HW1 Report

Jinxi Xiao
2021533005
ShanghaiTech University

Abstract

This report contains my solutions to the given problems as well as some of my own understandings. In addition, the sections listed in this report does not one-to-one correspond to the problems, but they do overall cover the whole problem set. [1]

1. Plot the Trajectory

First of all, I have installed *evo* library. And I have searched for the meaning of **APE** and **RPE**. Then I use this library to plot the trajectory of the motion of the agent.

1.1. APE

It stands for the absolute pose error, which is a metric for investigating the global consistency of a SLAM trajectory. Given two poses $P_{ref,i}, P_{est,i} \in SE(3)$ at timestamp i, we compute the relative pose between them:

$$E_i = P_{est,i}^{-1} P_{ref,i} \in SE(3)$$

and APE is defined as $||E_i - I||_F$

Notice that for the whole trajectory, we can define **RMSE** to be the "average" of all APEs at every timestamp, given by

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} APE_i^2}$$

1.2. RPE

It stands for the relative pose error, which is a metric for investigating the local consistency of a SLAM trajectory. Given four absolute poses $P_{ref,i}, P_{est,i}, P_{ref,j}, P_{est,j}$, we compute the relative poses based on delta pose difference:

$$E_{i,j} = (P_{ref,i}^{-1} P_{ref,j})^{-1} (P_{est,i}^{-1} P_{est,j})$$

and RPE is defined as $||E_{i,j} - I||_F$, where RMSE is similarly defined as

$$RMSE = \sqrt{\frac{1}{N}\sum_{\forall i,j} RPE_{i,j}^2}$$

1.3. Use *evo* to find the trajectory

Since only relative poses are given, define $\bar{R}, \bar{t}$ as the relative rotation and translation between state i and j. Given the pose of state i as R, t, we can compute the absolute pose for state j as

$$R' = R\bar{R}, \quad t' = R\bar{t} + t$$

Then I convert data into a .tum file where each line contains only

$$time, t_x, t_y, t_z, q_x, q_y, q_z, q_w$$

The trajectory is shown in Figure 1. And ideally the last state (13th state) should be the same as the first state, however, there is a accumulated drift that affects the final result. This is the very error that we want to minimize by using pose-graph optimization with *loop closure factor*.

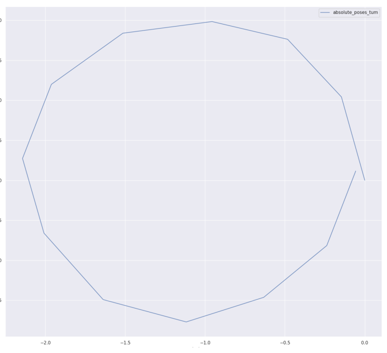


Figure 1. Trajectory with 13 states (we choose the first state's body frame as the world frame)

[1] Codes are stored at my github repo

2. Optimize Pose-Graph From Scratch

2.1. Notations

- *t2v()* is a function maps a transformation matrix in $\mathbb{R}^{3\times 3}$ to the corresponding lie-algebra in $\mathbb{R}^3$. Conversely, *v2t()* is the inverse function.

- $x_i = [t_i, \theta_i]^T \in \mathbb{R}^3, (i = 1, ..., 12)$ to be a state (node in the graph). $T_i \in \mathbb{R}^{3\times 3}, (i = 1, 2, ..., 12)$ to be the transformation matrix of a state. In addition,

$$T_i = \begin{bmatrix} R_i & t_i \\ 0 & 1 \end{bmatrix}$$

- $x^T = [x_1^T, ..., x_{12}^T] \in \mathbb{R}^{1\times 36}$ to be the state vector

- $z_{ij} = [t_{ij}, \theta_{ij}]^T \in \mathbb{R}^3$ to be the raw measurements (i.e. edges in the graph) of relative poses in lie-algebra form. $Z_{ij} \in \mathbb{R}^{3\times 3}$ to be the corresponding transformation matrices. In addition,

$$Z_{ij} = \begin{bmatrix} R_{ij} & t_{ij} \\ 0 & 1 \end{bmatrix}$$

- $\tilde{Z}_{ij}$ to be the calculated relative poses between each two states, where $\tilde{Z}_{ij} = T_i^{-1} T_j$

- $e_{ij}(x) \in \mathbb{R}^3$ to be the error vector.

2.2. Define the loss function

Generally, the loss function widely used is the least square of the prior Gaussian distribution:

$$F_{ij}(x) = e_{ij}(x)^T \Omega_{ij} e_{ij}(x)$$

However, only relative poses are given, we could not find the Ω_{ij} from raw measurements. Similar as [1], we rather force $\Omega_{ij} = I_3$

Also, we have two ways to define $e_{ij}(x)$, it would be

$$e_{ij}(x) = \tilde{Z}_{ij} - Z_{ij} \tag{1}$$

or

$$e_{ij}(x) = t2v(Z_{ij}^{-1} \tilde{Z}_{ij}) \tag{2}$$

We will show that these two definitions are the same in the context of computing gradients.

2.3. Compute Jacobians

We can find the relative transforamtion between each two states as

$$\tilde{Z}_{ij} = T_i^{-1} T_j = \begin{bmatrix} R_i^T R_j & R_i^T (t_j - t_i) \\ 0 & 1 \end{bmatrix}$$

It would be easy to find out

$$\tilde{z}_{ij} = t2v\left(\tilde{Z}_{ij}\right) = \begin{bmatrix} R_i^T (t_j - t_i) \\ \theta_j - \theta_i \end{bmatrix}$$

If we follow the definition in equation 1, we can conclude that

$$A_{ij} = \frac{\partial e_{ij}(x)}{\partial x_i^T} = \begin{bmatrix} -R_i^T & \frac{\partial R_i^T}{\partial \theta_i}(t_j - t_i) \\ 0 & -1 \end{bmatrix}$$
$$B_{ij} = \frac{\partial e_{ij}(x)}{\partial x_j^T} = \begin{bmatrix} R_i^T & 0 \\ 0 & 1 \end{bmatrix}$$

where

$$\frac{\partial R_i^T}{\partial \theta_i} = \begin{bmatrix} -\sin(\theta_i) & \cos(\theta_i) \\ -\cos(\theta_i) & -\sin(\theta_i) \end{bmatrix}$$

Otherwise, if definition 2 is used, we can find error vector as

$$\begin{aligned} e_{ij}(x) = & \begin{bmatrix} R_{ij}^T (R_i^T (t_j - t_i) - t_{ij}) \\ \theta_j - \theta_i - \theta_{ij} \end{bmatrix} \\ = & \begin{bmatrix} R_{ij}^T & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_i^T (t_j - t_i) \\ \theta_j - \theta_i \end{bmatrix} - \begin{bmatrix} R_{ij}^T t_{ij} \\ \theta_{ij} \end{bmatrix} \end{aligned}$$

And notice that R_{ij}, t_{ij} are measurements, which means that they can be regarded as constants, thus to some extend can be ignored. Then we can define the jacobian as

$$J_{ij} = \begin{pmatrix} 0 & \dots & A_{ij} & 0 & \cdots & 0 & B_{ij} & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{3\times 36}$$

the overall jacobian matrix would be $J = \sum_{i,j} J_{ij}$.

For the following part, I will use definition 1 to define error vector $e_{ij}(x)$.

2.4. Hessian-Information matrix

In my point of view, we use $J^T J$ to estimate the hessian matrix H. It would be easy to infer that

$$H = \begin{pmatrix} \ddots & & & & \\ & A_{ij}^T A_{ij} & \cdots & A_{ij}^T B_{ij} & \\ & \vdots & \ddots & \vdots & \\ & B_{ij}^T A_{ij} & \cdots & B_{ij}^T B_{ij} & \\ & & & & \ddots \end{pmatrix} \in \mathbb{R}^{36\times 36}$$

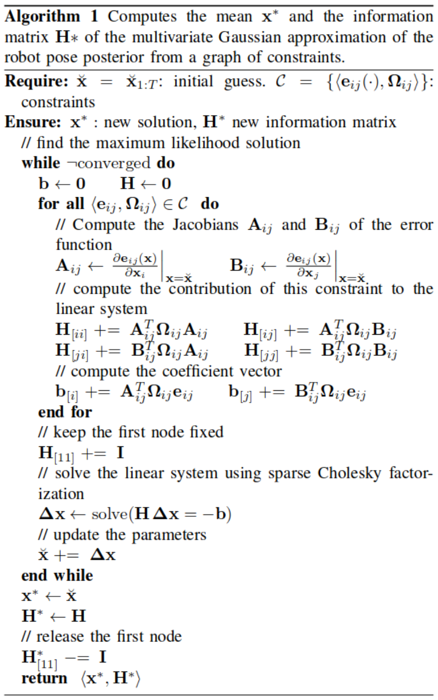
where the coefficient vector is defined as

$$b = \begin{pmatrix} \vdots \\ A_{ij}^T e_{ij}(x) \\ \vdots \\ B_{ij}^T e_{ij}(x) \\ \vdots \end{pmatrix} \in \mathbb{R}^{36\times 1}$$

Then we can iteratively find $\Delta x \in \mathbb{R}^{36}$ where $H\Delta x = -b$, and the process stops when the L2-norm of Δx is less than a threshold, which I set to it be 10^{-9} in practice.

2.5. Experiments and Results

The general pipeline of the optimization process is similar to the one shown in [2] (shown in Figure 2). I have run the program for 10 times with an average time of 0.018610 seconds, and the it takes 7 iterations to converge. Notice that the time I record is the running time of the *while-loop*, i.e. the main optimization process, which means that I do not consider all the time consumption of intialization-process as well as output-process. Trajectories are shown in Figure 3. In addition, the RMSE(full transformation) of APE of original slam poses is 0.166972 while for optimized poses is 0.083524. Notice that when computing APE, we drop the last edge of the origin slam pose $T_{12,1}$. So only 11 edges but 12 states are considered.



Algorithm 1 Computes the mean $\mathbf{x}^*$ and the information matrix $\mathbf{H}*$ of the multivariate Gaussian approximation of the robot pose posterior from a graph of constraints.

```
Require: $\breve{\mathbf{x}} = \breve{\mathbf{x}}_{1:T}$: initial guess. $\mathcal{C} = \{\langle \mathbf{e}_{ij}(\cdot), \mathbf{\Omega}_{ij} \rangle\}$: constraints
Ensure: $\mathbf{x}^*$ : new solution, $\mathbf{H}^*$ new information matrix
  // find the maximum likelihood solution
  while ¬converged do
    $\mathbf{b} \leftarrow \mathbf{0}$    $\mathbf{H} \leftarrow \mathbf{0}$
    for all $\langle \mathbf{e}_{ij}, \mathbf{\Omega}_{ij} \rangle \in \mathcal{C}$ do
      // Compute the Jacobians $\mathbf{A}_{ij}$ and $\mathbf{B}_{ij}$ of the error function
      $\mathbf{A}_{ij} \leftarrow \left.\frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$    $\mathbf{B}_{ij} \leftarrow \left.\frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$
      // compute the contribution of this constraint to the linear system
      $\mathbf{H}_{[ii]} \mathrel{+}= \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{A}_{ij}$    $\mathbf{H}_{[ij]} \mathrel{+}= \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{B}_{ij}$
      $\mathbf{H}_{[ji]} \mathrel{+}= \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{A}_{ij}$    $\mathbf{H}_{[jj]} \mathrel{+}= \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{B}_{ij}$
      // compute the coefficient vector
      $\mathbf{b}_{[i]} \mathrel{+}= \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}$    $\mathbf{b}_{[j]} \mathrel{+}= \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}$
    end for
    // keep the first node fixed
    $\mathbf{H}_{[11]} \mathrel{+}= \mathbf{I}$
    // solve the linear system using sparse Cholesky factorization
    $\mathbf{\Delta x} \leftarrow \text{solve}(\mathbf{H}\,\mathbf{\Delta x} = -\mathbf{b})$
    // update the parameters
    $\breve{\mathbf{x}} \mathrel{+}= \mathbf{\Delta x}$
  end while
  $\mathbf{x}^* \leftarrow \breve{\mathbf{x}}$
  $\mathbf{H}^* \leftarrow \mathbf{H}$
  // release the first node
  $\mathbf{H}^*_{[11]} \mathrel{-}= \mathbf{I}$
  return $\langle \mathbf{x}^*, \mathbf{H}^* \rangle$
```

Figure 2. General Algorithm

3. Optimize Via G2O

In this section, we will make use of the G2O [3] C++ library. Similar to Section 2, we define verteces as well as edges. And in order to be fair, I set the number of iterations to be the same(which is 7). In addition, I will only record *optimization* time, i.e. neither of initialization process nor output process will be included.

I first try Gauss-Newton method to optimize, and it turns out that the result of g2o is the **same** as my result shown in Section 2.5! Then I switch to Levenberg-Marquardt method for optimization. Surprisingly, when we just consider 7 iterations, the g2o-system will output the same result, which implies that the optimized odometries are very close to the **optimal** ones. However, by using the implemented Levenberg-Marquardt method of g2o, we will see a much quicker run time, which is 0.008550, the average running time of 10 times.

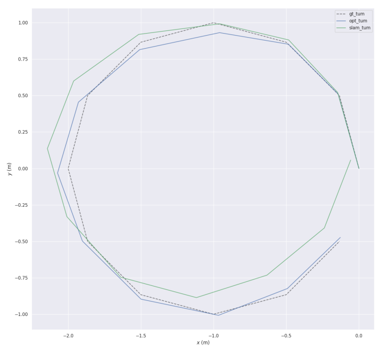


Figure 3. The dotted line is the generated ground truth; blue line is my optimized results; green line is the raw slam results

The general numerical result of different methods are shown in Table 1. And since trajectories of optimized results are the same, please refer to Figure 3 as the trajectory result.

	Raw SLAM	Scratch	G2O
Time(s)	-	0.018610	0.008550
RMSE	0.166972	0.083524	0.083524

Table 1. The time and accuracy of raw data, hand-written implementation(scratch) and g2o implementation.

4. Revisit Information Matrix

4.1. General View

A general view of the information is shown in Figure 4. I set all entries with value greater than 0 to be red, and entries with value value less than 0 to be blue. For the rest of the part, it should be grey. From the figure we can clearly see that it is sparse and symmetric, and most non-zero entries are located around the main-diagonal, which makes it similar to a band matrix. The left-down and right-up corners are not zero because there's a loop closure factor.

4.2. A more local view

Notice that

$$A_{ij}^T = \begin{bmatrix} -R_i & 0 \\ (t_j - t_i)^T \frac{\partial R_i}{\partial \theta_i} & -1 \end{bmatrix}$$
$$B_{ij}^T = \begin{bmatrix} R_i & 0 \\ 0 & 1 \end{bmatrix}$$

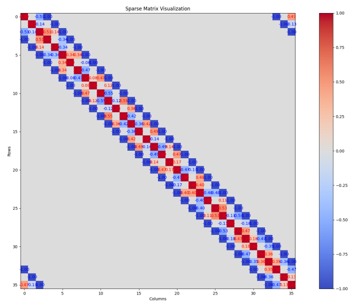


Figure 4. The Information Matrix After Convergence

Thus we can conclude that

$$A_{ij}^T A_{ij} = \begin{bmatrix} I & * \\ * & (t_j - t_i)^T (t_j - t_i) \end{bmatrix}, B_{ij}^T B_{ij} = I_3$$
$$A_{ij}^T B_{ij} = \left(B_{ij}^T A_{ij}\right)^T = \begin{bmatrix} -1 & 0 \\ * & -1 \end{bmatrix}$$

Then there are some more properties (besides that it is sparse and symmetric) for this information matrix:

$$H_{ij} = \begin{cases} 2, & i = j \neq 3k \\ -1, & j - k = 3 \text{ or } i - k = 3 \\ *, & otherwise \end{cases}$$

where $i, j = 1, ..., 36$ and $k = 1, ..., 12$.

References

[1] "Calculate information matrix for graph slam." `https://robotics.stackexchange.com/questions/22451/calculate-information-matrix-for-graph-slam`.

[2] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[3] "g2o library." `https://github.com/RainerKuemmerle/g2o/tree/master`.