Contents

List of Figures

List of Tables

Chapter 1

Methodology for Geometry Reconstruction

This chapter present the methodology to reconstruct the geometry of the scene. In Section 1.1, the laser scans of each acquisition are transformed into point clouds. In Section 1.2, two calibration methods, one of which was developed in this work, are described to obtain the extrinsic calibration of the laser scanner. Section 1.3 describes a method to estimate the normals, based on the structure of the point cloud. In Section 1.4, a method to find the transformations between acquisitions is described, to merge the acquisitions into one point cloud. Finally, in Section 1.5, three point cloud filters used in this work are described.

1.1 Point Registration

Each laser scan is a collection of points in polar coordinates, so each range point (r_i, θ_i) is transformed to a point in the laser frame of reference according to Equation (1.1). The angles are uniform distributed between a minimum and maximum angle, θ_{min} and θ_{max}, respectively, so $\theta_i = \{\theta_{min},\ \ldots,\ \theta_{max}\}, i = 1 \ldots N$. The index i is defined as the range index of each laser scan.

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \begin{pmatrix} r_i \cos(\theta_i) \\ r_i \sin(\theta_i) \\ 0 \end{pmatrix} \tag{1.1}$$

Further on, each point is registered in the referencial of the acquisition. According to the transformation graph (see Figure 1.1), there are two transformation from the acquisition frame and the laser scanner frame: the transform from the acquisition frame to the ptu frame T_{acq}^{ptu}, which is dynamic and changes for each laser scan, and the transformation from the ptu frame and the laser scanner frame (T_{ptu}^{laser}), which is static. This two transformations can be chained together to obtain the point in the acquisition frame, according to Equation (1.2).

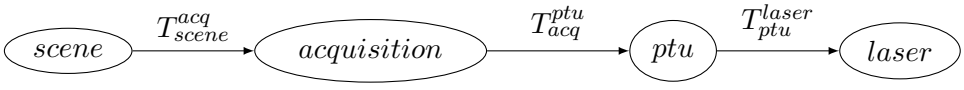


Figure 1.1: Transformation graph

$$P_{i,j} = \begin{pmatrix} x_{i,j} \\ y_{i,j} \\ z_{i,j} \\ 1 \end{pmatrix} = T_{acq}^{ptu} \cdot T_{ptu}^{laser} \cdot \begin{pmatrix} r_i \cos(\theta_i) \\ r_i \sin(\theta_i) \\ 0 \\ 1 \end{pmatrix} \tag{1.2}$$

At this phase, each point has 2 indexes, one for the laser scan index $j = 1 \dots L$ and another for the range index $i = 1 \dots N$, relative to the each laser scan. Despite that point clouds commonly only one index, at this stage the points are structured in a bidimensional structure of $L \times N$ points. This structure is useful in the normal estimation phase, explained in Section 1.3.

This reconstruction phase depends heavily on the transformation from the PTU to the laser scanner. This transformation is obtained by a calibration process and is commonly referred as the extrinsic calibration of the laser scanner. The calibration method used to obtain this extrinsic calibration is explained in detain in Section 1.2.

In conclusion, for each acquisition results a point cloud with $L \times N$ points, where L are the number of laser scans and N the number of range values in each laser scan. Each point can be indexes in a bidimensional index, which can be useful for subsequent algorithms.

1.2 Laser Extrinsic Calibration

The key for a good geometric reconstruction is the laser scanner extrinsic calibration, which has to be accurate, so that every point is correctly registered. Therefore, two calibration methods are here presented: the RADLOCC camera-laser calibration (Section 1.2.1) and a new method developed in this work (Section 1.2.2), that aims to achieve better results than the latter.

1.2.1 Radlocc Method

[4] presents a method for auto calibration of a camera with a laser scanner. This method, known as Radlocc, uses information from both sensors and tries to find point correspondences to optimize the calibration. In this work, this method was used together with the extrinsic calibration method (explained in section ??), to obtain the full extrinsic laser calibration.

To use this method, the user has to obtain a calibration dataset, which is a set of synchronized images and laser scans containing a chessboard in multiple poses. The chessboard serves as the calibration object, which is the link between the two sensors. In this work, a ROS package was developed to handle this capture and to convert between the ROS messages and the RADLOCC format. Its source code and all documentation can be found in `https://github.com/bernardomig/radlocc_calibration`. In the 3D scanner, the laser scanner was positioned such that the laser scans are horizontal and about 20 to 30 images were taken per dataset. Such images are shown in Figure 1.2.

First, a chessboard extraction algorithm finds both finds the intrinsic calibration of the camera, as well as the poses of each chessboard in the camera coordinate frame. Then, laser scans are segmented into board/background, and all the board points are extracted, as seen on Figure 1.3.

Then, the reprojection error of the laser scans points to the chessboard plane are computed, and the transformation from

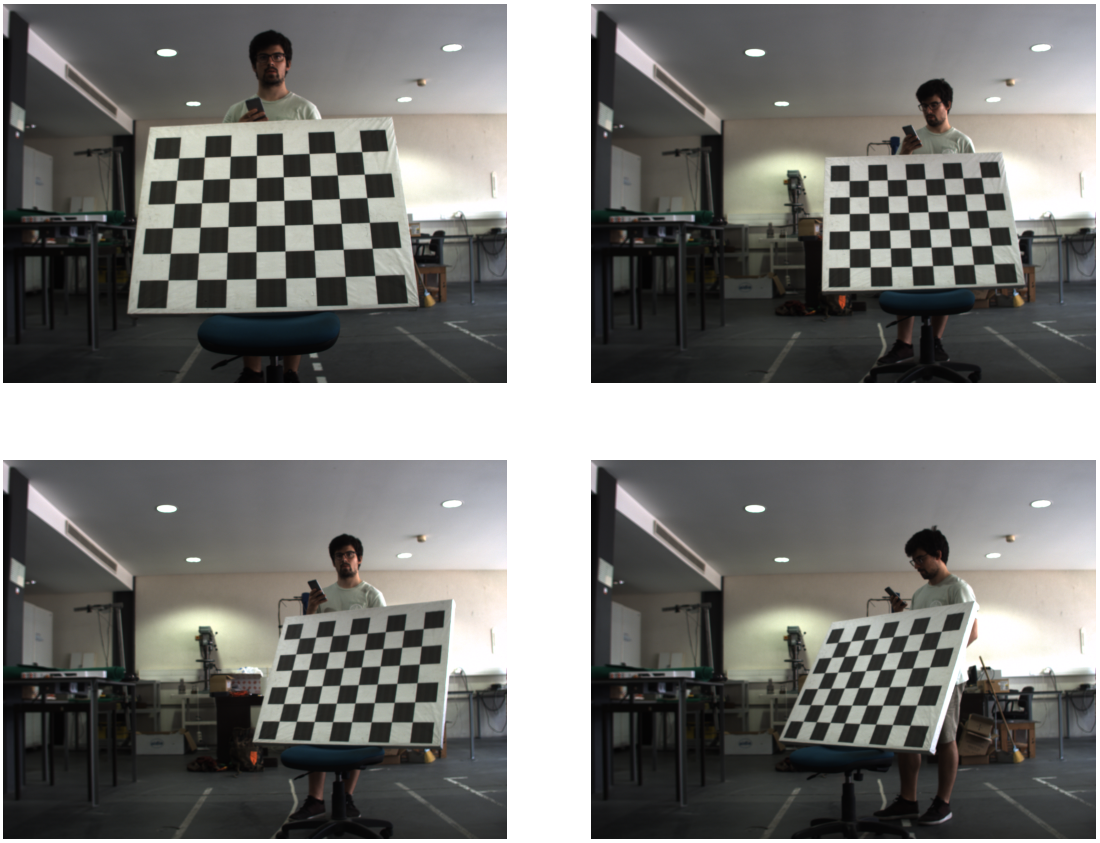


Figure 1.2: Images captured for RADLOCC

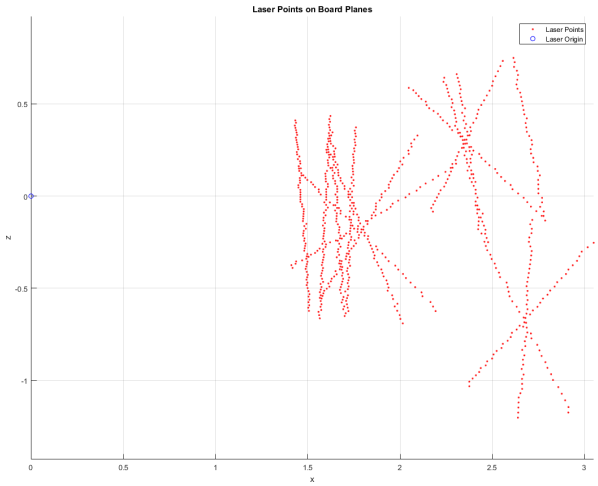


Figure 1.3: Radlocc laser scans chessboard extraction

Then, the reprojection error of the laser scans points to the chessboard plane are calculated, depending of the transformation from the camera to the laser scanner. The reprojection error is minimized during the calibration optimization, and the transformation from the camera to the laser if found.

Finally, the extrinsic calibration can be calculated as in Equation (1.3).

$$T_{calibration} = T_{PTU}^{camera} \cdot T_{camera}^{laser} \tag{1.3}$$

1.2.2 Proposed Method

A new method was developed in this work to calibrate the laser scanner in this system. One of the key differences to the previous method (in Section 1.2.1) is that the calibration is done only using the laser range data, and the calibration from the PTU to the laser scanner is directly find. The hypothesis that this method relies, is that in a good calibration the deviation of a point set is minimal. In other words, in a point set representing a planar surface, the deviation from the points to the planar surface is the lowest, if the extrinsic calibration is the correct one.

This method is, therefore, an optimization problem. For each extrinsic calibration transformation T, corresponds a point cloud P, following the method shown on Section 1.1. Then this point cloud is evaluated by a loss function, which determines quantitatively how good each generated point cloud is. Finally, an optimizer will find the transformation T that minimizes the loss function. Each one of this steps is described in detail next.

Segmentation

This calibration method uses an acquisition as its dataset, which is a big advantage. Also, a point cloud has to be generated using a estimation of the calibration transformation. This point cloud does not have to be geometrically accurate but needs to be sufficient for the plane segmentation, which is done manually prior to the calibration. In this work, the software CloudCompare was used to segment the point cloud into multiple planes, and the data was saved as a scalar index in each point. An example of a segmentation can be seen in Figure 1.4, where each cluster is represented with a different color.

The segmentation was not done automatically because most segmentation algorithms, for example the RANSAC algorithm, were not capable of achieving a good segmentation for the initial estimate, because the point cloud had too much deformation. On the other hand, manual segmentation is easy and accurate, considering that it is a one-time process.

During the optimization, this segmentation serves as a blueprint for all the segmentations. Each point cloud is generated in the same way, so the sequence of points is always the same. Therefore, it is always possible to match any point on the generated point cloud to the point in the segmented point cloud, and get the corresponding cluster index for all the points.

Loss Function

A loss function, or cost function, is a measure in optimization that compares the result of a model with its expected result, and returns a value that signifies the difference between the two. More concretely, in this calibration the loss function has two components: the loss function per cluster and the loss function per point cloud, which combines the loss of all the clusters.

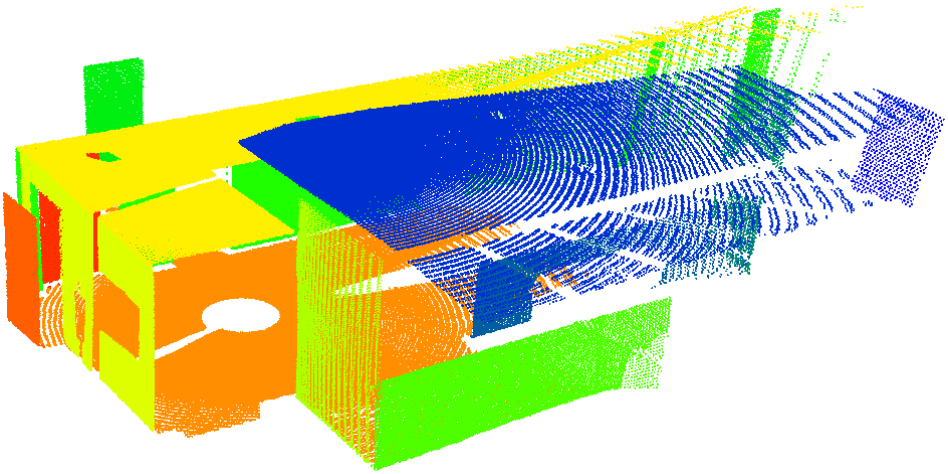


Figure 1.4: Example of a plane segmentation. Each color represents a cluster

To start with, the plane equation for each cluster is found. The same method as in Section 1.3, Principal Component Analysis, is used. First the centroid of each plane is found, which is the same as the mean value of all the points $\bar{p}$ (Equation (1.4)). Then, the covariance matrix $\mathcal{C}$ is calculated (Equation (1.5)).

$$\bar{p} = \sum_i p_i \tag{1.4}$$

$$\mathcal{C} = \sum_i (p_i - \bar{p}) \otimes (p_i - \bar{p}) \tag{1.5}$$

Then, the principal axes of the plane is find by an eigen decomposition of the covariance matrix. The smallest eigenvalue λ_3 will be the variance σ^2 of the cluster. In other words, σ^2 is the mean square of the orthogonal distance of all points in the cluster to the plane. So, σ^2 can be quantitatively to measure the loss for each plane. Formally, let us admit that the σ^2 has two components: the statistical error of the laser sensor σ^2_{sensor}, which is not affected by the calibration and a second component σ^2_{calib}, which results from the calibration error. Thus, it is possible that by minimizing σ, a exact calibration can be obtained. Therefore, the loss of each cluster will be the σ value, which is also known as the Root Mean Square Deviation or RMS.

Next, the losses of the clusters are combined into a scalar value, which is the loss of the point cloud. The method found was to, again, calculate the RMS of the values of the partial losses $loss_i$, according to the Equation (1.6). This value is expected to be minimal when all the partial losses are minimal which, according to this hypothesis, corresponds to a correct calibration.

$$RMS = \sqrt{\sum_i^N x_i^2} \tag{1.6}$$

Paramerization

The parameters in this calibration should define a geometric transformation in space, which is, in the end, a transformation matrix (Equation (1.7)). This transformation can be decomposed into two components, a translation and a rotation. The translation can be represented as the vector $t = (t_x, t_y, t_z)$, and the rotation can be represented as a 3×3 rotation matrix R. Since a rotation matrix has only $3 \times 3 = 9$ elements but only 3 degrees of freedom, another parameterization has to be used to represent a rotation. Popular parameterization for rotations are euler angles, quaternions and axis/angle representation.

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.7}$$

However, not all representations are suitable for an optimization. In fact, in [2] the term **fair parameterization** was introduced: a parameterization is called fair, if it does not introduce more numerical sensitivity than the one inherent to the problem itself. Therefore, fair parameterization are a requirement for optimizations, as it increases the changes of convergence. For example, euler angles, which are probably the most used angle parameterization, are not suitable for optimizations, because they do not yield smooth movements, each rotation is non-unique and, most notably, there are singularities, so-called *gimbal-lock* singularities, where one DOF is lost [3]. Also, quaternions are not suitable for optimizations, because quaternions have 4 components which are norm-1 constrained. Despite being a fair parameterization, quaternions introduce some complexity in the algorithm to handle the norm-1 constrain, so they are not usually used for optimizations.

The axis/angle parameterization is the most widely used to represent a rotation in an optimization, as it is a fair parameterization and has only three components. Any rotation can be represented as a rotation around an axis a, by an angle θ. Since a only represent the direction of the rotation (hence only has 2 degrees of freedom), it can be combined with the angle θ into a single vector ω, as in Equation (1.8).

$$\begin{aligned} \theta &= |\omega| \\ a &= \frac{\omega}{|\omega|} \end{aligned} \tag{1.8}$$

Computing the rotation matrix from ω is done using the Rodrigues' formula (Equations (1.9) and (1.10)) [3].

$$R = I + \frac{\sin\theta}{\theta}[\omega] + \frac{1 - \cos\theta}{\theta^2}[w] \tag{1.9}$$

$$[w] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{1.10}$$

In conclusion, the parameter vector will have 6 values: 3 representing the translation (t_1, t_2, t_3) and 3 representing the rotation in the axis/angle representation (r_1, r_2, r_3). So, the parameter vector is shown in Equation (1.11).

$$P = \{t_1, t_2, t_3, r_1, r_2, r_3\} \tag{1.11}$$

Optimizer

The optimization is performed by the Powell's method. This method finds a local minimum of a multi-dimensional unconstrained function, and does not require the gradient of this function, which fits this particular optimization. This method is implemented in the python scientific library SciPy.

Overview

To sumarize, the overview of the entire steps of the calibration is shown in Figure 1.5.

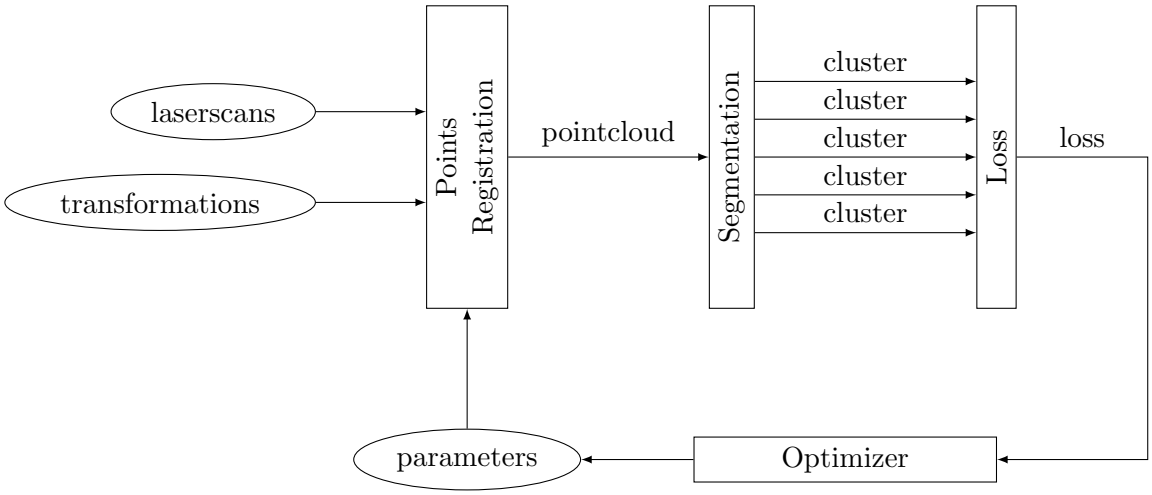


Figure 1.5: Calibration Overview

1.3 Normal Estimation

Surface normals are an important property of geometric surfaces, and are fundamental for meshing algorithms and computer graphics, as for example, to calculate the shading and other visual effects [1]. As an example, the Stanford Rabbit model rendered with and without normals are show in Figure 1.6. Normal estimation is quite trivial for surfaces, but for point clouds the process is quite not as easy. Usually there are two ways to estimate the normals: either by meshing the surface first, and then calculate the normals for the mesh, or using the point cloud itself to infer the normals. However, most meshing algorithms already require the normals to achieve a good result, so the latter option is more effective.

The most common solution is, for each point, to find the k closest point, defined as the k-neighborhood of a point, and calculate the normal of the best-fitting plane formed by this points. However, finding the k-neighborhood of all the points in a point cloud has a time complexity $O(N \log N)$, so it can become quite slow for point clouds with a large number of points. In this work a new solution was used to find the closest points, exploiting the bidimensional structure of the point cloud. This solution has a linear time complexity $O(N)$, which makes it a valuable solution for large point clouds.

The solution were presented acknowledges that each point in the point cloud resulting from Section 1.1 has two indexes, one for the range index (i) and one for the laser scan j. For

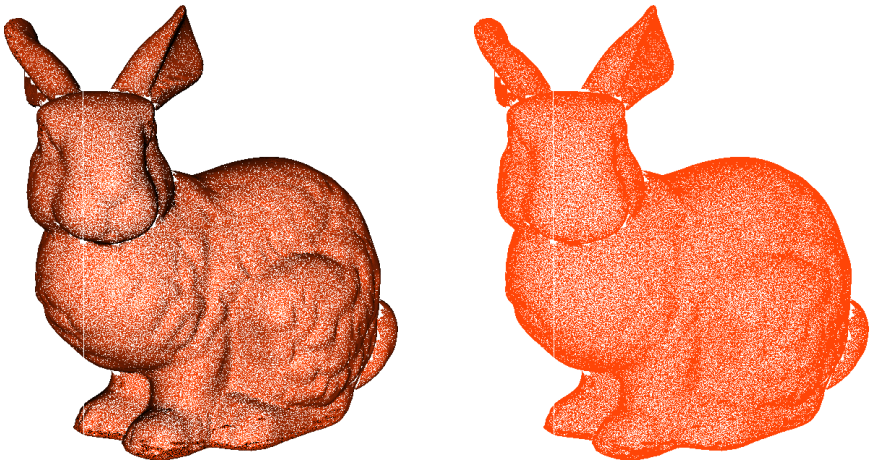


Figure 1.6: Stanford Rabbit Rendering with (left) and without (right) normals

each laser scan, each point p_i has a neighborhood $p_{i-k}, \ldots, p_{i+k}$, because each subsequent point has an increasing angle to the previous point. Between successive laser scans each point has an increasing angle (the pan angle) to the previous one. Therefore, for this algorithm, the neighborhood of each point is show in Equation (1.12). The value of k_1 and k_2 have to be adjusted for a better result, because if the values are too big, fine details are going to disappear and edges are going to be smeared, and on the other hand if the values are too small, the surface will appear as too noisy.

$$neighborhood(p_{i,j}, k_1, k_2) = \{p_{i-k_1,j-k_2}, \ \ldots, \ p_{i+k_1,j+k_2}\} \tag{1.12}$$

Then, for each point, the tangent plane that fits the neighborhood is calculated, which in turn is a least-square plane fitting problem. This is usually solved by an analysis of the eigenvalues and eigenvalues of the covariance matrix of the neighborhood points. This method is also known as the Principal Component Analysis.

To start with, for any neighborhood, the covariance matrix is assembled as in Equation (1.13), where n is the number of points in the neighborhood and $\bar{p}$ is the centroid of the points ($\otimes$ is the tensor product). Then, an eigen-decomposition is performed in the covariance matrix to calculate the eigenvalues λ_i and the corresponding eigenvectors $\mathtt{v}_i$. Finally, the direction of the normal vector of the point is the eigen vector corresponding to the smallest eigenvalue, so $\hat{n} = \pm\mathtt{v}_3$.

$$\mathcal{C} = \frac{1}{n}\sum_{i}^{n}(p_i - \bar{p}) \otimes (p_i - \bar{p}) \tag{1.13}$$

Now, the orientation of the point has to be defined, because the result of the PCA is ambiguous, which may lead to inconsistent normals in the point cloud. In this case, the solution found was to orientate the normals towards the frame of the 3D scanner, which for each acquisition if the zero position. Therefore, each normal has to satisfy the Equation (1.14).

$$\hat{n} \cdot p < 0 \tag{1.14}$$

Further, there should be some filtering, specially for points at an edge, because the neighborhoods method described here is going to fail in this case. So, a proposed solution is to

reject any point that exceeds a certain threshold distance to the center point. This has the advantage that the normal estimation can still work in the edges.

1.4 Acquisition Registration

During an acquisition multiple acquisitions are done and to each one corresponds a transformation (position and orientation) to the scene referencial. In this section, a method is described to find each one of this transformations, so all the acquisitions are merged into a single point cloud. The method chosen is the ICP or Iterative Closest Point. This method is capable of aligning two point clouds, the reference and the target point cloud, by finding the transformation between the second to the first one. This is also known as point cloud registration.

1.4.1 ICP

This method can formally be described as follows: let P be the target point cloud and Q the reference point cloud. Then, the aim of the registration is to estimate the transformation T from the referencial of P to Q by minimizing the error function $error(P, Q)$ in Equation (1.15), where $T(P)$ is the result of the application of the transformation T to the point cloud P.

$$T = \underset{T}{argmin}(error(T(P), Q)) \tag{1.15}$$

The error function $error(P, Q)$ is computed on pair of points that are associated between the two point clouds. This association is, ideally, between points that are closest in position in both point clouds. Then, the distance between the matching points (p_i, q_i) are used in the error function in Equation (1.16). The matching algorithm can be based on features or geometric properties, so a better matching can be found. In this work, a simple point-to-point matching.

$$error(P, Q) = \sum_{(p_i, q_i)} |p_i - q_i| \tag{1.16}$$

In order to make the error function more robust, outlier points can be removed first from the match list. In addition, weights w_i can be associated to each matching points (p_i, q_i) to increase or decrease the influence of each matching points in the error function. As an example, normals can be used as a weight, so points with similar normals ($w_i = n_{p_i} \cdot n_{q_i}$) have a greater influence in the error function.

However, the result of this minimization is always always dependant on the association between the points, which, unless the descriptors are good enough (like visual correspondences), the matching is not perfect, and is worst the farther apart both point clouds are. The idea behind the ICP algorithm is that, even with a bag association, the resulting estimate can be used to find a better one. So, the ICP algorithm creates a series of transformation T_i at each iteration, yielding a new transformed point cloud P_i. Then, the next transformation is found:

$$T_{i+1} = \underset{T}{argmin}(error(T_i(P_i), Q)) \tag{1.17}$$

Finally, the final transformation estimate is the composition of all the intermediary transformations:

$$T = T_1 \circ T_2 \circ \cdots \circ T_N \tag{1.18}$$

1.4.2 Multiple Point Cloud ICP

However, ICP can only register pairs of point clouds, whereas this work requires a registration of N point clouds, corresponding to n acquisitions. So, a technique has to be found so that the ICP algorithm can be used with n point clouds. Three of this such techniques are now describes, ordered by their complexity.

The first approach and the easier one to implement is to register each point clouds sequencially. In other words, this method registers the point cloud P_i to the point cloud P_{i-1} and the transformation T_{i-1}^{i} is found. The final accumulated point cloud is assembled using the Equation (1.19). This method is the one that requires less overall registration, but has the downside that the transformation errors increases for each successive point cloud. This approach is shown in Figure 1.7a.

$$P = \bigcup \left(T_1^2 \circ T_2^3 \circ \cdots \circ T_{i-1}^{i}\right)(P_i) \tag{1.19}$$

The next approach is widely used in robotics for Simultaneous Location and Mapping (SLAM). This method holds an accumulated point cloud A in memory, and each new incoming point cloud P registers to the accumulated point cloud and afterwards it is merged into A, which is then used for the next iteration, as shown in Figure 1.7b. It has the advantage that each new registration is done against a wider point cloud and has a smaller influence than in the previous method. Also, at each iteration the current pose of the 3D scanner is obtained, which is used as an initial estimate for the next iteration. However, the accumulated point cloud grows at each iteration, so a down-sampling is done at each iteration to keep the number of points bounded. In conclusion, each iteration can be calculated as:

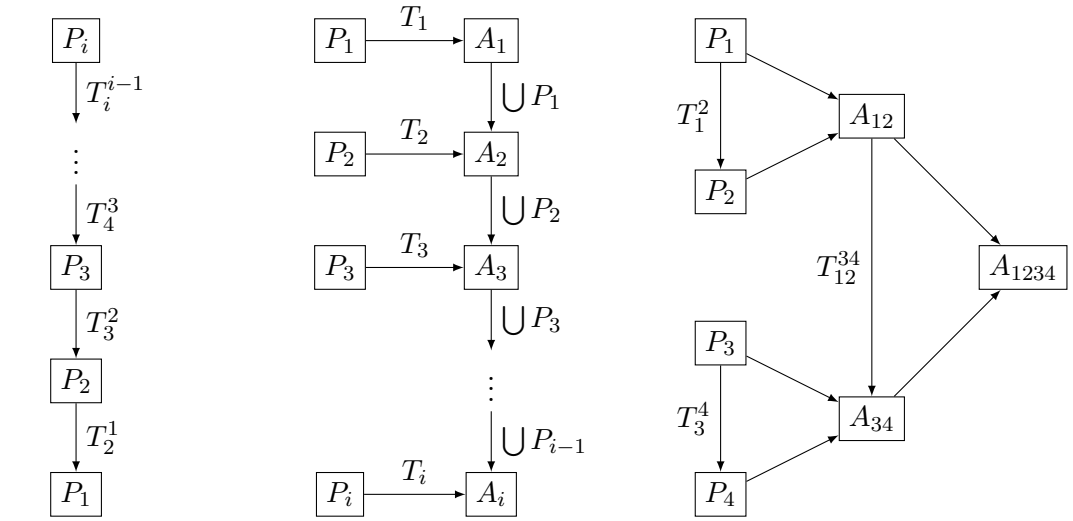
$$T_i = ICP(A, P_i, T_0 = T_{i-1}) \tag{1.20}$$

$$A_{i+1} = Ai \bigcup T_i(P_i) \tag{1.21}$$

The last approach is the most complex. The idea of this approach is to minimize the number of transformation combinations, to minimize the propagation of the error. In particular, the registrations for the N points clouds are done pairwise and are merged together to create $N/2$ point clouds. Then, this process is done recursively until an unique point cloud is obtained. This way, the maximum number of transformation combinations are equal to the number of levels of the tree, which is $\log_2(N)$, instead of N combinations in the first approach. This algorithm is formalized in Equations (1.22) and (1.23), for a list of point clouds $S = \{P_1, P_2, \ \ldots, \ P_n\}$. At each level l a new list of point clouds ${}^{l}P$ and transformations ${}^{l}T$ are computed, as shown in Figure 1.7c.

$${}^{l}T = \left\{ICP({}^{l-1}P_1, {}^{l-1}P_2), \ \ldots, \ ICP({}^{l-1}P_{n-1}, {}^{l-1}P_n)\right\} \tag{1.22}$$

$${}^{l}P = \left\{{}^{l-1}P_1 \bigcup {}^{l}T_1({}^{l-1}P_2), \ \ldots, \ {}^{l-1}P_{n-1} \bigcup {}^{l}T_{n/2}({}^{l-1}P_n)\right\} \tag{1.23}$$



(a) First method (b) Second method (c) Third method

Figure 1.7: Multiple Point Cloud ICP approaches

In conclusion, three methods are possible to extend the ICP algorithm to multiple point clouds. After all the registrations are performed, point cloud can be assembled from all the point clouds, to form a big point cloud of the final scene. There is, however, a limitation of all this methods, because all of them have the principle that every point cloud is close to the previous one, which can be false. In this work, this was ensured in the capture methodology.

1.5 Filters

The final point cloud, after the assembly from every acquisition's point cloud, can have unnecessary or redundant information, which can make the point cloud too big for any use. A common solution is to use filters to remote unnecessary points and downsample the point cloud.

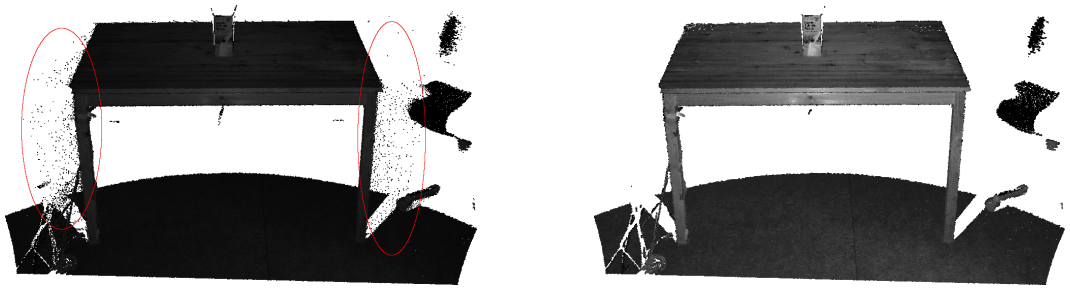
1.5.1 NaN Removal

The first filter is the NaN removal. In the acquisition, any range that is not measured is stored as a NaN, to signal that they are missing. During the point registration phase, all this missing ranges remain as nan, and should be removed, because their information is irrelevant and take as much space as a real value. So, each point that contains a NaN value is removed from the final point cloud.

1.5.2 Statistic Outlier Removal

Usually point clouds contains different point densities, dependent on the distance of the object to the sensor. Also, measurement errors also occur next to edges or corners. As a result, point clouds tend to have sparse outliers that can affect subsequent algorithms, like segmentation or registration algorithms. A usual solution is to perform an statistically analysis on each point, removing the ones that do not reach a certain criteria. In particular, the mean distance

of each point to its neighbors is computed, and if this distance is outside an interval centered in the mean of all the distances, then it is removed. An example can be seen in Figure 1.8.



(a) Before SOR (b) After SOR

Figure 1.8: SOR filter in a point cloud

1.5.3 Voxel Grid Downsampling

This method downsamples, that is, reduce the number of points of a point cloud, using a voxel grid. A voxel is a cubical space and is the element in a tridimensional grid. So, each point in the point cloud belongs to some voxel. Then, in each voxel, all the points are represented by their centroid. This is an effective and fast method to downsample a point cloud. The level of detail can be parameterized with the voxel leaf-size (the size of each voxel in the x, y, z direction). A smaller leaf-size maintains more details but generates a bigger point cloud. A bigger leaf-size does not keep as much detain but generates a smaller point cloud. As an example, Figure 1.9 shows the Lucy dataset after a voxel grid downsampling with different leaf size values: Figure 1.9a with 2 mm (288.000 pts), Figure 1.9b with 5 mm (55.000 pts) and Figure 1.9c with 2 mm (18.000 pts).

(a) Leaf size of 2 mm (b) Leaf size of 5 mm (c) Leaf size of 8 mm

Figure 1.9: "Lucy" scan after a voxel grid downsampling with different leaf sizes

Bibliography

[1] *Estimating Surface Normals in a PointCloud*. URL: http://pointclouds.org/documentation/tutorials/normal_estimation.php (visited on 09/02/2018).

[2] Joachim Hornegger and Carlo Tomasi. "Representation Issues in the ML Estimation of Camera Motion". In: *ICCV*. 1999.

[3] Jochen Schmidt and Heinrich Niemann. "Using Quaternions for Parametrizing 3-D Rotations in Unconstrained Nonlinear Optimization". In: (Jan. 2001).

[4] Qilong Zhang and R. Pless. "Extrinsic calibration of a camera and laser range finder (improves camera calibration)". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. Sept. 2004, 2301–2306 vol.3. DOI: 10.1109/IROS.2004.1389752.