batch`.

We observe that without some noise implemented with the given prior, the results of the Gauss-Newton solver are actually less accurate than the unoptimized vertices in the dataset, which is unexpected.

Part C

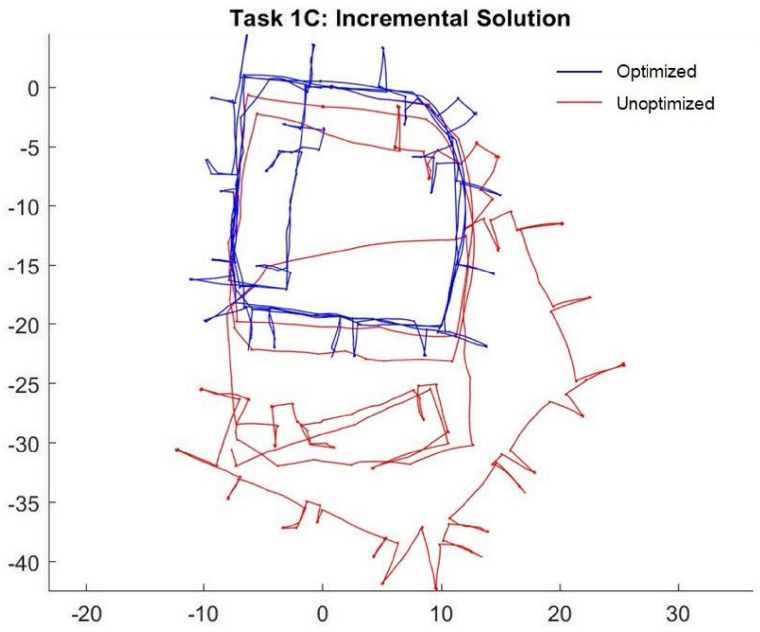


Figure 2: Comparison between optimized trajectory using a incremental smoothing and mapping (iSAM) and unoptimized trajectory of the Intel dataset

$$priorMean = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, priorNoise = \begin{bmatrix} 0.3^2 & 0 & 0 \\ 0 & 0.3^2 & 0 \\ 0 & 0 & 0.1^2 \end{bmatrix} = \begin{bmatrix} 0.09 & 0 & 0 \\ 0 & 0.09 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}$$

For the incremental solution, we use the same prior factor as the batch solution. After creating the ISAM2 object `isam`, we add the prior factor to the NonlinearFactorGraph variable `fg_inc` and the first vertex pose to a Values variable `initial_inc`. `isam` get updated with both, we perform `calculateEstimate`, then store the result in `result_inc`. We then reset the `fg_inc` and `initial_inc`.

Next, we loop through all columns of `vertices`, and within that loop we loop through the columns of `edges`, check if they connect a previous vertex with the current one, and if they do, we add it to `fg_inc`. If this edge also happens to connect the current vertex to its adjacent one (i.e. odometry), we store its pose information to use later.

Once out of the `edges` for loop, we pull the updated version of previous node from `results_inc` and use the odometry pose information stored earlier to find our updated current vertex. It is added to `initial_inc`, then `isam` is updated with `fg_inc` and `initial_inc` again, `results_inc` is updated from the new vertex and edges, and `fg_inc` and `initial_inc` are reset, ready to be used for the next iteration of the `vertices` loop.

Once all the vertices are looped through, we plot the iSAM results against the initial vertices, which are still stored in `initial_batch`.

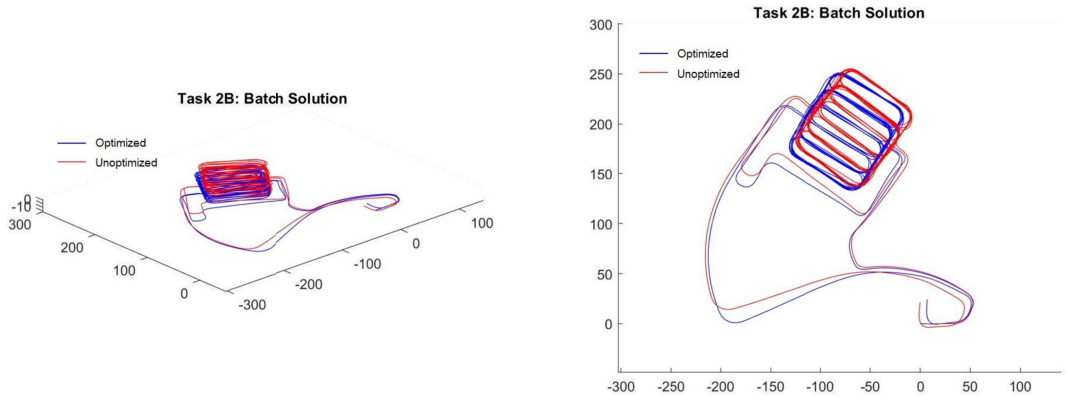
The results of the iSAM map look much more accurate to what we expect from the Intel dataset, with a clear path being traced through a space that resembles what we plotted in Problem Set 3. Comparing it to the batch solution, the incremental one is far superior.

Task 2: 3D Graph SLAM

Part A

The function I wrote to read into the Intel data G2O files is called `readg2o_3D` and is located at the bottom of the `ROB530_PS4_Task2.m` file.
It functions very similarly to `readg2o_2D` from Task 1A, but it uses different `formatSpecs`, because in 3 dimensions, vertices and edges require more values to be properly characterized. Vertices now have 7 doubles, 3 for the location in 3D Cartesian space and 4 for the quaternion that describes the orientation at that vertex. Edges now have 28 doubles, 7 for the pose as described above, and 21 for the upper-triangular values of the 6×6 information matrix.
As a result, the vertex `fscanf` operation has dimensions set at $8 \times$ `vertex_end`, and the edge `fscanf` is done with dimensions $30 \times Inf$.

Part B



(a) Isometric view

(b) Top-down view

Figure 3: Comparison between optimized trajectory using a Gauss-Newton batch solver and unoptimized trajectory of the parking structure dataset

$$priorMean: t = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$priorNoise = \begin{bmatrix} 0.3^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.3^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1^2 \end{bmatrix} = \begin{bmatrix} 0.09 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.09 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.09 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.01 \end{bmatrix}$$

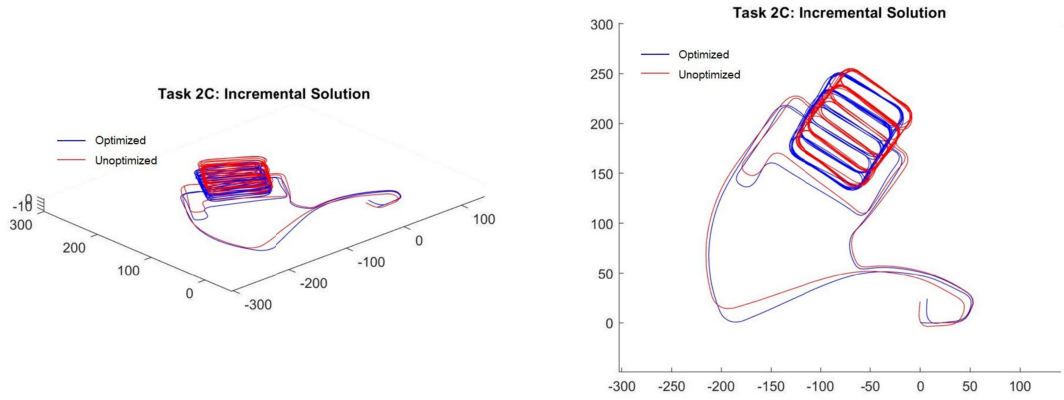
The prior mean that we initialize and use for the Gauss-Newton batch solver is just the pose from the first vertex in the G2O file. The (x, y, z) values are given in the G2O, and the rotation matrix is derived from the quaternion using the following equation:

$$R = \begin{bmatrix} 1 - 2(qy^2 + qz^2) & 2(qx * qy - qz * qw) & 2(qx * qz + qy * qw) \\ 2(qx * qy + qz * qw) & 1 - 2(qx^2 + qz^2) & 2(qy * qz - qx * qw) \\ 2(qx * qz - qy * qw) & 2(qy * qz + qx * qw) & 1 - 2(qx^2 + qy^2) \end{bmatrix}$$

The prior noise is referenced from the example code `Pose2SLAMExample.m` and roughly translated to three dimensions. The prior mean and noise are used to build a prior factor, which we

add to the NonlinearFactorGraph variable `fg_batch`.
The steps mirror the batch solution for two dimensions: add all the edges to `fg_batch`, all the vertices to `initial_batch`, feed them to the `GaussNewtonOptimizer`, `optimizeSafely`, then plot against `initial_batch`. The main difference is when we are finding the pose for a vertex or an edge, we need to find t and R, instead of just plugging in (x, y, θ).
Looking at the plots, we see that the optimized and unoptimized trajectory differ slightly due to accumulated error in the odometry, but that the unoptimized trajectory still shows a shape that resembles the optimized trajectory. This is in contrast to the Intel dataset, which we see from Figure 2 is inaccurate by a rather wide margin.

Part C



(a) Isometric view

(b) Top-down view

Figure 4: Comparison between optimized trajectory using a incremental smoothing and mapping (iSAM) and unoptimized trajectory of the parking structure dataset

$$priorMean : t = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$priorNoise = \begin{bmatrix} 0.3^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.3^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1^2 \end{bmatrix} = \begin{bmatrix} 0.09 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.09 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.09 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.01 \end{bmatrix}$$

For the incremental solution, we use the same prior factor as the 3D batch solution. From there, The steps mirror the incremental solution for two dimensions: perform the initial update and `calculateEstimate` with the prior factor and first vertex, loop through the vertices, adding edges that correspond to the current vertex to `fg_inc` and adding the current vertex using the corresponding odometry transformation to `initial_inc`, then updating the ISAM2 object and `results_inc`. Similar to the batch solution, this differs from its two-dimensional counterpart because we need to find the t and R that correspond to every vertex and edge.
Comparing the results between the batch and incremental solutions for the parking structure dataset, we see that they are indistinguishable. This differs from what was observed with the Intel dataset, where the batch solution yielded worse results than the unoptimized trajectory while the incremental solution performed much better than it.