Point Registration in 3D

Antonio Tammaro

Abstract

In this project I implemented a three dimensional version of the renowned algorithm *Iterative Closest Point* (ICP) based on the *Least Squares Estimation* with automatic *Data Association* and in a *Manifold* topological space. Firstly the algorithm was coded in Octave [5] for testing purposes and for a deeper understanding of the problem. Once done the program was entirely rewritten in C++ to achieve better performances and for a possible future use in a larger project. The C++ version uses several libraries like FLANN for data association [2], Eigen for matrices handling [4] and libQGLViewer for the interactive GUI [3]. Firstly a brief introduction of the problem is given, then the mathematical tools and the solution are described, and then in the last section the results are shown and some future improvements are suggested.

1 Introduction

The problem of *point registration* is very common in computer vision and pattern recognition and usually is formulated in this way: Two point sets (or clouds), which have a partial or total overlap region, are given. The task is to find a transformation that aligns or match these two sets. The process of finding such a transformation includes merging multiple data sets into a globally consistent model, and mapping a new measurement to a known data set to identify features or to estimate its pose. Figure 1 shows an example of the process. A point set may be raw data from a *Kinect* or a depth sensor and the result of the process can be used to reconstruct 2D or 3D surfaces from different views.

Theoretically if the correct association is known all the correspondences between the points are established and the transformation between the views could be computed in closed form. This usually isn't the case and a kind of *matching* has to be made. An operation like this one is called data association and it can be made by:

- Probabilistic formulation
- RANdom SAmple Consensus - RANSAC
- Nearest Neighbour search

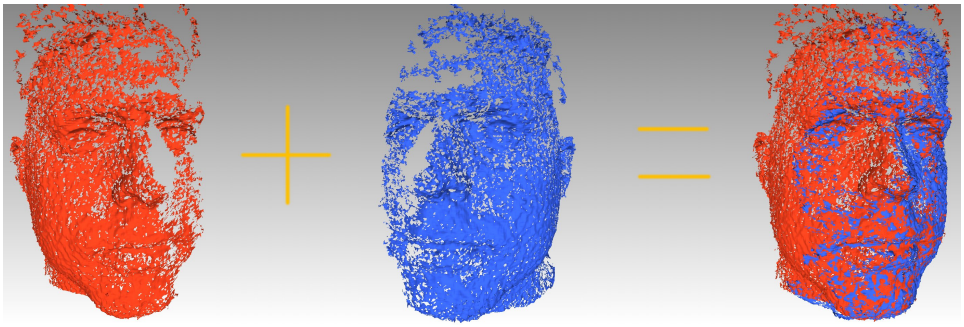

Figure 1: An example of two point clouds with an overlapping region. The transformation between these two sets of points is found and a new model is created from both previous views.

I chose the Nearest Neighbour search for two reasons, it is very effective and really fast. Moreover there is an easy to use and reliable library called FLANN [2] which provide an intuitive framework to execute NN searches. Basically what the FLANN library does is to search, for every point of the cloud, the point in the other view which is topologically closest to the first one. The main advantage of this kind of approach is that the process of searching can be made in a K-dimensional feature space, and for therefore can be applied to a vast variety of problems.

Once a method for data association is established, to meet all the prerequisites for using the *Gauss-Newton minimization algorithm for manifold measurement and state spaces* we only have to make some choices regarding the representation of the points in a *minimal* and *extended* form and on the *box plus/minus* operations. All these choices are explained in detail in Section 2.

2 Algorithm Description

The algorithm has a precise precise structure divided in steps where the input are two unassociated 3D point clouds, and the output is a transformation between the views.

1. Fix the minimal and extended representation for the state variables and the observations. In my implementation I used the *Roll, Pitch, Yaw* to describe the angles in a minimal parametrization and the *rotation matrices* for the extended parametrization, while the observations live in a Euclidean space and are composed simply of x, y and z coordinates. A complete description of these matrix is formulated in [6]. Then our parametrizations of the states and observations are:

$$\mathbf{x} = [t_x, t_y, t_z, \alpha, \beta, \gamma] \tag{1}$$

$$\mathbf{X} = \begin{bmatrix} & \mathbf{R}_{\alpha\beta\gamma} & & t_x \\ & & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$\mathbf{z} = [p_x, p_y, p_z] \tag{3}$$

Where the α, β and γ are respectively the Roll Pitch and Yaw angles and $\mathbf{R}_{\alpha\beta\gamma}$ is the resulting rotation matrix of the three consequent rotations.

2. Define the operation to go from one representation to the other and vice versa, these are going to be our *v2t* and *t2v* operations. This is the critical part because there is a *Gimbal Lock* singularity [7] for a pitch angle of $\pi/2$. Alternative angles representations are considered later in the discussion.

3. Because we want to work with *Smooth Manifolds*, we have to define also the $\boxplus$ and $\boxminus$ operations. These are essential in the computation of the error function and the Jacobian. The $\boxminus$ operation in our case is simply a difference because our observations are defined in an Euclidean space, while the $\boxplus$ is:

$$\mathbf{X} \boxplus \Delta\mathbf{x} = \mathbf{X} \cdot v2t(\mathbf{x}) \tag{4}$$

4. Now we can start with the iterative procedure of least squares, firstly use the NN search to create the associations between the points of the two clouds. For this search we utilized a randomized KD-tree using 4 KD-trees and with 128 checks.

5. Compute the Error function and its Jacobian. As I said before the $\boxminus$ translates in a simple difference, then the error is computed as:

$$\tilde{\mathbf{e}}_k \leftarrow \hat{\mathbf{z}}_k - \mathbf{z}_k \tag{5}$$

For our choice of parameters the Jacobian can be computed simply in closed form computing the derivatives:

$$\mathbf{J}_k \leftarrow \left. \frac{\partial \mathbf{e}_k(\mathbf{h}_k(\breve{\mathbf{x}} \boxplus \mathbf{\Delta x}), \mathbf{z}_k)}{\partial \mathbf{\Delta x}_k} \right|_{\mathbf{\Delta x}_k = \mathbf{0}} \tag{6}$$

which results in

$$\begin{bmatrix} \mathbf{I}_{3\text{x}3} & \mathbf{S}(\mathbf{p}') \end{bmatrix} \tag{7}$$

where $\mathbf{I}_{3\text{x}3}$ is the 3x3 identity matrix and $\mathbf{S}(\mathbf{p}')$ is the skew-symmetric matrix of the transformed point $\mathbf{p}'$.

6. At this stage we have all the elements to define the linear system. The solution of this system is the increment which will be added at every iteration to itself, in this way once the execution terminates the sum of all the increments will be approximatively the 3D transformation between the two views.

7. To visualize the results all the previous steps are embedded in a *QApplication* [8], and more precisely into the init() function. In this way all the computations are made before the launch of the application and stored into a std::vector. Afterwards the results are rendered with the libQGLViewer [3].

3 Discussion

The application has a fast and reliable convergence rate with small perturbations; on average the transformation is found after $15 \sim 20$ iterations with a χ^2 error in the order of 10^{-9}. Some videos and applications are attached to this report to show the results.
However there are some limit cases in which the algorithm does not converge and mainly this happens for two reasons:

- Pitch angle near or equal to $pi/2$ which gives singular values in the *t2v* function.
- Very big values of t_x, t_y, t_z which lead to an error in the Data Association phase.

These problems can be resolved with few but substantial changes in the application:

- Use of *quaternions* instead of euler angles
- Make sure to subtract the corresponding center of mass from every point in the two point sets before calculating the transformation.
- (Optional but appealing) Improve the GUI of the application.

References

[1] Giorgio Grisetti *Notes on Least-Squares and SLAM.* June 2014

[2] FLANN - *Fast Library for Approximate Nearest Neighbors.* `http://www.cs.ubc.ca/research/flann/`

[3] libQGLViewer - *Library for visualization.* `http://www.libqglviewer.com/`

[4] Eigen - *C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.* http://eigen.tuxfamily.org/

[5] Octave - *GNU Octave.* https://www.gnu.org/software/octave/

[6] Angles - *Roll Pitch Yaw angles and thei rotation matrices* http://planning.cs.uiuc.edu/node102.html

[7] Gimbal Lock - *Gimbal Lock explanation* http://en.wikipedia.org/wiki/Gimbal_lock

[8] Qt - *QApplication class* http://doc.qt.io/qt-5/qapplication.html