Contents

List of Figures

List of Tables

Chapter 1

Methodology for Scene Capture

This chapter describes the concepts and methodology around a capture of a scene. To further explain the methodology, some core concepts need to be defined first:

Acquisition is a collection of sensor data collected in a specific pose in the scene. This sensor data are ether images, taken from a camera, or laser scans, taken from a 2D laser scanner. Each one of this sensor data is always tagged with temporal and spatial information, to identify where and when this data was recorded.

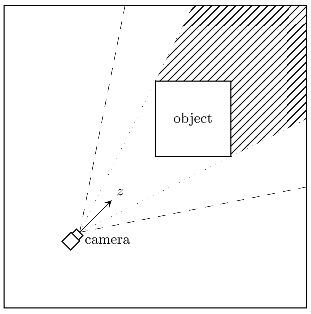
Capture is a collection of acquisitions taken from the same scene, from different poses and times.

A single acquisition has only limited information about the scene and is insufficient to create a complete reconstruction. This is often caused by occlusions, hardware limitations (like a small aperture, low range limits or low resolution) or environment factors (like reflective surfaces or lightning conditions). To circumvent this limitations, multiple acquisitions are taken from the same scene, to get enough data from it. However, this also comes with some challenges, for example, how to merge all the acquisitions and how to handle with all the redundant data.

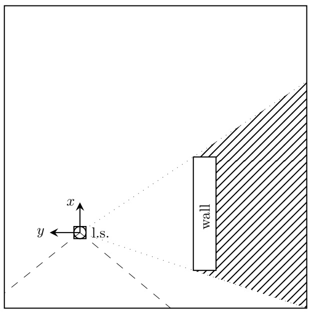
So, acquisitions and captures are different levels and each one has a different method and objectives. In an acquisition level, the focus is on how to operate the scanner and define how the data is recorded. In a capture level, the focus is on how to plan multiple acquisitions so a good reconstruction is possible. In the following sections, both acquisitions and captures are further explained.

1.1 Acquisition

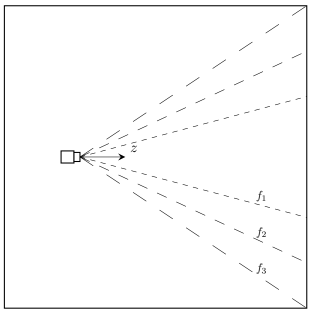
An acquisition is a collection of sensor data (laser scans and images) collected by the sensors in the scanner. Both sensors sample only a small subset of the whole environment: the laser scans only have points from a planar region of the space and cameras are limited by their focal length. To overcome this limitation, both sensors are moved to different poses in space to cover a wider space. In this case, the cause of movement of the sensors is the movement of the joints of the PTU.



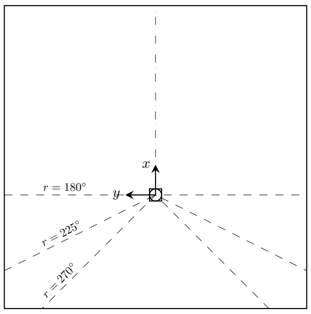
(a) Occlusion in a camera



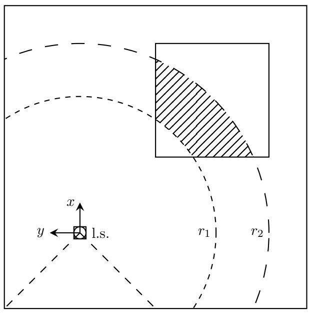
(b) Occlusion in a laser scanner



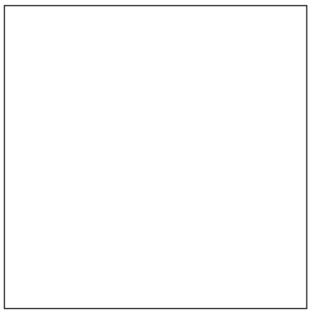
(c) Different focal points of a camera



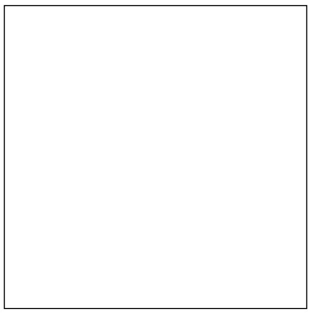
(d) Radial aperture of a laser



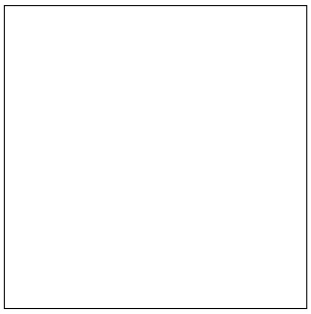
(e) Range limitations



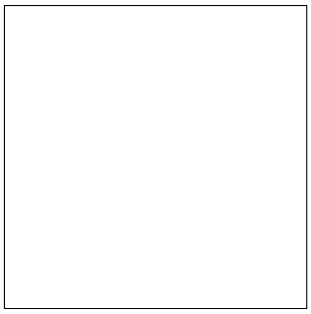
(f) yet



(g) yet



(h) yet



(i) yet

Figure 1.1: Limitations of a single acquisition

1.1.1 Movement Programming

To program the motion of the PTU joints, a list of waypoints in pan and tilt are defined and the joints move from waypoint to waypoint. The waypoints are defined in a grid in the joint-space and the movement between waypoints is the one that defines the shortest path possible and most of the movement is done in pan. So, each acquisition is parameterized with the following parameters: the range (minimum and maximum angle) of pan/tilt, the speed of each join and the number of waypoints in pan/tilt. An example of this parameterization can be seen in Figure 1.2.

Once the movement of the scanner was defined, the next step is to define when to record the laser scans and the images, according to it. Because of the nature of both sensors, it was established that laser scans are captured continuously during the pan movement between the waypoints and images are captured at every waypoint.

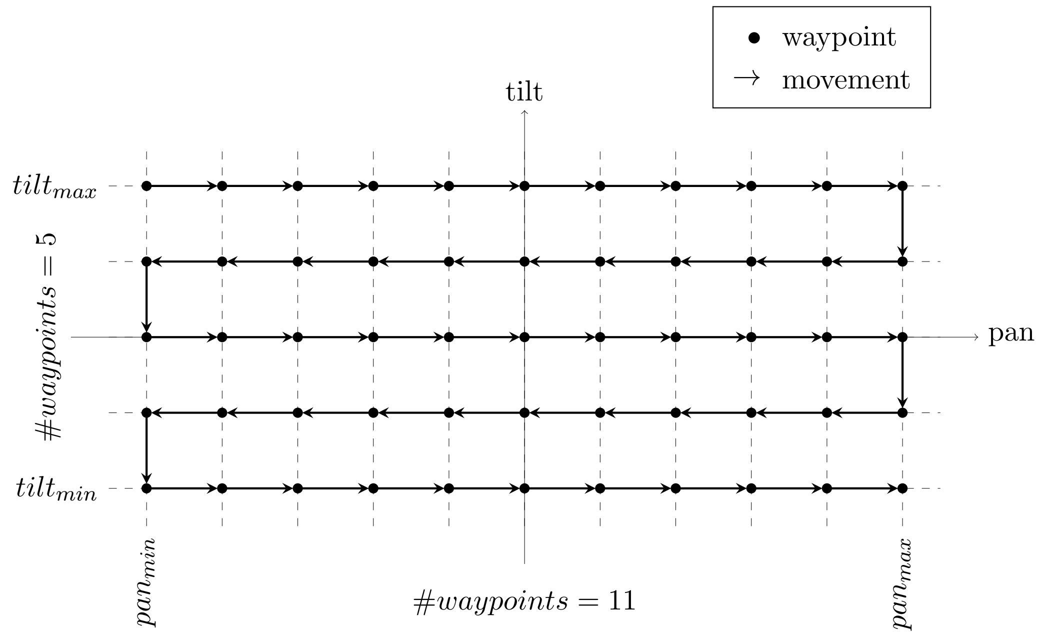


Figure 1.2: Waypoints and movements in the pan/tilt joint space

1.1.2 Parameterization Considerations

This methodology has the following implications:

- The pan and tilt range is only limited by the PTU capabilities, but it beneficial to use the maximum range possible, in order to get as much data as it is possible, even if some data is redundant. In this work, the mobile scanner have a laser configuration such that the interpolation if different tilt angles is almost exclusively redundant data. However, this improves the final reconstruction by increasing the density of the point cloud.
- The number of laser scans recorded is going to depend on the pan speed and the frequency of scanning of the 2D laser scanner. So, it is expected that a laser scanner with a lower scanning frequency to require a slower speed compared to a faster one, to get the same results.

- The camera used in this work did not have stabilization so, to get sharp images, a complete immobilization was required in each waypoint. This was achieved by setting a time between the stop of all joints and the capture of the image by the camera. In this work, a time of 1.5 s was enough.

- The waypoints' angle increment has to be enough so that part of the previous image appear in the next image, so that every observable part of the scene is seen at least once. This depends heavily on the focal point of the camera: the bigger the focal point, the least area it captures and more waypoints are required.

1.1.3 Default Parameters

1.1.4 Acquisition node

To implement this functionality, a ROS node was developed according to the previously defined specifications. This node, called *single_acquisition_node* is present in the *lemonbot_acquisition* package and the way is implemented is the following: the PTU movement is controlled by it and the selected messages are republished into a new topic. For convenience, all the acquisition topics are republished into the */acquisition* namespace. So, during an acquisition, two topics can be found, each one corresponding to each sensor, inside this namespace: the laser scans are in */acquisition/laserscans* and the images are in */acquisition/images.* This idea of republishing all the important messages greatly improved the acquisition organization, so all the topics that were required were also republished into this namespace. This topics were the */acquisition/camera_info*, containing the intrinsic parameters of the camera, and */acquisition/tf* and */acquisition/tf_static*, containing all the transformations of the robot.

Now, data from this topics need to be saved permanently, so this was done using a ROS tool called *rosbag*, that saves all the data from a predefined set of topics into a binary file called a *bag* file. This was a easy and powerful solution, because it allows the acquisition to be reproduced again, by republishing all the messages back into the system. To save a set of topics, a node called *record* from the *rosbag* package is run with the list of topics that required to be recorded into disk. In this case, the required topics are all the topics inside the */acquisition* namespace.

To streamline the acquisition process, all this components (the acquisition node, the topic republisher nodes and the rosbag record node) can be all launched through a *launch file.* A set of all the parameters required for each acquisition can be override over the default parameters shown in Section 1.1.3. Therefore, running an acquisition just requires a single command:

```
roslaunch lemonbot_acquisition single_acquisition.launch \
    pan_min:=-90 pan_max:=90 pan_vel:=10 pan_nsteps:=25 \
    tilt_min:=-15 tilt_max:=15 tilt_nsteps:=5
```

In conclusion, running the previous command will run an acquisition and in the end, a bag file will be present, with the topics *images*, *laserscan*, *camera_info*, *tf* and *tf_static*, therefore all the information relevant for the reconstruction.

To have a better insight in the bag file, a tool called *rosbag info* can be used. All the details about when the calibration took place, how long it took as well as how many messages it contains are printed. An example of this information is:

```
path:        acquisition_2018-09-07-16-01-46.bag
version:     2.0
duration:    4:53s (293s)
start:       Sep 07 2018 16:01:47.11 (1536332507.11)
end:         Sep 07 2018 16:06:40.96 (1536332800.96)
size:        87.0 MB
messages:    6690
compression: none [16/16 chunks]
types:       sensor_msgs/CameraInfo [c9a58c1b0b154e0e6da7578cb991d214]
                sensor_msgs/Image      [060021388200f6f0f447d0fcd9c64743]
                sensor_msgs/LaserScan  [90c7ef2dc6895d81024acba2ac42f369]
                tf2_msgs/TFMessage     [94810edda583a504dfda3829e70d7eec]
topics:      camera_info    953 msgs    : sensor_msgs/CameraInfo
                images          10 msgs    : sensor_msgs/Image
                laserscan     2788 msgs    : sensor_msgs/LaserScan
                tf            2938 msgs    : tf2_msgs/TFMessage
                tf_static        1 msg     : tf2_msgs/TFMessage
```

Figure 1.3: Example of a recorded bag file info

1.1.5 Data Serialization

Despite it's potentiality, bag files are not the best way to store the acquisition data for the reconstruction pipeline. There are some limitations of bag files in this application. The most noticeable is that the full transformation graph is stored, while in fact only the transformations between the start and end frame of the PTU are needed, as well as the transformations between the PTU mount link and each one of the sensors, which are static. Also, this transformation messages are not synchronized with the laser scans and image messages, which means an interpolation has to be performed each time the data is read. Another drawback is that bag files stores messages in it's own format, which hinder reading and inspecting the data, which can be helpful to check if an acquisition was successful. For example, the images are serialized into a ROS message, instead of being in a file with a known format, like *JPG*, which would allow for easier access and inspection.

To solve this issues, a preprocessing of the bag files was performed, to convert and extract all the important information into well known and useful formats. Each laser scan was stored in a *AVRO* file row that contains the timestamp (when it was taken), the minimum and maximum angle (aperture of the laser scan), the minimum and maximum ranges that the laser can capture, the transformation of the ptu and the list of all the measured ranges. An example of such row is show in Figure 1.4, obtained using the *avro cat* command. Each image was stored in a separate *JPEG* file and it's timestamp and transformation was stored in a row, again in a *AVRO* file. The parameters inherent to the acquisition, such as the name of the bag, the extrinsic and intrinsic calibration of the camera used and the extrinsic calibration of the laser was stored in a *YAML* file. The transformations in both the images and laser scans are stored as vector for translation and quaternion for rotation.

```
{
    "ranges" : [ ... ],
    "limits" : {
        "min" : 0.100000001490116,
        "max" : 29
    },
    "timestamp" : 1536174204611117487,
    "angles" : {
        "min" : -2.35619449615479,
        "max" : 2.35619449615479
    },
    "transform" : {
        "rotation" : [ ... ],
        "translation" : [ ... ]
    }
}
```

Figure 1.4: Example of laser scan row

```
bag: acquisition_2018-09-05-20-02-46.bag
camera:
    extrinsic:
    translation: [ ... ]
    rotation: [ ... ]
    intrinsic:
    principal_point: [ ... ]
    height: 1448
    focal_lenghts: [ ... ]
    width: 1928
    distortion_coef: [ ... ]
    distortion_model: plumb_bob
laser:
    extrinsic:
    translation: [ ... ]
    rotation: [ ... ]
    limits:
        max: 29
        min: 0.1
    angles:
        max: 2.356194
        min: -2.356194
```

Figure 1.5: Example of the parameters YAML file

1.2 Capture

As seen before, acquisitions only capture a subset of the scene geometry and color, so multiple acquisitions are required. This problem can be partially solved by recording multiple acquisitions instead of once. Therefore, a capture is a collection of acquisitions of the same scene and it's goal is to contain as much data as possible from the scene, in order to create a proper reconstruction. However, this raises some challenges, on how to plan and execute the multitude of acquisitions and how to merge the data from all of the acquisitions (discussed in [section X]).

Planning determines where should the 3D scanner be places in each acquisition and the sequence of the acquisitions. In this work, this was done prior to any acquisitions, according to a set of rules, to minimize the occlusion, maintain a minimum point density on all surfaces, capture color information of as much surfaces as possible and minimize the processing errors (specially the acquisition registration (see chapter III)). Each one of this problems and its solutions are explained in more detail hereupon.

To begin with, occlusion and range limitations restrict the covered area of an acquisition to a subset of the scene, which is dependent of the position and orientation of the 3D scanner in the scene. For example, in the example room, a 3D scanner placed in two different positions and orientations captures just a limited area of the scene.

Secondly, the point density decreases with the distance of the object to the sensor, which can influence the reconstruction, specially if the objects have smaller details. For example, a wall does not need a big point density, but a smaller object such as a chair or table should have a higher one. Therefore, the position and orientation of the acquisitions should regard this, such that the point density is adequate to the dimensions of the objects.

At last, the acquisition registration requires that between each acquisition there is enough overlap between the point clouds, so a transformation between acquisitions can be determined. So, between each acquisition there should be a maximum distance, such that this registration is possible. Also, this registration requires a good initial estimate for the transformation, otherwise it is not able to find a correct transformation. The solution proposed is to define a sequence of acquisitions such that each subsequent acquisition is near to the previous one and the relative rotation is small.

In conclusion, a good capture planning requires that key acquisitions are made to minimize occlusion and maintain a adequate point density and multiple acquisition have to be made, connecting the key points, and each acquisitions should be close enough to the previous one, such that the registration between acquisitions are possible. In this work, we determined this sequence of acquisitions by determining a path inside the scene. This process, however, can be very subjective and dependent of the user, and the evaluation of the capture is all done afterward, because no feedback exists during the capture.

Chapter 2

Methodology for Geometry Reconstruction

2.1 Point Registration

missing introduction here

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 (2.1). 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.

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \begin{pmatrix} r_i \cos(\theta_i) \\ r_i \sin(\theta_i) \\ 0 \end{pmatrix} \tag{2.1}$$

Further on, each point is registered in the referencial of the acquisition. According to the transformation graph (see Figure 2.1), 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 (2.2).

$$P_{i,j} = \begin{pmatrix} x_{i,j} \\ y_{i,j} \\ z_{i,j} \\ 1 \end{pmatrix} = T_{acq}^{ptu} \cdot T_{ptu}^{laser} \cdot \begin{pmatrix} r_i \cos(\theta_i) \\ r_i \sin(\theta_i) \\ 0 \\ 1 \end{pmatrix} \tag{2.2}$$

At this phase, each point has 2 indexes, one for the laser scan index $j = 1 \ldots L$ and another for the range index $i = 1 \ldots N$, relative to the each laser scan. Despite that point

$$\begin{tikzcd} \mathit{scene} \arrow[r, "T_{scene}^{acq}"] & \mathit{acquisition} \arrow[r, "T_{acq}^{ptu}"] & \mathit{ptu} \arrow[r, "T_{ptu}^{laser}"] & \mathit{laser} \end{tikzcd}$$

Figure 2.1: Transformation graph

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 2.3.

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 the [section XX].

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.

2.2 Laser Extrinsic Calibration

2.2.1 Radlocc Method

2.2.2 Proposed Method

Paramerization

Plane Segmentation

Plane Fitting and RMS error

Loss Function

Optimizer

Overview

2.3 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 2.2. 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 2.1 has two indexes, one for the range index (i) and one for the laser scan j. For each laser scan, each point p_i has a neighborhood $p_{i-k}, \dots, p_{i+k}$, because each subsequent point has an increasing angle to the previous point. Between successive laser scans each point

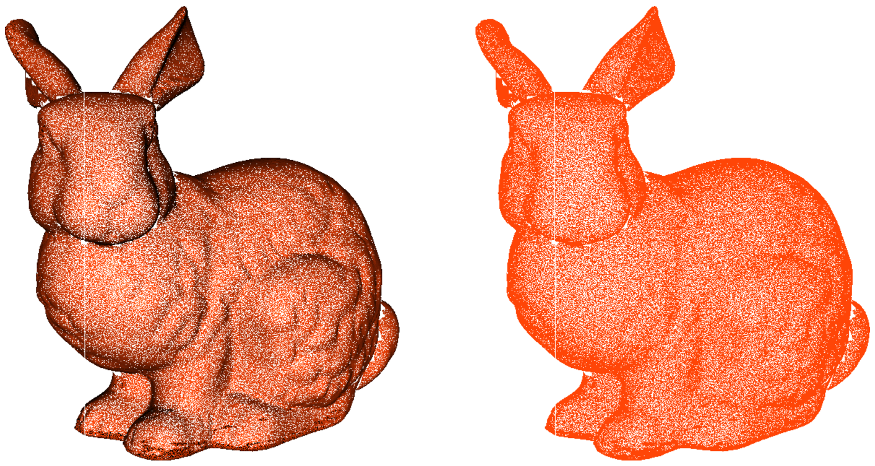


Figure 2.2: Stanford Rabbit Rendering with (left) and without (right) normals

has an increasing angle (the pan angle) to the previous one. Therefore, for this algorithm, the neighborhood of each point is show in Equation (2.3). 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.

$$neighborhood(p_{i,j}, k_1, k_2) = \{p_{i-k_1,j-k_2},\ \ldots,\ p_{i+k_1,j+k_2}\} \tag{2.3}$$

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 (2.4), 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 v_i. Finally, the direction of the normal vector of the point is the eigen vector corresponding to the smallest eigenvalue, so $\hat{n} = \pm \mathrm{v}_3$.

$$\mathcal{C} = \frac{1}{n} \sum_{i}^{n} (p_i - \bar{p}) \otimes (p_i - \bar{p}) \tag{2.4}$$

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 (2.5).

$$\hat{n} \cdot p < 0 \tag{2.5}$$

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.

2.4 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.

2.4.1 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 (2.6), where $T(P)$ is the result of the application of the transformation T to the point cloud P.

$$T = \underset{T}{argmin}(error(T(P), Q)) \tag{2.6}$$

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 (2.7). 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.

$$error(P, Q) = \sum_{(p_i, q_i)} |p_i - q_i| \tag{2.7}$$

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:

$$T_{i+1} = \underset{T}{argmin}(error(T_i(P_i), Q)) \tag{2.8}$$

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

$$T = T_1 \circ T_2 \circ \cdots \circ T_N \tag{2.9}$$

2.4.2 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_{i-1} is found. The final accumulated point cloud is assembled using the Equation (2.10). 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 2.3a.

$$P = \bigcup \left(T^2_1 \circ T^3_2 \circ \cdots \circ T^i_{i-1}\right)(P_i) \tag{2.10}$$

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 2.3b. 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:

$$T_i = ICP(A, P_i, T_0 = T_{i-1}) \tag{2.11}$$

$$A_{i+1} = Ai \bigcup T_i(P_i) \tag{2.12}$$

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 (2.13) and (2.14), for a list of point clouds $S = \{P_1, P_2, \ldots, P_n\}$. At each level l a new list of point clouds ${}^lP$ and transformations ${}^lT$ are computed, as shown in Figure 2.3c.

$${}^lT = \left\{ ICP({}^{l-1}P_1, {}^{l-1}P_2),\ \ldots,\ ICP({}^{l-1}P_{n-1}, {}^{l-1}P_n) \right\} \tag{2.13}$$

$${}^lP = \left\{ {}^{l-1}P_1 \bigcup {}^lT_1({}^{l-1}P_2),\ \ldots,\ {}^{l-1}P_{n-1} \bigcup {}^lT_{n/2}({}^{l-1}P_n) \right\} \tag{2.14}$$

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.

$$\begin{tikzcd}
\boxed{P_i} \arrow[d, "T_i^{i-1}"] \\
\vdots \arrow[d, "T_4^3"] \\
\boxed{P_3} \arrow[d, "T_3^2"] \\
\boxed{P_2} \arrow[d, "T_2^1"] \\
\boxed{P_1}
\end{tikzcd}$$

(a) First method

$$\begin{tikzcd}
\boxed{P_1} \arrow[r, "T_1"] & \boxed{A_1} \arrow[d, "\bigcup P_1"] \\
\boxed{P_2} \arrow[r, "T_2"] & \boxed{A_2} \arrow[d, "\bigcup P_2"] \\
\boxed{P_3} \arrow[r, "T_3"] & \boxed{A_3} \arrow[d, "\bigcup P_3"] \\
 & \vdots \arrow[d, "\bigcup P_{i-1}"] \\
\boxed{P_i} \arrow[r, "T_i"] & \boxed{A_i}
\end{tikzcd}$$

(b) Second method

$$\begin{tikzcd}
\boxed{P_1} \arrow[dr] \arrow[dd, "T_1^2"'] & & \\
 & \boxed{A_{12}} \arrow[ddr] \arrow[dddd, "T_{12}^{34}"'] & \\
\boxed{P_2} \arrow[ur] & & \\
 & & \boxed{A_{1234}} \\
\boxed{P_3} \arrow[dr] \arrow[dd, "T_3^4"'] & & \\
 & \boxed{A_{34}} \arrow[uur] & \\
\boxed{P_4} \arrow[ur] & &
\end{tikzcd}$$

(c) Third method

Figure 2.3: Multiple Point Cloud ICP approaches

2.5 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.

2.5.1 NaN Removal

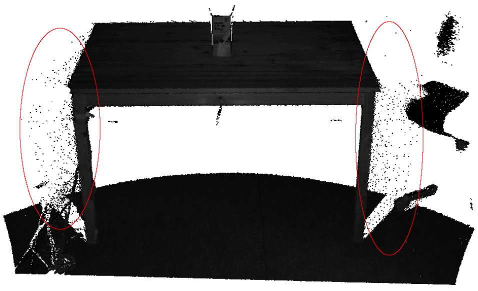
The first filter is the NaN removal. In the acquisition, any range that is not measured is stored as a NaN, to signal that they are missing. During the point registration phase, all this missing ranges remain as nan, and should be removed, because their information is irrelevant and take as much space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

2.5.2 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 2.4.

2.5.3 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


(a) Before SOR

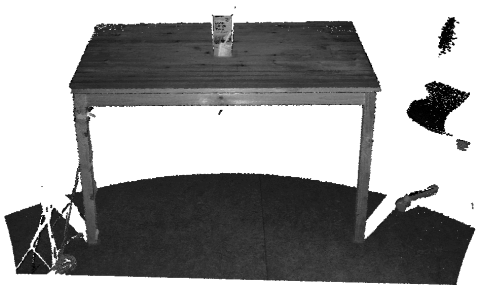
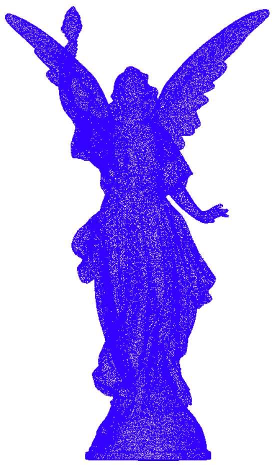
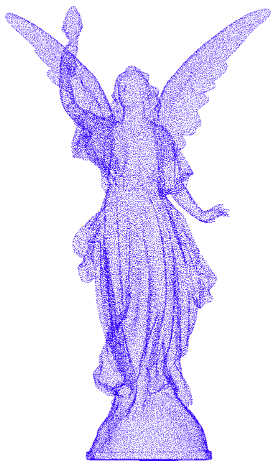

(b) After SOR

Figure 2.4: SOR filter in a point cloud

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 2.5 shows the Lucy dataset after a voxel grid downsampling with different leaf size values: Figure 2.5a with 2 mm (288.000 pts), Figure 2.5b with 5 mm (55.000 pts) and Figure 2.5c with 2 mm (18.000 pts).


(a) Leaf size of 2 mm


(b) Leaf size of 5 mm

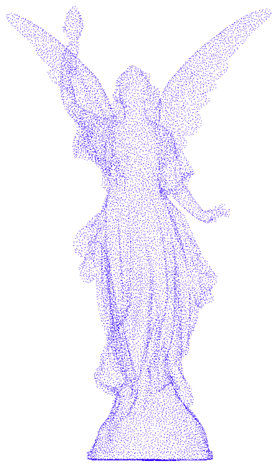

(c) Leaf size of 8 mm

Figure 2.5: "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).