MVA: NPM3D (2020/2021) **Final Project**

Generalized-ICP

Work by *Remi CARNEC*

Contents

1 Abstract

2 Introduction

Generalized Iterative Closest Point (Generalized-ICP) was introduced by Segal et. al. [3]. This algorithm can for instance be used for Scanmatching - to align two point clouds. It lies at the crossing of the two following methods, and combines them into a probabilistic framework:

- The standard ICP, introduced in [1].
- The Point-to-plane method, introduced in [2].

It is argued in [3] that Generalized-ICP outperforms these two methods in numerous datasets, in addition to giving a flexible probabilistic interpretation. In this report, we first briefly recall the basic principles of standard ICP and point-to-plane. We then start giving more details about the Generalized-ICP approach, its probabilistic interpretation, the algorithm as well as our implementation. Eventually, we compare the performances of all three algorithms in terms of convergence and computational time.

3 The algorithm

3.1 Standard ICP

3.2 Point-to-plane

3.3 Generalized-ICP

4 Implementation

In order to compare the different methods, we need to implement each optimization problem. This section aims at giving some insight as to how the optimization is done for each method.

- We have seen in class that we can write a close form solution for the standard ICP method. Again, the standard ICP loss writes:

$$l(R,t) = \sum_i (b_i - Ra_i - t)^T (b_i - Ra_i - t)$$

  Computing the centered clouds $A' = A - \hat{a}$ and $B' = B - \hat{b}$, as well as the covariance matrix $H = A'B'^T$, we find:

$$R = VU^T \text{ and } T = \hat{b} - R\hat{a} \tag{1}$$

  Where USV^T is the singular value decomposition of H. Although [3] suggests to use the Conjugate Gradient method to find the solution, this is faster to use this expression.

- When it comes to the *point-to-plane* method, we need to minimize a more complex function. As suggested in [3], we do so by using the *Conjugate Gradient* algorithm. Python functions such as `scipy.optimize.minimize` allow to use this method without providing the gradient using an approximation. However, this is far more expensive than having an exact formula for the gradient. Instead, we try to compute the gradient of the loss $l(R,t)$ to accelerate the optimization. Recall that:

$$l(R,t) = \sum_i (b_i - Ra_i - t)^T P_i (b_i - Ra_i - t)$$

  On the one hand, we have:

$$\nabla_t l(R,t) = -2 \sum_i P_i (b_i - Ra_i - t) \tag{2}$$

On the other hand, we can write and develop the loss function as:

$$\begin{aligned} l(R,t) &= \sum_i \mathrm{Tr}(P_i(b_i - Ra_i - t)(b_i - Ra_i - t)^T) \\ &= \sum_i \mathrm{Tr}\left(P_i\left((b_i - t)(b_i - t)^T + Ra_i a_i^T R^T - (b_i - t)a_i^T R^T - Ra_i(b_i - t)^T\right)\right) \\ &= \sum_i \mathrm{Tr}\left(P_i(b_i - t)(b_i - t)^T\right) + \sum_i \mathrm{Tr}\left(Ra_i a_i^T R^T P_i\right) - 2\sum_i \mathrm{Tr}\left(Ra_i(b_i - t)^T P_i\right) \end{aligned}$$

Now, using the facts that $\frac{\partial}{\partial X}\mathrm{Tr}(XB) = B^T$ and $\frac{\partial}{\partial X}\mathrm{Tr}(X^T BXC) = BXC + B^T XC^T$ (cf [4]), we obtain:

$$\nabla_R l(R,t) = -2\sum_i P_i(b_i - Ra_i - t)a_i^T \tag{3}$$

We can now directly provide the gradient as a callable function to the optimizer.

- The *Plane-to-plane* (Generalized-ICP) method is even more complex. The loss function is:

$$l(R,t) = \sum_i (b_i - Ra_i - t)^T \left(C_i^B + RC_i^A R^T\right)^{-1} (b_i - Ra_i - t)$$

We first notice that the central term $\left(C_i^B + RC_i^A R^T\right)^{-1}$ requires inverting n matrices. Moreover, while P_i did not depend on the transformation (R,t), this this term does. This means that everytime we compute the loss function or the gradient, we need to invert as many 3×3 matrices as there are points. This shows how computationally expensive the Conjugate Gradient can be, especially if we approximate the gradient using several values of loss (numerical approximations), and highlights the necessity to find an exact formula for the gradient. Since the central term is independent of t, we first have:

$$\nabla_t l(R,t) = -2\sum_i \left(C_i^B + RC_i^A R^T\right)^{-1} (b_i - Ra_i - t) \tag{4}$$

Now, we wish to compute $\nabla_R l(R,t)$. As explained before, the difference with *Point-to-plane* is that the central term depends on R. We can still use the same reasoning as before, but we need to add another term due to this dependence. First, we write:

$$l(R,t) = \sum_i \mathrm{Tr}(\left(C_i^B + RC_i^A R^T\right)^{-1} (b_i - Ra_i - t)(b_i - Ra_i - t)^T)$$

[4] states that for symmetric matrices B and C, we have:

$$\frac{\partial}{\partial X}\mathrm{Tr}\left[(X^T CX)^{-1} A\right] = -(CX(X^T CX)^{-1})(A + A^T)(C^T CX)^{-1}$$

Along with a few well known properties of the trace, this is enough to compute the gradient. We finally obtain:

$$\begin{aligned} \nabla_R l(R,t) = -2\sum_i \Big[&\left(C_i^B + RC_i^A R^T\right)^{-1} d_i a_i^T \\ &+ \left(C_i^B + RC_i^A R^T\right)^{-1} d_i d_i^T \left(C_i^B + RC_i^A R^T\right)^{-1} RC_i^A \Big] \end{aligned} \tag{5}$$

Where we denoted $d_i = b_i - Ra_i - t$.

Note that for *Point-to-plane* and *Plane-to-plane*, our work does not stop here: R has to be a rotation matrix. To satisfy this constraint, we use Euler's formulation to express R as a function of three angles: θ_x, θ_y, θ_z.

5 Experiments

6 Conclusion

References

[1] Besl P. and McKay H. *A method for registration of 3-D shapes.* IEEE Trans. Pattern Anal. Mach. Intell., 1992

[2] Kok-Lim Low. *Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration.* University of North Carolina, 2004

[3] Aleksandr V. Segal, Dirk Haehnel and Sebastian Thrun. *Generalized-ICP.* Robotics: Science and Systems, 2009

[4] Kaare Brandt Petersen, Michael Syskind Pedersen *The Matrix Cookbook.* 2005