Sparse Iterative Closest Point
Written report - Nuage de point et modélisation

Pierre-Alain Langlois
MVA - ENPC
langloispierrealain@gmail.com

Abstract

This document presents an implementation and an analysis of the Sparse Iterative Closest Point as described in [1]. It was realized for the M2 course "Nuage de point et Applications" at Ecole Normale Supérieure de Cachan.

Contents

1 Introduction **2**

2 The algorithm **2**

2.1 Point-to-point algorithm 2

2.2 Point-to-plane algorithm 2

2.3 Robust distance function 2

3 Implementation **3**

3.1 Dependencies 3

3.2 Approximating the normals 3

3.3 The algorithm itself 3

3.3.1 Point-to-point 3

3.3.2 Point-to-plane 4

4 Results **4**

4.1 Setup 4

4.2 Test and Analysis 5

4.2.1 Point-to-point 5

4.2.2 Point-to-plane 5

5 Conclusion **6**

1 Introduction

The iterative closest point algorithm tries to tackle the issue of finding the transformation between two point clouds, which is a problem that has a lot of applications. Among the applications, the most important is the ability to assemble a global model from several point clouds that have been acquired from different point of views. There are mostly two classes of algorithm. The first one focuses on rigid transformations, and the other one focuses on non-rigid transformations. In this project, we focus on cases where the transformation is assumed to be rigid, meaning that we can go from one point cloud to another by only making a rotation and a translation. Even if simple cases can be treated efficiently, in real-life, we face two main troubles : since the data are often not taken from the same point of view, there is no perfect matching between the given point clouds. What is more, acquisition devices often generate noise which can of course not be matches between the two point clouds. These two reason leads us to go beyond the perfect case in order to tackle the issue efficiently and to find a way to take the noise and the missing data into account.

2 The algorithm

In the ideal case, given two point clouds $\mathbf{X}$ and $\mathbf{Y}$, we want to compute the rigid transformation $\tau = [R, t]$ such that $\tau(\mathbf{X}) = \mathbf{Y}$ where R is a rotation and t a translation. In order to do so, the traditional approach is to alternate two steps :

$$\min_{\mathbf{Y}' \subseteq \mathbf{Y}} \sum_{x_i \in \mathbf{X}} ||Rx_i + t - y_i'||_2^2 \tag{1}$$

is the matching step in which we try to find the closest point in $\mathbf{Y}$ for each point of $\mathbf{X}$.

The second step consists in finding the transformation that brings $\mathbf{X}$ to $\mathbf{Y}'$. For this purpose, there are two main approaches.

2.1 Point-to-point algorithm

The first approach is the point to point algorithm. In this version, the problem we want to solve is :

$$\underset{R \in SO(k), t}{\arg\min} \sum_i ||Rx_i + t - y_i'||_2^2 \tag{2}$$

In this case, we penalize a couple of point 'i' based on the square of their L_2 distance.

2.2 Point-to-plane algorithm

The second approach is the point to plane algorithm. It is an higher order approach because it makes use of the normal information on each point of the cloud $\mathbf{Y}'$. In this case, we want to solve the following problem :

$$\underset{R \in SO(k), t}{\arg\min} \sum_i \left(n_i^T (Rx_i + t - y_i)\right)^2 \tag{3}$$

In this equation, we see that what is now important is to fit each x_i in the tangent plane of y_i. This approach has several virtues that will be presented in the experiments.

2.3 Robust distance function

The main contribution of [1] is to bring the use of the so-called robust distance function. Actually, instead of using underlying L_2 norms in equation 1, equation 2 and 3, they suggest to change the metric. As a matter of fact, they suggest that using L_2 norms tends to add an important cost to outliers, which is true because the cost grows quadratically with the distance between an outlier and its matching point.

This problem is pretty well known and used to be tackled by adding confidence weights to the couple of points. The disadvantage of this approach is that it yields a lot of free parameters. The authors of [1] suggest here that using a metric of the shape

$$\phi(x, y) = ||x - y||_2^p$$

would be a better idea since an outlier with a strong L_2 norm will be less penalizing for the global cost function. The authors illustrate this idea by conducting the mathematical developments of this method which yields to an algorithm that is just an adaptation of the classical point to point and point to plane algorithms already presented.

3 Implementation

For this work, I made a C++ implementation of the algorithms presented which can be found at https://github.com/palanglois/icpSparse.

3.1 Dependencies

There are only two dependencies to this implementation which are :

1. Nanoflann : A fast and light-weight implementation of the kd-tree data structure which enables one to efficiently perform k-nearest-neighbor search in a point cloud by recursively dividing the space into octree branches until a single branch contains only a few points.
2. Eigen : A fast and light-weight linear algebra library which was useful to manage the point clouds, the singular value decompositions and the eigen values problems.

All the dependencies are included in the repository, and therefore the code is totally cross-platform and just requires CMake and a C++ compiler to be compiled.

Notice that so far, the program can only import object file and will output the resulting file in the .ply format in order to handle the normals.

3.2 Approximating the normals

In order to perform the point-to-plane approach, we need to be able to estimate the normals for a point-cloud if these ones were not provided. We compute the normal by the k nearest neighbour ACP aproach. For a given point x_i in the point cloud, we compute its k nearest neighbors $(x'_i)_{i\in\{1,k\}}$. We then compute the correlation matrix of these points and we interprete the eigen vector associated to the smallest eigen value of the correlation matrix as the normal to the point x_i.
This approach is efficiently implemented thanks to the kd-tree library Nanoflann.

3.3 The algorithm itself

The algorithm is coded in a IcpOptimizer C++ object. The use of an object was made in order to easily manage the parameters. As a matter of fact, there is only one object for both the point-to-point and the point-to-plane approach. A boolean variable set at the construction tells the object which method to use.
The point matching has been implemented with the kd-tree library. The shrinkage part has been made with 3 steps as suggested in the paper. The variable μ has been set by default to 10 as sugested in the author's reference implementation but it is possible for the user to change it.

3.3.1 Point-to-point

The point-to-point variant has been implemented thanks to the SVD method :
Let $\bar{\mathbf{X}}$ and $\bar{\mathbf{Y}}$ be the centered point cloud to match (each element $\bar{x}_i$ and $\bar{y}_i$ are column vectors). We

compute

$$W = \sum_i \bar{x}_i * \bar{y}_i^T$$

and the singular value decomposition of W :

$$W = U\Sigma V^T$$

Then we have :

$$R = V * U^T \tag{4}$$

$$t = \bar{Y} - R * \bar{X} \tag{5}$$

3.3.2 Point-to-plane

In order to compute the point-to-plane part, we first compute

$$\forall i, c_i = a_i \times n_i$$

(with each c_i being a columng vector).
Then, we compute

$$L = \sum_i \left[\begin{array}{c|c} c_i * c_i' & c_i * n_i' \\ \hline n_i * c_i' & n_i * n_i' \end{array}\right] \text{ and } R = -\sum_i (a_i - b_i).n * \left[\begin{array}{c} c_i \\ n_i \end{array}\right]$$

and we just invert the linear system $LS = R$ with

$$S = \left[\begin{array}{c} \alpha \\ \beta \\ \gamma \\ t_1 \\ t_2 \\ t_3 \end{array}\right]$$

Finally, we have $R = rot(\alpha, \beta, \gamma)$ (the rotation generated by the angles α, β and γ) and $t = \left[\begin{array}{c} t_1 \\ t_2 \\ t_3 \end{array}\right]$.

The system inversion is made efficiently thanks to the Cholesky decomposition in Eigen in order to take advantage of the symmetrical nature of L.

4 Results

4.1 Setup

In order to test the algorithm, a benchmark was prepared. The benchmark was based on the classical bunny mesh. However, in order to have a more realistic study case, both point clouds were partially truncated before being moved as we can see on Figure 1.

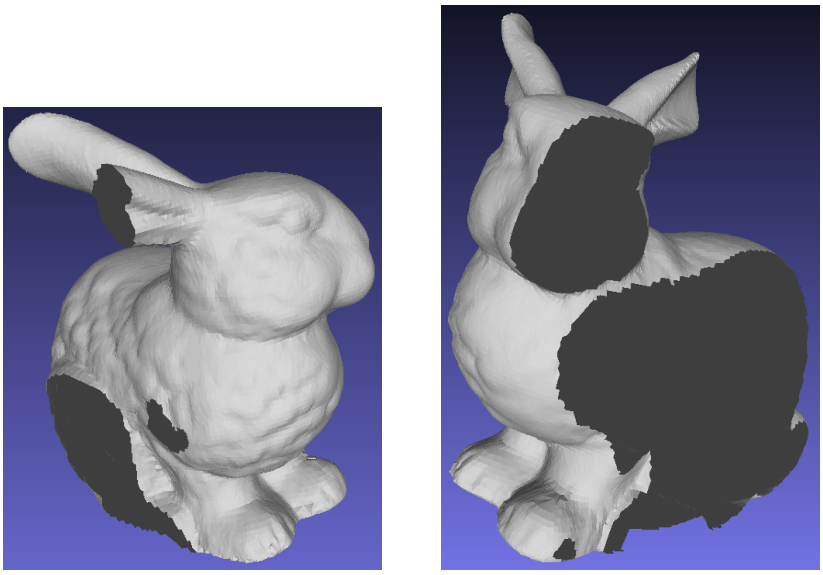
Then, the two meshes are being moved from one an other as we can see on Figure 2.

The initial number of vertice for each point cloud is given on Table 1.

Point cloud 1	31025
Point cloud 2	26186

Table 1: Number of vertice for each point cloud

In this study, it was necessary to have an idea of the influence of noise on the algorithm's performance. As a consequence, uniform noise was added in the bounding box of each point cloud. For both files, copies were made with 250, 500, 1000, 3000, 4000 and 5000 outliers.



(a) First point cloud ($\mathbf{X}$) (b) Second point cloud ($\mathbf{Y}$)

Figure 1: The two basic meshes used for experiments

4.2 Test and Analysis

In order to evaluate the algorithm's performance, it was necessary to set up a metric. Since a transformation τ can be interpreted as a 4x4 matrix in homogeneous coordinates, the following metric was taken. We call τ_{ref} the ground truth transformation. As a consequence, the following error was taken :

$$\epsilon(\tau) = trace((\tau - \tau_{ref}) * (\tau - \tau_{ref})^T)$$

In order to have a reference value, the error before doing anything was computed :

$$\epsilon(Id) \approx 1.0501$$

Experiments were conducted for both versions of the algorithm (point-to-point and point-to-plane). Each time, the evaluated parameters were the number of outliers in each file and the value of p.
For the point-to-point algorithm, results are presented on Figure 4. For the point-to-plane algorithm, results are presented on Figure 5.

4.2.1 Point-to-point

The point to point variant would give convergence result up to 500 outliers. As we can see on Figure 4, the smallest value of p would often give better convergence results than the standard value of 2. We also see that the optimal value seems to be around 0.8 which is a bit more than what was sugested in the article (0.4).

4.2.2 Point-to-plane

We can se on Figure 5 that the point-to-plane variant is more stable to the noise. This can be interpreted by the fact that the computed normals on the uniform noise are of random direction. As a consequence, the criterion helps a lot the algorithm to identify the noise. What is more, with a parameter p lower than 1, we see that the noise does not penalize too much the cost function. As a

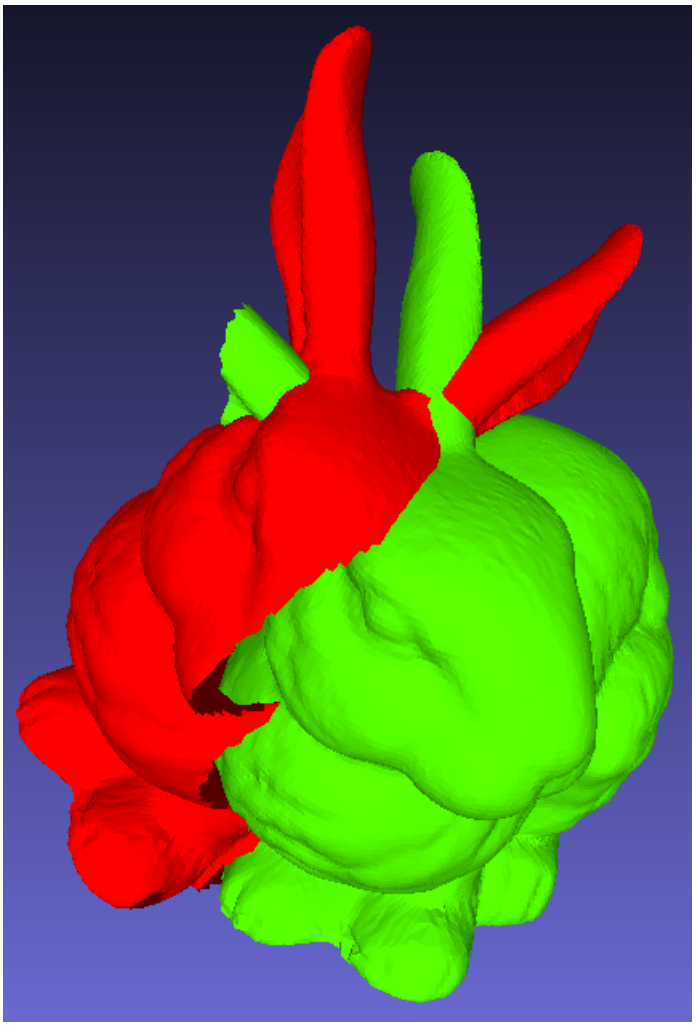


Figure 2: Initial testing configuration

consequence, we obtain good convergence results even for the situation in which 4000 outliers were added on each file.

5 Conclusion

This project was a great occasion to acquire practice with complex linear algebra algorithms in a C++ environment. We can see that the results presented in [1] are relevant and reproducible. However, a global study would also require to test the algorithm on other kinds of noises which are more localized around the surface. As a matter of fact, in this case the normals are more cogent in the noise, and the outliers are less easily detectable by the algorithm.

Regarding the implementation part, the code I produced is reasonable in terms of complexity but could maybe be optimized since some complex parts can be parallelized : the computation of the linear system in the point-to-plane part is independent on each couple of point for example.

Finally, some parameters such as μ were not discussed much in the article. As a consequence, even if the authors claim to have only one free parameter, it may not be so easy to adapt the algorithm to real-life cases. The research on a noise insensitive iterative closest point algorithm remains an open field.

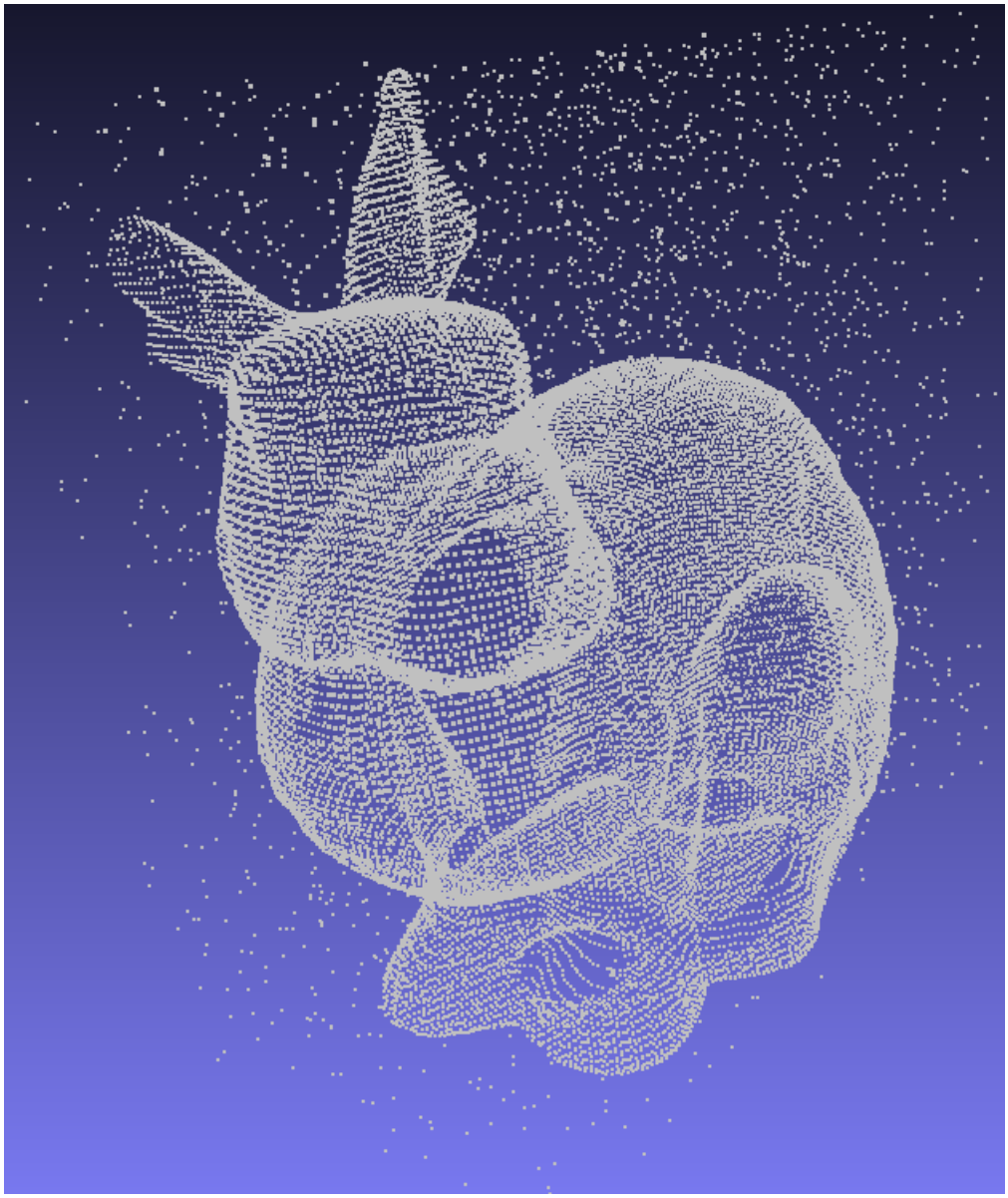


Figure 3: The first point cloud with 4000 outliers added

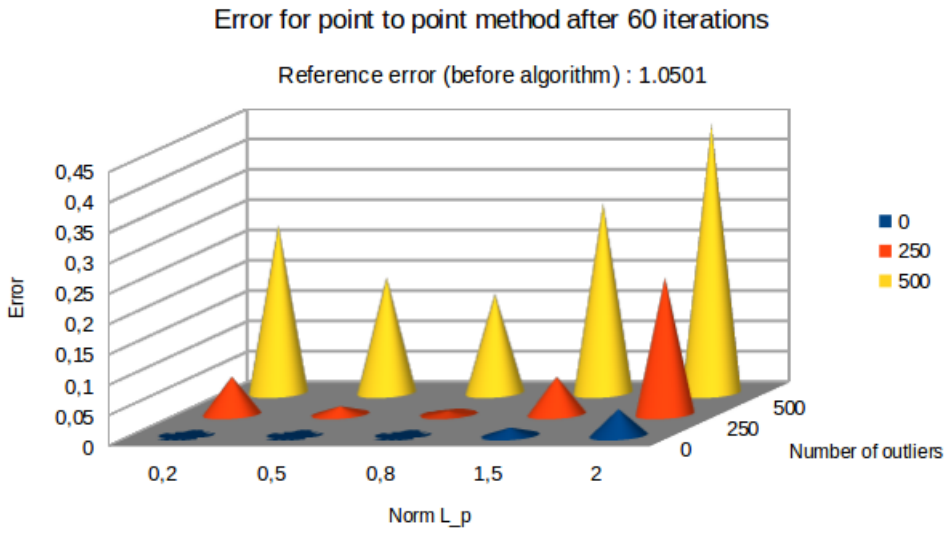


Figure 4: Point to point error analysis

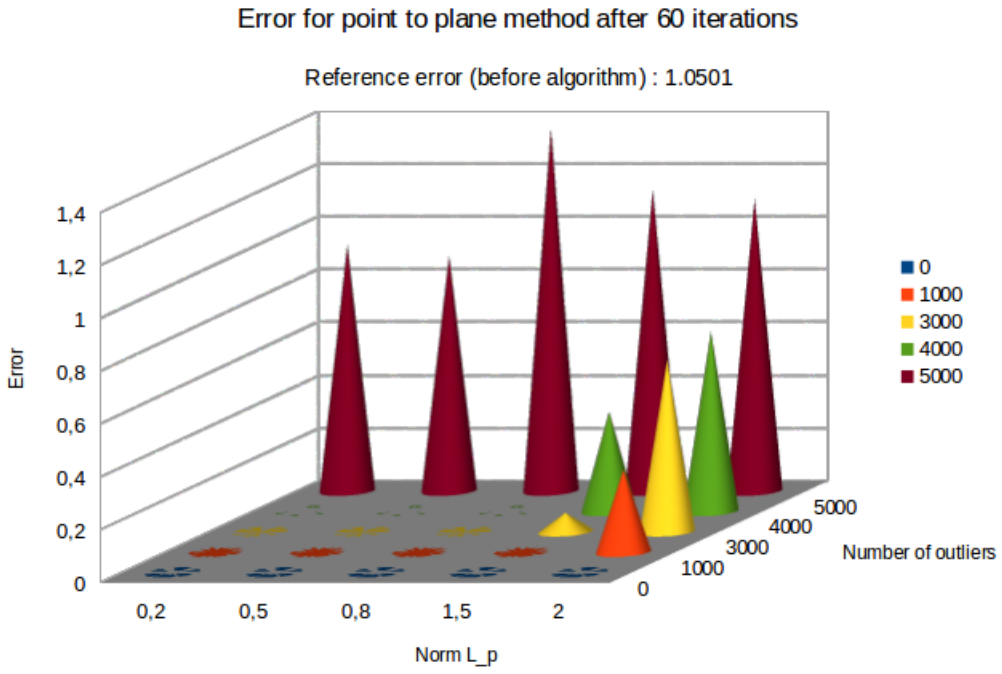


Figure 5: Point to plane error analysis

References

[1] Sofien Bouaziz, Andrea Tagliasacchi, and Mark Pauly. Sparse Iterative Closest Point. In *Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing*, SGP '13, pages 113–123, Aire-la-Ville, Switzerland, Switzerland, 2013. Eurographics Association.