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

Generalized-ICP

Work by *Remi CARNEC*

Contents

1 Abstract

The following report aims at summarizing my work done for the final project of the MVA course NPM3D (2020-2021). In particular, the topic of my work was the Generalized-ICP method, introduced in [3]. You will find in this report some details about my understanding of this method, the mathematical foundation necessary to implement the algorithm, as well as my main takeaways from some experiments that I ran.

2 Introduction

Generalized Iterative Closest Point (Generalized-ICP) was introduced by Segal et. al. [3]. This algorithm can for instance be used for Scanmatching - i.e. 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, in addition to giving a flexible probabilistic interpretation, Generalized-ICP outperforms these two methods for numerous datasets. In this report, we first briefly recall the basic principles of standard ICP and *Point-to-plane* procedures. We then start giving more details about the Generalized-ICP approach, its probabilistic interpretation, the different steps of 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

All three algorithms have the same global structure, which consists in iteratively repeating:

- Computing the correspondance $(a_{k_i})_i = (m_i)_i \leftrightarrow (b_i)_i$ between the two scans.
- Computing a transformation $T = (R, t)$ that minimizes a certain loss $l(R, t, (m_i)_i, (b_i)_i)$.

We explicit the procedure below. Note that the threshold $d_{\max}$ is used to deal with the violation of the assumption of full overlap. The loss l is different from one method to another as it contains the intuitive sense of what we want to minimize. More details are given about $l(R, t)$ in the following subsections.

Algorithm 1: Scanmathing algorithm structure

Input: Two pointclouds: $A = \{a_i\}$, $B = \{b_i\}$
Result: The correct transformation R, t that aligns A and B
Initialize R, t;
while *not converged* **do**
  for $i \leftarrow 1$ *to* N **do**
    $m_i \leftarrow$ `FindClosestPointInA`$(T \cdot b_i)$;
    if $\|m_i - T \cdot b_i\| \leq d_{\max}$ **then**
      $w_i \leftarrow 1$;
    else
      $w_i \leftarrow 0$;
    end
  end
  $T \leftarrow \text{argmin}_{R,t} l(R, t, (m_i)_i, (b_i)_i)$
end

3.1 Standard ICP

In the case of Standard ICP, minimizing the loss is a least square approach: it consists in minimizing the total squared distance between corresponding points. Formally, it writes:

$$l(R, t, (m_i)_i, (b_i)_i) = \sum_{i=1}^{N} w_i \|m_i - T \cdot b_i\|_2^2 = \sum_{i=1}^{N} w_i \|m_i - t - Rb_i\|_2^2$$

3.2 Point-to-plane

The *Point-to-plane* approach differs with the Standard ICP in that it takes advantage of surface normal information. At each iteration, it aims at minimizing the error along the surface normal. Formally, the loss function becomes:

$$l(R,t,(m_i)_i,(b_i)_i) = \sum_{i=1}^{N} w_i|\eta_i \cdot (m_i - T \cdot b_i)|_2^2 = \sum_{i=1}^{N} w_i|\eta_i \cdot (m_i - t - Rb_i)|_2^2$$

Where η_i is the normal of the surface at a_i. The loss function is illustrated in Figure 1. It is known to be more robust than standard ICP.

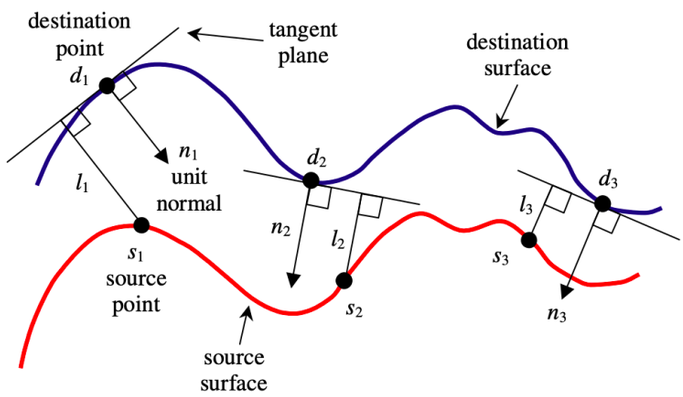


Figure 1: Point-to-plane error between two surfaces (source: [2])

3.3 Generalized-ICP

Generalized-ICP is an extension of the previous methods to a probabilistic framework. It relies on the assumption that the pointclouds have been generated according to a gaussian distribution. More precisely, all points a_i and b_j are assumed to be simulated according to local Gaussian distributions $\mathcal{N}(\hat{a}_i, C_i^A)$ and $\mathcal{N}(\hat{b}_j, C_j^B)$. Using this assumption, we know that with T^* the correct transformation:

$$d_i^{T^*} \sim \mathcal{N}\left(0, C_i^B + T^* C_i^A {T^*}^T\right) \text{ where } d_i^T = b_i - T \cdot a_i$$

Therefore, as described in [3], writing the *Maximum Likelihood Estimation* maximization problem boils down to solving:

$$T = \text{argmin}_T \sum_i {d_i^{(T)}}^T (C_i^B + T C_i^A T^T)^{-1} d_i^{(T)}$$

The key feature of this algorithm is the possibility for the user to set the covariance matrices C_i^A and C_i^B as they see fit. Since we want these matrices to model a surface structure (that is a very low covariance in the surface normal direction), they are typically set to be equivalent to I_ϵ in the vector base $(e_i^{1,C}, e_i^{2,C}, e_i^{3,C})$ ($e_i^{1,C}$ is the normal vector to the surface at the i^{th} point of the cloud C, here A and B), where:

$$I_\epsilon = \begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In other words, if R_i^A and R_i^B are the rotation matrices that transform $(1,0,0)$ into $e_i^{1,A}$ and $e_i^{1,B}$, we have:

$$C_i^A = R_i^A I_\epsilon {R_i^A}^T, \text{ and } C_i^B = R_i^B I_\epsilon {R_i^B}^T$$

However, one can choose to use a difference covariance structure, which explains why it is considered a generalization. For instance, setting $C_i^B = I$ and $C_i^A = 0$ leads to the Standard-ICP problem. On the other hand, $C_i^B = P_i^{-1}$ (more precisely, an invertible approximation of the projection matrix onto $e_i^{1\,B}$) and $C_i^A = 0$ models the *Point-to-plane* problem.

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.

4.1 Mathematical foundation

- We have seen in class that we can write a closed form solution for the standard ICP method, which we recall here. 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:

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

Where P_i is the projection matrix onto the linear span of the surface normal. 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 $\frac{\partial}{\partial X}\mathrm{Tr}(X^T B X C) = BXC + B^T X C^T$ (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:

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

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

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

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 (see [5] for more details) to express R as a function of three angles: θ_x, θ_y, θ_z. The chain rule writes:

$$\frac{\partial}{\partial\theta} l(R(\theta),t) = \sum_{i,j=1}^{3} \frac{\partial l(R,t)}{\partial R_{i,j}} \frac{\partial R_{i,j}}{\partial\theta}$$

Where $\frac{\partial l(R,t)}{\partial R_{i,j}}$ is given by (3) for the *Point-to-plane* method and by (5) for the Generalized-ICP, while $\frac{\partial R_{i,j}}{\partial\theta_x}$ is obtained from Euler's formulation.

4.2 Implementation strategy

We briefly enumerate the key aspects of our implementation:

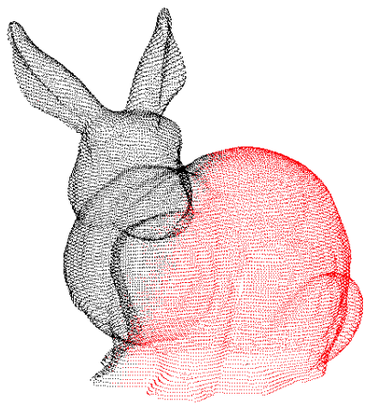
- For the *Point-to-point* method, we used the exact formula (see 1) of the optimization problem, rather than using the Conjugate gradient method.
- As evoked in section 4.1, the gradient of the loss can be numerically approximated by evaluating the loss multiple time using small variations, but the loss can be quite expensive to compute, particularly for the Generalized-ICP framework that requires inverting numerous matrices. We thus implemented the exact calculation of the loss and gradient using a linear solver instead of inverting matrices since most of the times, we actually want to compute $y = A^{-1}x$ rather than A^{-1} itself.
- The value of ϵ in I_ϵ (see section 3.3) is typically set to 10^{-3}.
- The eigenvectors are required for the *Point-to-plane* and *Plane-to-plane* algorithms. We typically use PCA using the $k = 20$ nearest neighbours to estimate them, as suggested in [3]. Moreover, to avoid repetitive numerical calculation, we calculate and store beforehand the eigenvectors, the projection and covariance matrices, not to do it in every passage of the *While* loop in algorithm 1.
- Note that the optimization problem $\min_{R,t} l(R,t)$ is not convex for the Generalized-ICP. Consequently, the optimizer can get stuck in local minimas and can spend a lot of time before reaching the global minima. We realized that setting an upper bound for the number of iteration of the optimizer often led to faster overall convergence, as illustrated in 11.

5 Experiments

5.1 Comparisons

We first compare the performance of the different algorithms on the *Perturbed* and *Original Bunny* dataset that we used for the second practical session. If not precised otherwise, the tolerance was set to 10^{-5} and $d_{\max}$ was set to 0.1. We note that *Point-to-plane* perfoms better than *Point-to-point* in terms of number of iterations, and Generalized-ICP seems to be the best method.

(a) Original clouds



(b) Generalized-ICP final transformation

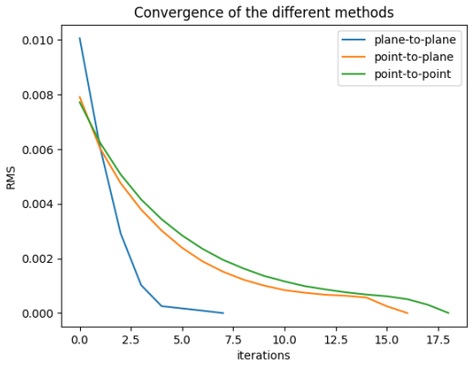
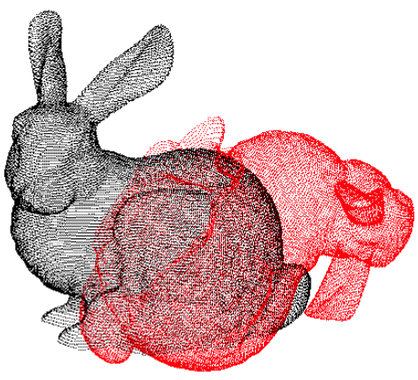


Figure 2: Comparison of convergence between the different methods

Similarily, we try the three methods on more distant datasets by applying a random transformation as seen in Figure 3. Again, Generalized-ICP outperforms the two other algorithms that seem to get stuck in a local minima. On the contrary, although Generalized-ICP reached the maximum number of iterations (40), it seems to converge quite well.



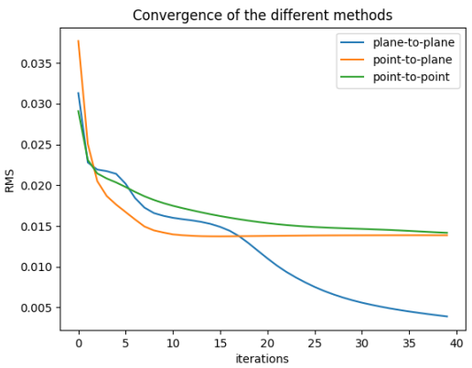


Figure 3: Comparison of convergence between the different methods for the original point clouds (left)

Even for clouds that do not completely coincide, the *Plane-to-plane* approach behaves fairly well. We decided for example to eliminate randomly half of the points for the *Original* and *Perturbed* datasets, and as shown in Figure 6, it still outperforms. Note that the RMS

cannot converge to 0 as the clouds are not identical (we eliminated different points in each cloud).

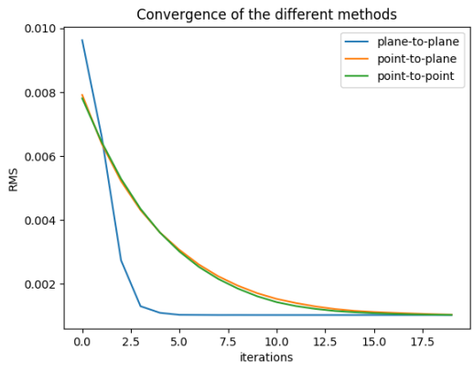


Figure 4: Randomly selecting 1/2 of the points

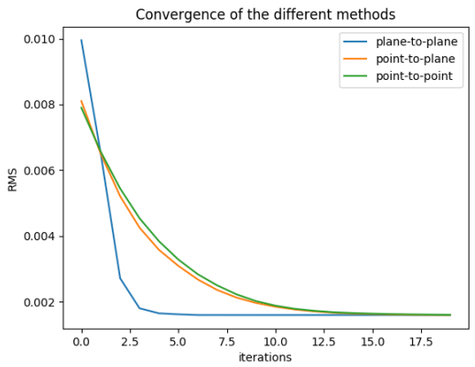
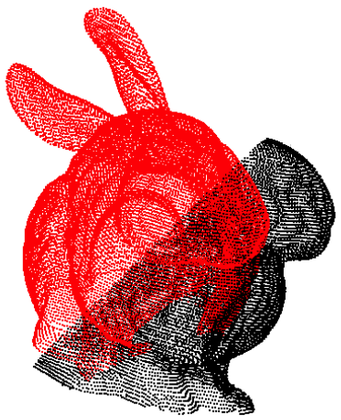


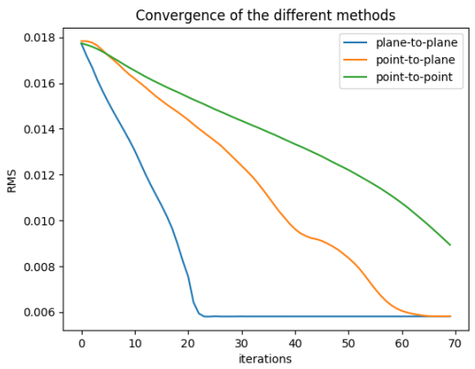
Figure 5: Randomly selecting 1/5 of the points

Figure 6: Convergence of the different methods

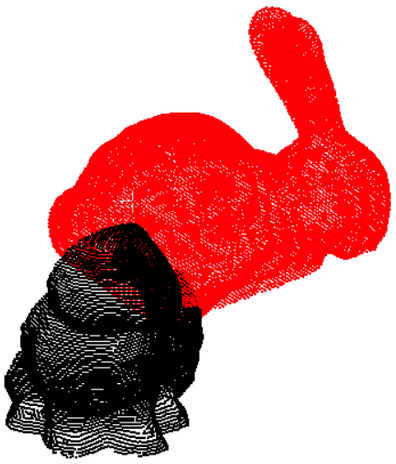
It is also important to see if the algorithm performs well if we violate the full overlapping assumption. We thus create two distinct pointclouds that slightly overlap (by choosing two hyperplanes and selecting the points from one side and another), and we choose $d_{\max} = 0.005$. The results are illustrated in Figure 7. However, Generalized-ICP seems to be quite unstable when dealing with such datasets, and does not necessarily lead to a good solution.



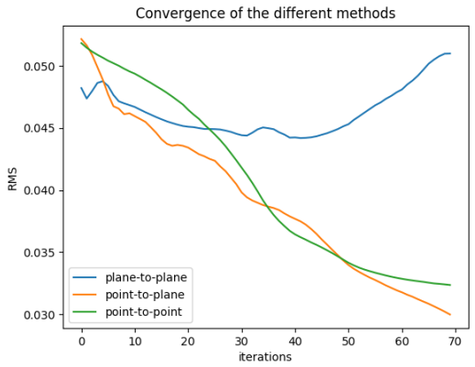
(a) A random transformation



(b) The associated convergence



(c) Another random transformation



(d) The associated convergence

Figure 7: Comparison of convergence between the different methods for partial overlapping

We want to see how *Plane-to-plane* performs on other datasets. We use the *Dragon* dataset from the Stanford 3D Scanning Repository, apply a random transformation and

compare the different methods as shown in Figure 10. Again, Generalized-ICP converges faster in terms of iterations. What is interesting is that *Point-to-plane* does not even seem to converge. This may be due to the fact that the surface of the *Dragon* cloud is more complex and irregular than the *Bunny*.

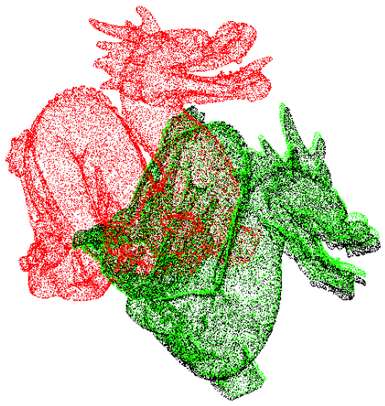


Figure 8: Original (black), perturbed (red) and final (green) pointclouds

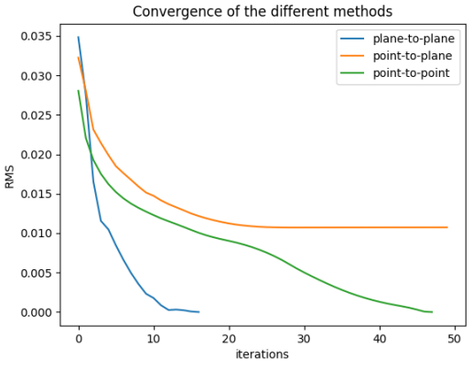


Figure 9: Convergence

Figure 10: Convergence of the different methods on the dragon pointcloud. Generalized-ICP took 106.77s to converge, *Point-to-point* needed 12.7s (but more iterations) and *Point-to-plane* did not converge.

However, Generalized-ICP is much slower than the two other methods. Indeed, the computation of the loss and of the gradient requires inverting numerous matrices of the form $C_i^B + R_i C_i^A R_i^T$. For instance, running the three algorithms on the dataset shown in Figure 2 leads to the following computational times:

- Point-to-point: 2.52s
- Point-to-plane: 11.57s
- Plane-to-plane: 37.46s

5.2 Improving Generalized-ICP?

- As we hinted at before, the optimization problem of Generalized-ICP is not convex. This causes the optimizer (Conjugate gradient) to get stuck in local minimas and spend many iterations (and time) converging to solutions we are not interested in. We can solve this problem by reducing the `max_iter` parameter, as illustrated by Figure 11. Note that there is a trade-off: if this parameter is too low, then the algorithm has trouble converging and the number of overall iterations (in the *While*-loop in algorithm 1) increases again. Setting `max_iter` $= 10$ seems to be a good compromise.

- Since the main computational cost of Generalized-ICP lies in the inversion of the matrices $C_i^B + RC_i^A R^T$, we tried to find another way to converge at each iteration. We thus assumed that the principal variation of $l(R,t)$ with respect to R was due to the quadratic term $d_i d_i^T$, where $d_i = b_i - t - Ra_i$. We proceed as described in algorithm 2.

Algorithm 2: Variant of Generalized-ICP

Input: Two linked pointclouds: $A = \{a_i\}$, $B = \{b_i\}$
Result: A good approximation to the $\text{argmin}_{R,t} l(R,t)$
Initialize $R_0, t_0, (C_i^B + R_0 C_i^A R_0^T)^{-1}$;
Initialize $k \leftarrow 1$;
while *not converged* **do**
  $R_k, t_k \leftarrow \text{argmin}_T \sum_i {d_i^{(T)}}^T (C_i^B + R_{k-1} C_i^A R_{k-1}^T)^{-1} d_i^{(T)}$;
  $k \leftarrow k + 1$;
end

After running a few experiments, our feelings are somewhat mitigated. Although it seems to be much faster than the exact Generalized-ICP, it is also less robust. As

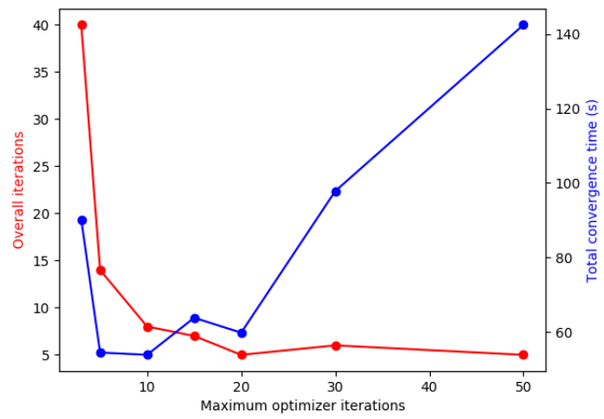


Figure 11: Influence of the maximum number of iterations on the convergence

an example, we show how both methods perform on the original dataset shown in 2, and on the more difficult dataset shown in Figure 7. One can see how expensive the computation of the gradient can get, and a good solution could be to use this relaxed version, but only for datasets which transformation is not too dificult to find. That being said, note that out of the 268.8s taken by the exact Generalized-ICP on the right figure, only 60s are needed for it to converge (around 15 iterations).

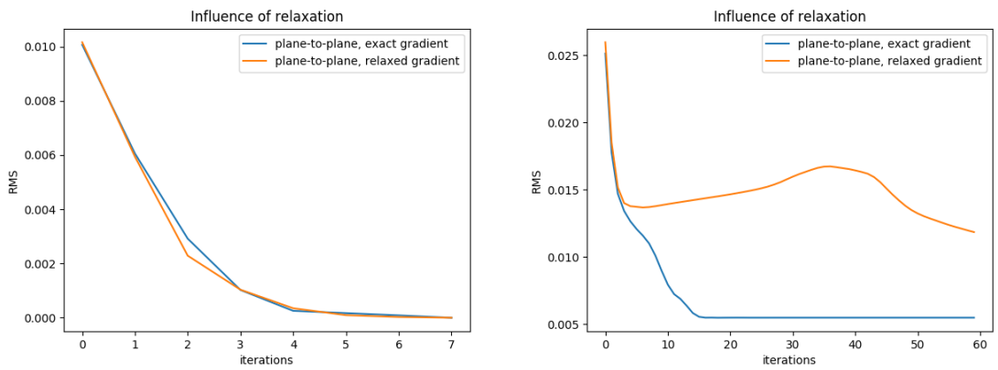


Figure 12: Influence of relaxation, full (left) and partial (right) overlapping. The exact method took respectively 51.22s and 268.8s, and the relaxed one took 8.32s and 16.3s.

6 Conclusion

In this report, we summarized the basic principles of Iterative Closest Point (ICP), *Point-to-plane* and Generalized ICP. We ran the calculation we needed to implement an optimization method for each algorithm, and finally we compared their performance. We showed that Generalized-ICP seems to perform better in terms of total iterations and convergence for many different situations: complete overlap, partial overlap, random transformation. Even if we saw that it could sometimes be unstable, in most cases it outperforms *Point-to-point* and *Point-to-plane*. That being said, we would like to highlight that it is also much more computationally expensive, due to the tedious loss and gradient calculation. The total computation time can be orders of magnitudes higher than the other methods, even for very simple datasets. This was not made very clear in the original paper [3], which in our opinion overlooked this essential issue. We investigated two possibilities to accelerate this procedure: *Early-stopping* and *Gradient-relaxation*. However, the latter can sometimes lack robustness.

A good idea could be to investigate the role of the parameter $d_{\max}$ on the convergence and robustness for the different methods (including the relaxed Generalized-ICP) and how the trade-off between convergence and accuracy can be handled in order to find the right solution.

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

[5] *Matrice de rotation* https://fr.wikipedia.org/wiki/Matrice_de_rotation