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.

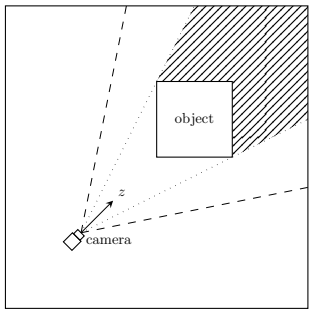
Capture is a collection of acquisitions taken from the same scene, from different poses and times.

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.

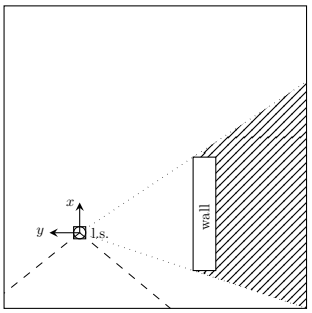
So, acquisitions and captures are different levels and each one has a different method and objectives. In an acquisition level, the focus is on how to operate the scanner and define how the data is recorded. In a capture level, the focus is on how to plan multiple acquisitions so a good reconstruction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

1.1 Acquisition

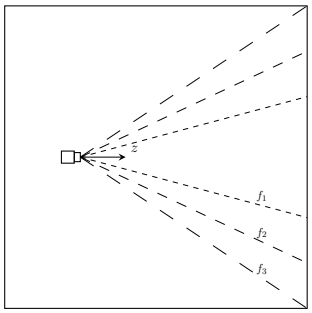
An acquisition is a collection of sensor data (laser scans and images) collected by the sensors in the scanner. Both sensors sample only a small subset of the whole environment: the laser scans only have points from a planar region of the space and cameras are limited by their focal length. To overcome 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.



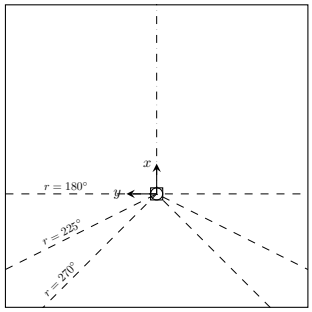
(a) Occlusion in a camera



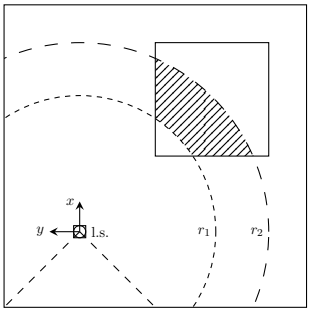
(b) Occlusion in a laser scanner



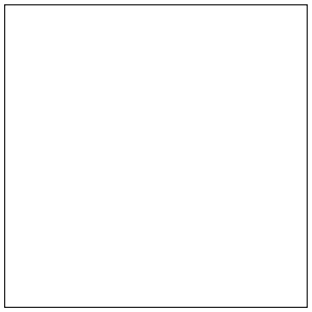
(c) Different focal points of a camera



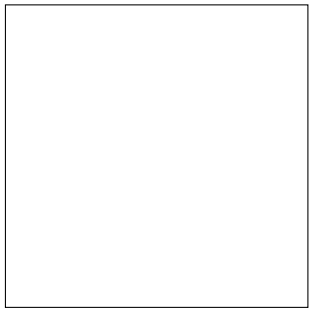
(d) Radial aperture of a laser



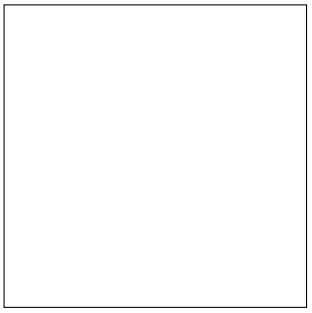
(e) Range limitations



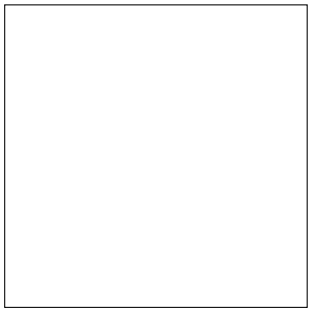
(f) yet



(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.

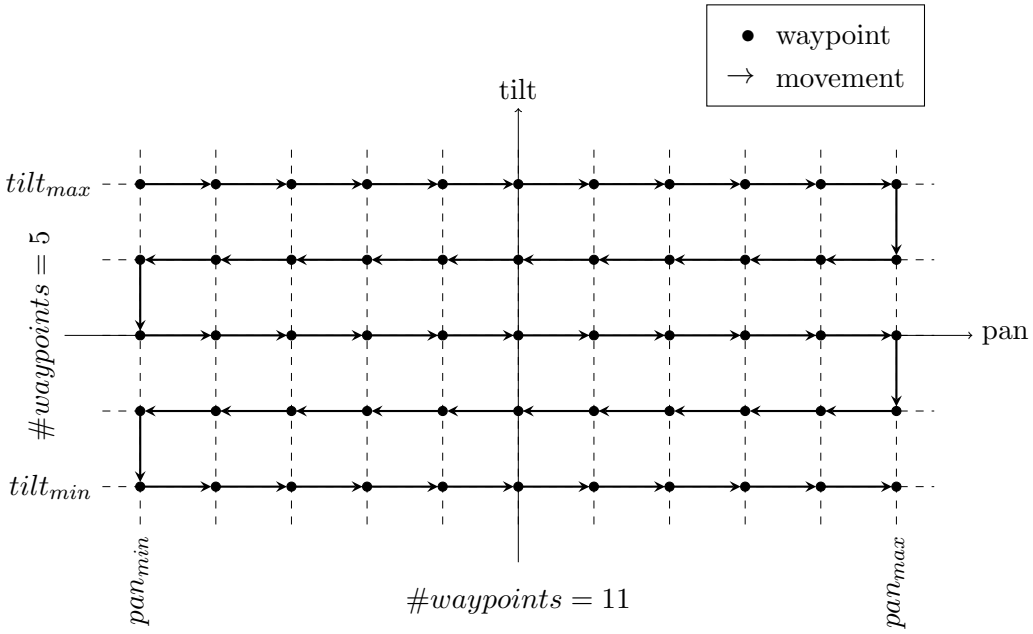


Figure 1.2: Waypoints and movements in the pan/tilt joint space

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 2.4).

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

This chapter present the methodology to reconstruct the geometry of the scene. In Section 2.1, the laser scans of each acquisition are transformed into point clouds. In Section 2.2, two calibration methods, one of which was developed in this work, are described to obtain the extrinsic calibration of the laser scanner. Section 2.3 describes a method to estimate the normals, based on the structure of the point cloud. In Section 2.4, a method to find the transformations between acquisitions is described, to merge the acquisitions into one point cloud. Finally, in Section 2.5, three point cloud filters used in this work are described.

2.1 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 (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).

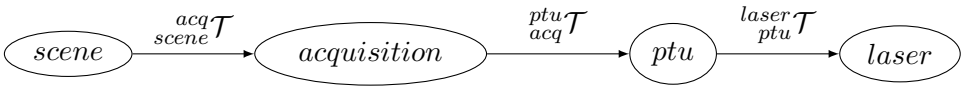


Figure 2.1: Transformation graph

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

At this phase, each point has 2 indexes, one for the laser scan index $j = 1 \dots L$ 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 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 in Section 2.2.

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.

2.2 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 2.2.1) and a new method developed in this work (Section 2.2.2), that aims to achieve better results than the latter.

2.2.1 Radlocc Method

[6] 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 2.2.

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 2.3.

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

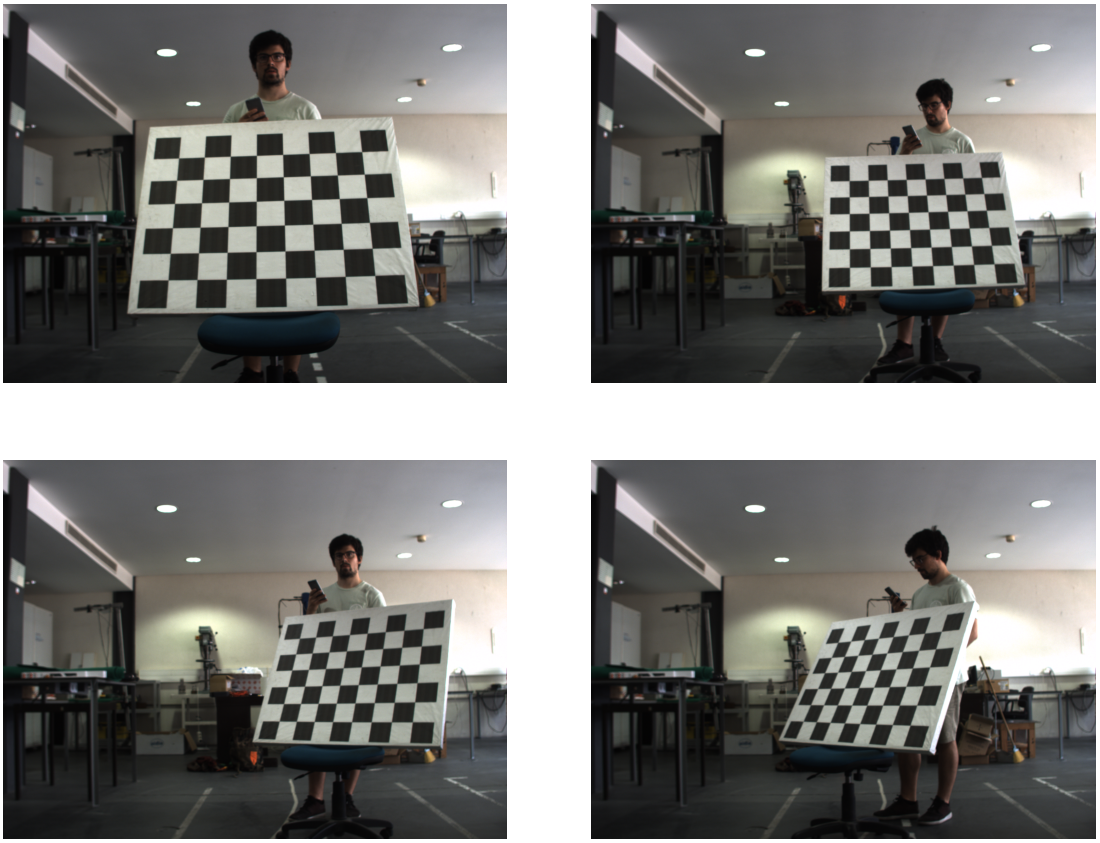


Figure 2.2: Images captured for RADLOCC

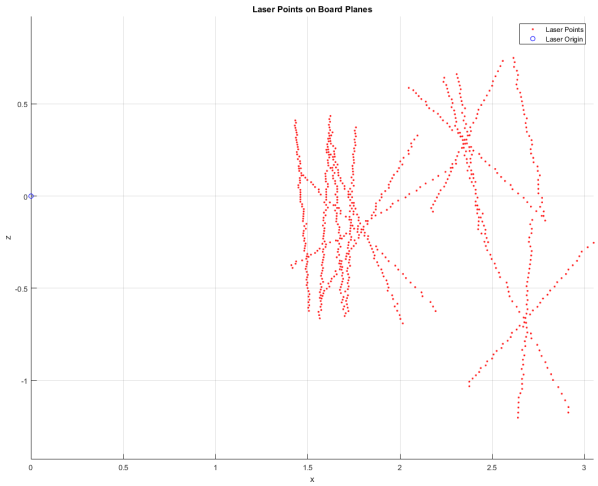


Figure 2.3: Radlocc laser scans chessboard extraction

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 (2.3).

$$\mathcal{T}_{calibration} = {}^{camera}_{PTU}\mathcal{T} \cdot {}^{laser}_{camera}\mathcal{T} \tag{2.3}$$

2.2.2 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 2.2.1) 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 $\mathcal{T}$, corresponds a point cloud $\mathcal{P}$, following the method shown on Section 2.1. 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 $\mathcal{T}$ 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 2.4, 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.

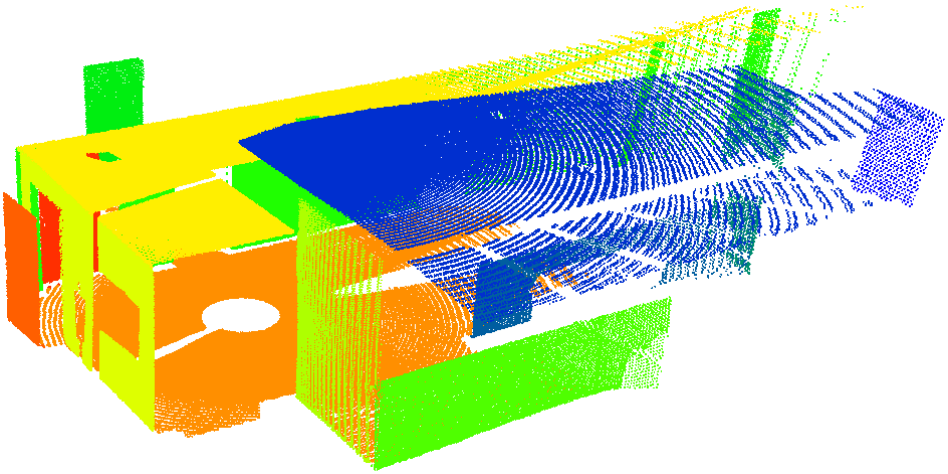


Figure 2.4: Example of a plane segmentation. Each color represents a cluster

To start with, the plane equation for each cluster is found. The same method as in Section 2.3, 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 (2.4)). Then, the covariance matrix $\mathcal{C}$ is calculated (Equation (2.5)).

$$\bar{p} = \sum_i p_i \tag{2.4}$$

$$\mathcal{C} = \sum_i (p_i - \bar{p}) \otimes (p_i - \bar{p}) \tag{2.5}$$

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 (2.6). This value is expected to be minimal when all the partial losses are minimal which, according to this hypothesis, corresponds to a correct calibration.

$$RMS = \sqrt{\sum_i^N x_i^2} \tag{2.6}$$

Paramerization

The parameters in this calibration should define a geometric transformation in space, which is, in the end, a transformation matrix (Equation (2.7)). 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.

$$\mathcal{T} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

However, not all representations are suitable for an optimization. In fact, in [3] 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 [5]. 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 (2.8).

$$\begin{aligned} \theta &= |\omega| \\ a &= \frac{\omega}{|\omega|} \end{aligned} \tag{2.8}$$

Computing the rotation matrix from ω is done using the Rodrigues' formula (Equations (2.9) and (2.10)) [5].

$$R = I + \frac{\sin\theta}{\theta}[\omega] + \frac{1 - \cos\theta}{\theta^2}[w] \tag{2.9}$$

$$[w] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{2.10}$$

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 (2.11).

$$P = \{t_1, t_2, t_3, r_1, r_2, r_3\} \tag{2.11}$$

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 2.5.

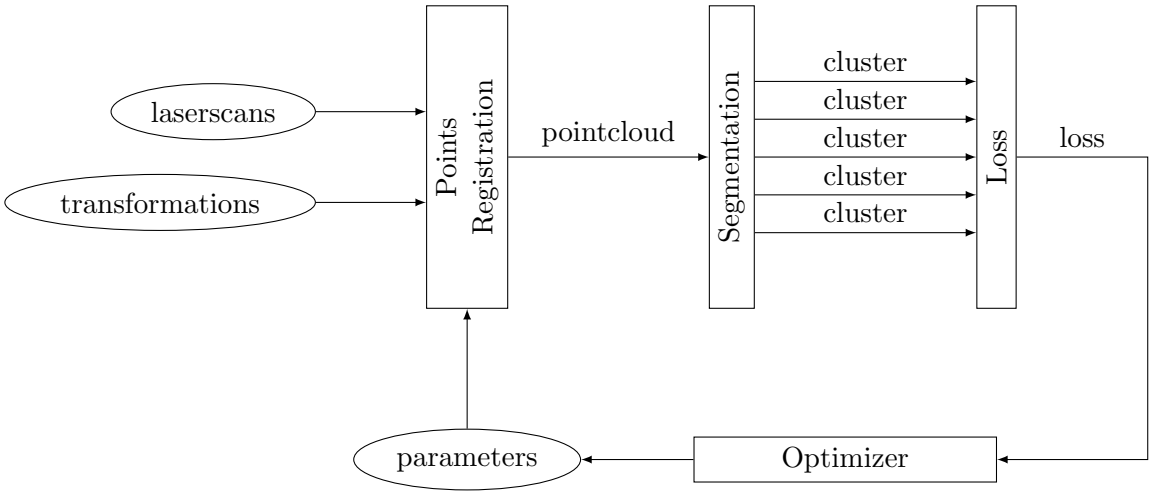


Figure 2.5: Calibration Overview

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 2.6. 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 2.1 has two indexes, one for the range index (i) and one for the laser scan j. For

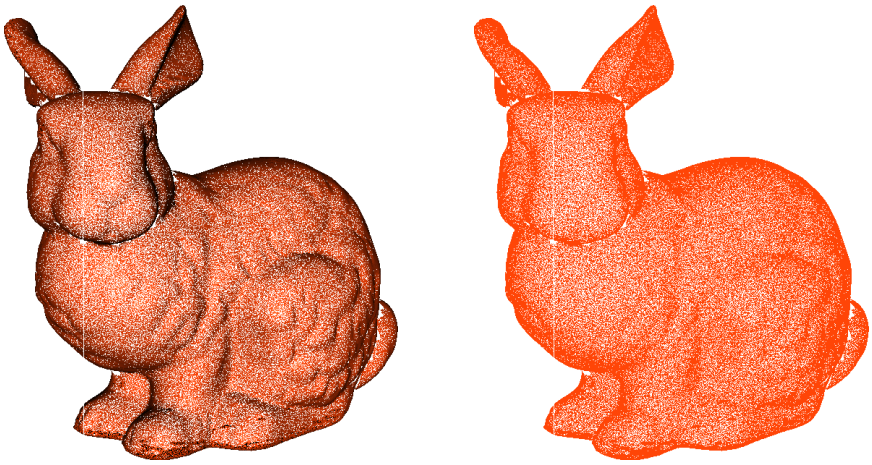


Figure 2.6: Stanford Rabbit Rendering with (left) and without (right) normals

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 (2.12). 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.

$$neighborhood(p_{i,j}, k_1, k_2) = \{p_{i-k_1,j-k_2}, \ \ldots, \ p_{i+k_1,j+k_2}\} \tag{2.12}$$

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 (2.13), 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 $\hat{n} = \pm\mathtt{v}_3$.

$$\mathcal{C} = \frac{1}{n}\sum_{i}^{n}(p_i - \bar{p}) \otimes (p_i - \bar{p}) \tag{2.13}$$

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 (2.14).

$$\hat{n} \cdot p < 0 \tag{2.14}$$

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.

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 $\mathcal{P}$ be the target point cloud and $\mathcal{Q}$ the reference point cloud. Then, the aim of the registration is to estimate the transformation $\mathcal{T}$ from the referencial of $\mathcal{P}$ to $\mathcal{Q}$ by minimizing the error function $\text{error}(\mathcal{P}, \mathcal{Q})$ in Equation (2.15), where $\mathcal{T}(\mathcal{P})$ is the result of the application of the transformation $\mathcal{T}$ to the point cloud $\mathcal{P}$.

$$\mathcal{T} = \underset{\mathcal{T}}{\text{argmin}}(\text{error}(\mathcal{T}(\mathcal{P}), \mathcal{Q})) \tag{2.15}$$

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 (2.16). 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.

$$\text{error}(\mathcal{P}, \mathcal{Q}) = \sum_{(p_i, q_i)} |p_i - q_i| \tag{2.16}$$

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 $\mathcal{T}_i$ at each iteration, yielding a new transformed point cloud $\mathcal{P}_i$. Then, the next transformation is found:

$$\mathcal{T}_{i+1} = \underset{T}{\text{argmin}}(\text{error}(\mathcal{T}_i(\mathcal{P}_i), \mathcal{Q})) \tag{2.17}$$

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

$$\mathcal{T} = \mathcal{T}_1 \circ \mathcal{T}_2 \circ \cdots \circ \mathcal{T}_N \tag{2.18}$$

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 $\mathcal{P}_i$ to the point cloud $\mathcal{P}_{i-1}$ and the transformation $\mathcal{T}_{i-1}^{i}$ is found. The final accumulated point cloud is assembled using the Equation (2.19). 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 2.7a.

$$\mathcal{P} = \bigcup \left(\mathcal{T}_1^2 \circ \mathcal{T}_2^3 \circ \cdots \circ \mathcal{T}_{i-1}^{i}\right)(\mathcal{P}_i) \tag{2.19}$$

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 $\mathcal{P}$ 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 2.7b. 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:

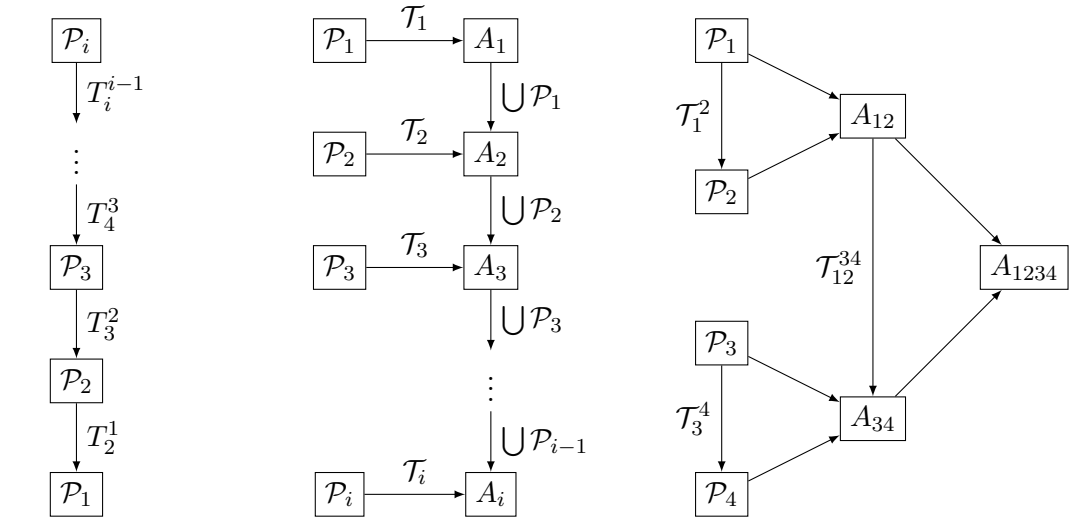
$$\mathcal{T}_i = \text{ICP}(A, \mathcal{P}_i, \mathcal{T}_0 = \mathcal{T}_{i-1}) \tag{2.20}$$

$$A_{i+1} = Ai \bigcup \mathcal{T}_i(\mathcal{P}_i) \tag{2.21}$$

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 (2.22) and (2.23), 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 2.7c.

$${}^{l}\mathcal{T} = \left\{\text{ICP}({}^{l-1}\mathcal{P}_1, {}^{l-1}\mathcal{P}_2), \ \ldots, \ \text{ICP}({}^{l-1}\mathcal{P}_{n-1}, {}^{l-1}\mathcal{P}_n)\right\} \tag{2.22}$$

$${}^{l}\mathcal{P} = \left\{{}^{l-1}\mathcal{P}_1 \bigcup {}^{l}\mathcal{T}_1({}^{l-1}\mathcal{P}_2), \ \ldots, \ {}^{l-1}\mathcal{P}_{n-1} \bigcup {}^{l}\mathcal{T}_{n/2}({}^{l-1}\mathcal{P}_n)\right\} \tag{2.23}$$



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

Figure 2.7: 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.

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.

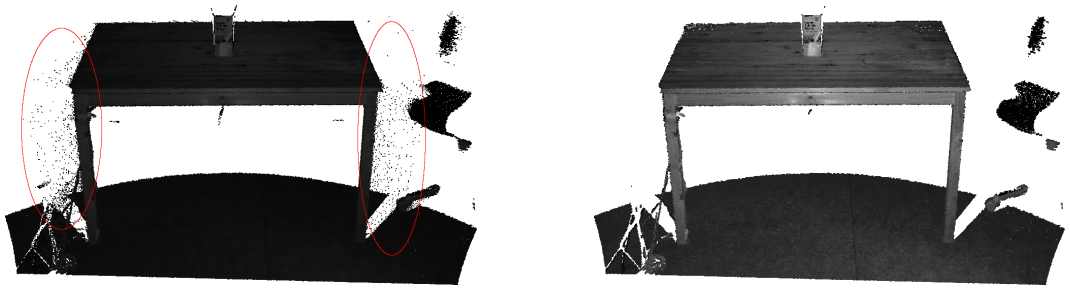
2.5.1 NaN Removal

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 2.8.



(a) Before SOR (b) After SOR

Figure 2.8: SOR filter in a point cloud

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 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 2.9 shows the Lucy dataset after a voxel grid downsampling with different leaf size values: Figure 2.9a with 2 mm (288.000 pts), Figure 2.9b with 5 mm (55.000 pts) and Figure 2.9c 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 2.9: "Lucy" scan after a voxel grid downsampling with different leaf sizes

Chapter 3

Methodology for Image Registration

This chapter describes the methodology for image registration, that is, the process that colorizes (defines the color) the point cloud based on the images taken in the acquisitions. This method can be split into two parts: the Color Registration (Section 3.1), where the process is described per-image and each image colorizes a portion of the point cloud, and the Color Fusion (Section 3.2), where all the colorized point cloud partials are merged into the final colorized point cloud. Also, the pixel registration relies on a camera calibration, both the intrinsic calibration and also the extrinsic, so two methods are shown to obtain this calibration (Sections 3.3 and 3.4).

3.1 Color Registration

This method describes how to colorize a point cloud based on a single image, using a working principle similar to the ray tracing used in computer graphics. As an overview, each point in the point cloud can be transformed as a ray in the camera perspective, which is basically the path from the eye point to the point. This ray can be used to retrieve the original color of the point from the image. However, this process is not so straightforward, because the position and orientation of the camera has to be very precise and occluded points need to be rejected.

3.1.1 Point to pixel coordinates transformation

To start, each point has to be transformed, because the original point is registered in the scene coordinate frame (p_{scene}) and has to be registered into the camera coordinate frame (p_{camera}). So, the transformations ${}^{acquisition}_{scene}\mathcal{T}$, ${}^{ptu}_{acquisition}\mathcal{T}$, ${}^{camera}_{ptu}\mathcal{T}$ can be used according to Equation (3.1). The ${}^{acquisition}_{scene}\mathcal{T}$ transformation is obtained in Section 2.4, ${}^{ptu}_{acquisition}\mathcal{T}$ is the transformation of the PTU and ${}^{camera}_{ptu}\mathcal{T}$ is the extrinsic calibration of the camera and the method to obtain it is in Section 3.4. The transformation graph can be seen in Figure 3.1.

$$p_{scene} = {}^{acquisition}_{scene}\mathcal{T} \cdot {}^{ptu}_{acquisition}\mathcal{T} \cdot {}^{camera}_{ptu}\mathcal{T} \cdot p_{camera} \tag{3.1}$$

Next, each point was transformed into pixel coordinates (u, v), using the pinhole camera model. This model defined how a light ray in projected in the image sensor of a camera and has two parameters: the focal length $f = (f_x, f_y)$ and optical center $(c = (c_x, c_y))$. This

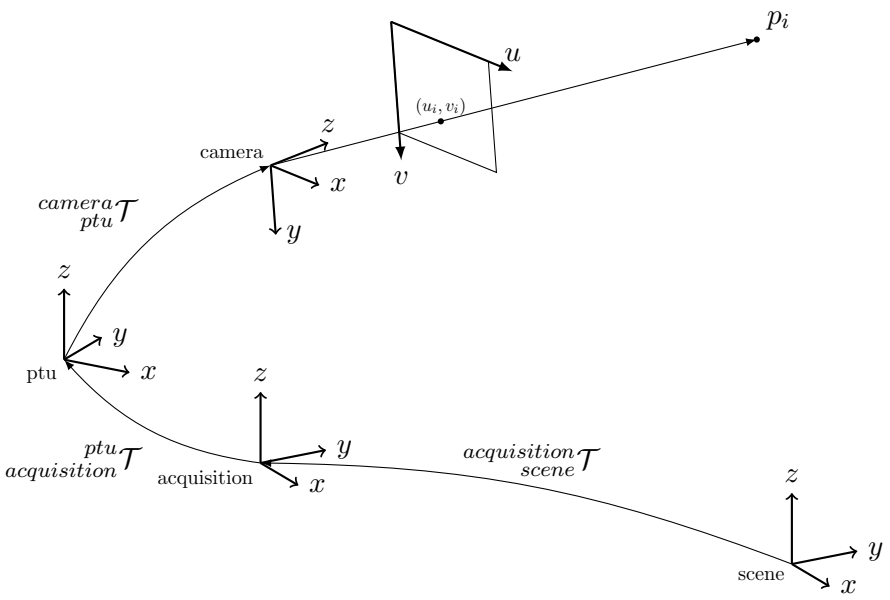


Figure 3.1: Color registration for a single point

parameters are obtained in the intrinsic calibration of the camera (Section 3.3). According to this model, each point is projected as pixel coordinates (u, v) to a plane located a unit distance from the camera eye point, using the perspective projection matrix in Equation (3.2), according to Equation (3.3).

$$\mathcal{P} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$\begin{pmatrix} uz \\ vz \\ z \end{pmatrix} = \mathcal{P} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{3.3}$$

3.1.2 Camera Distortion

The pinhole camera model does not regard the distortion caused by the lens, which is not negligible for most cameras. The two sources of distortion are radial and tangential distortion. Radial distortion makes straight lines appear curved, known as the barrel distortion and pincushion distortion. This distortion is highly noticed in images taken with fish-eye lenses, as seen in Figure 3.2. This distortion can be solved by transforming the (u, v) with Equation (3.4). Similarly, tangential distortion is caused by a misalignment of the lens to the imaging plane, which causes areas in the image to appear closer than expected. This deformation can be solved with the Equation (3.5). In brief, to undistort the image five parameters need to be determined, also known as the distortion coefficients: $\{k_1, k_2, p_1, p_2, k_3\}$, which are obtained in the camera intrinsic calibration method, described in Section 3.3.

$$\begin{pmatrix} u \\ v \end{pmatrix}_{calibrated} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{pmatrix} u \\ v \end{pmatrix} \tag{3.4}$$

$$\begin{pmatrix} u \\ v \end{pmatrix}_{calibrated} = \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} 2p_1 uv + p_2(r^2 + 2u^2) \\ p_2(r^2 + 2v^2) + 2p_2 uv \end{pmatrix} \tag{3.5}$$

Figure 3.2: Barrel distortion in fish eye lens

3.1.3 Point filtering

Not all points are eligible for the color registration, based on it's location and camera properties, so two filtering steps were used: the first filter removes the points outside the view frustum and the second removes the hidden points.

View Frustrum Removal Filter

The view frustum is defined as the region of space that is captured by the camera sensor, which for pinhole cameras is a pyramid truncated by two parallel planes (the near and far clipping planes), as seen in Figure 3.3. The sides of the frustum are limited by the size of the sensor, so the points that lie outside the bounding box defined by the points $(0, 0)$ and $(width, height)$ is excluded.

The near and far clipping planes should reject the points based on the depth of field of the camera. The depth of field, or DOF, is the distance from the camera to the objects range, so that this objects are in focus. If the object falls out of this range, it starts to loose focus incrementally the farther out it is. There is a point where the object is sharper, which is called point of optimal focus. Two factor influences the DOP: the aperture size and the focal length. The aperture influences the amplitude of the DOF, so a bigger aperture results in a narrower DOF, and the focal length defines the point of optimal focus, so a bigger focal

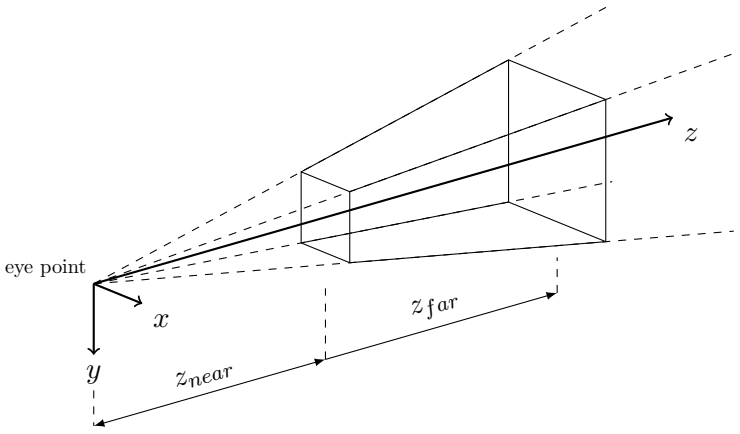


Figure 3.3: Representation of the visual frustum of the camera

length moves the DOF farther away from the camera. The near and far clipping plane should defined to be the boundaries of the DOF, so that all points are in focus.

Based on the frustum of the camera defined above, the points that do not respect Equation (3.6) should be rejected.

$$\begin{aligned} 0 \quad &< u < width \\ \wedge\ 0 \quad &< v < height \\ \wedge\ z_{near} \quad &< z < z_{far} \end{aligned} \tag{3.6}$$

Hidden Point Removal Filter

Not all points that lie on the frustum of the camera are seen by the camera, because some of this points are occluded by nearer objects, so they need to be removed.

In [4], an simple and fast operator, the Hidden Point Remove, or HPR, determines the visibility of point sets, viewed from a given viewport. This method is easily implemented and has a asymptotic complexity of $O(n \log n)$, where n is the number of points in the point cloud. Moreover, this method work well for both sparse and dense point clouds.

The HPR operator operates on a set of points $\mathcal{P} = \{p_i | i = 1 \dots n\}$, and the goal is to determine whether p_i is visible from a viewpoint $\mathcal{C}$. In this application, C is the origin of the point cloud. The algorithm consists of two steps: the inversion and the convex hull construction.

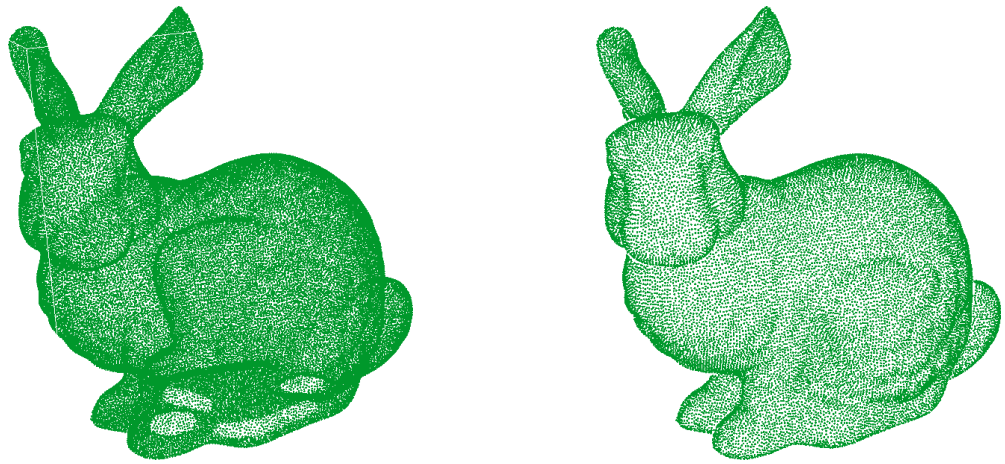
The inversion step maps each point p_i along the ray from $\mathcal{C}$ to p_i, such that $|p_i|$ is monotonically decreasing. There are multiple ways to perform the inversion, but in [4] the *spherical flipping* was used. Spherical flipping reflects a point p_i with respect to a sphere of radius R to the new point $\hat{p_i}$ by applying the Equation (3.7).

$$\hat{p_i} = p_i + 2(R - |p_i|)\frac{p_i}{|p_i|} \tag{3.7}$$

Afterwards, the convex hull of $\hat{\mathcal{P}} \bigcup \{\mathcal{C}\}$, where $\hat{\mathcal{P}}$ is the transformed point set and $\mathcal{C}$ is the center of the sphere, is computed. Finally, the points that lie in the complex hull are the visible points of the point set.

This algorithm only has a parameter, which is the radius R of the sphere used for the spherical flipping, which influences the amount of false positives of the algorithm. In general, R is determined based on the maximum point length $max(|p_i|)$ and a exponential factor α, such that $R = max(|p_i|) \times 10^{\alpha}$. In this application, a factor of $\alpha = 3$ was adequate.

As an example, the HPR operator was used in the Stanford Bunny point cloud, as seen in Figure 3.4 and, as seen, Figure 3.4b only presents the points that are visible, as opposed to Figure 3.4a.



(a) Before the HPR (b) After the HPR

Figure 3.4: Result of the HPR operator in the Bunny point cloud

3.1.4 Color Attribution

Finally, the color is extracted from the image at the pixel coordinates (u, v) and saved for the correspondent pixel. Because images are discrete, the color is interpolated using a bilinear interpolation, which uses the neighbor pixels to interpolate the color C at (u, v) in an image I according to Equation (3.8) (the ceil and floor operators are, respectively, $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$). The interpolation can be visualized in Figure 3.5.

$$\begin{aligned} C(u,v) = &(u - \lceil u \rceil)\,(v - \lceil v \rceil)\; I_{\lfloor u \rfloor, \lfloor v \rfloor} \\ &+ (u - \lceil u \rceil)\,(v - \lfloor v \rfloor)\; I_{\lfloor u \rfloor, \lceil v \rceil} \\ &+ (u - \lfloor u \rfloor)\,(v - \lceil v \rceil)\; I_{\lceil u \rceil, \lfloor v \rfloor} \\ &+ (u - \lfloor u \rfloor)\,(v - \lfloor v \rfloor)\; I_{\lceil u \rceil, \lceil v \rceil} \end{aligned} \tag{3.8}$$

3.2 Color Fusion

In an capture with N_a acquisitions, each one with N_i images, the total number of images account to $N_a \times N_i$. Each one of this images will yield a partial colorized point cloud, according to Section 3.1, and now the point clouds need to be merged into a final point cloud. More specifically, each point p_i has multiple correspondent colors, one for each registered image. The method here described determines the final color in a point-wise fashion and does not account for the neighbor points.

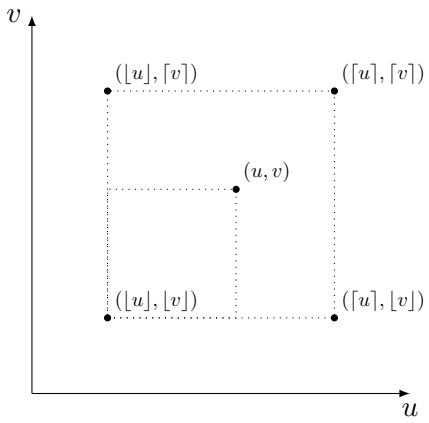


Figure 3.5: Bilinear interpolation in an image

Let us admit that the point p has a set $C = \{c_i | i = 1 \dots k\}$ of k registered colors. The final color of this point c should be a combination of the colors in C. The first and easier approach is to average the colors to obtain the color c, as seen in Equation (3.9). However, this is a poor heuristic as it considers that all colors have the same error, which is not true. For example, an image taken closer to an object is more precise than one taken away from it.

$$c = \frac{1}{k} \sum_{i}^{k} c_i \tag{3.9}$$

A common solution for this mean limitation is to use an weighted mean, shown in Equation (3.10). The w_i are the weights for each color and should reflect the quality of each color, because colors with bigger weight have a bigger influence in the final color.

$$c = \frac{\sum_i^k w_i c_i}{\sum_i^k w_i} \tag{3.10}$$

In this work, the quality measurement was determined based on an heuristic that depends on two factors, that are obtained in the color registration phase (Section 3.1).

The first factor f_1 depends on the distance d from the camera to the point and on the optimal focus point d_f. f_1 is smaller the bigger the distance between d and d_f. The function used was the gaussian centered on d_f. The second factor f_2 depends on the distance from the pixel coordinates (u, v) to the center of the optical center (c_x, c_y). Again, a gaussian distribution was used to calculate f_2, and a bigger distance also yields a smaller f_2. In brief, both factors f_1 and f_2 are calculated according to Equations (3.11) and (3.12). The parameters α and β determine how wide the gaussian function is, so points farther from the peak point influence more or less.

$$f_1 = e^{-\frac{(d-d_f)^2}{2\alpha^2}} \tag{3.11}$$

$$f_2 = e^{-\frac{(u-c_x)^2+(v-c_y)^2}{2\beta^2}} \tag{3.12}$$

$$\tag{3.13}$$

Figure 3.6: Interface for the *cameracalibrator* node

The two factors are then combined into the weight w factor of the color, based on a linear combination, dependent on a parameter s, which determines the influence of each factor, as seen on Equation (3.14).

$$w = sf_1 + (1 - s)f_2 \tag{3.14}$$

In conclusion, for each point p_i the color c_i is attributed, based on the registered colors of each image. The fusion of all this colors is based on a weighted mean, where the weight of each color is determined by an heuristic that considers the location of the color in pixel coordinates and the distance of the point to the camera, in order to benefit points that have a better quality in the measurement, for example, points that are in focus or points that are closer to the camera center. This process is repeated for all the points of the point cloud until every point has a color (however, some points have no color registered, because no color was registered before).

3.3 Camera Intrinsic Calibration

The intrinsic calibration determines the intrinsic parameters of the camera, such as the focal lenght ($f = (f_x, f_y)$), the optical center ($c = (c_x, c_y)$) and the distortion coefficients to model the lens distortion. This parameters are required to create the distortion matrix, as well as the undistortion function, which is essencial for the point projection (as described in **??**).

The calibration procedure used in this work is a standard procedure for cameras with low distortion and is known as the chessboard camera calibration. This method calibrates a monocular camera with fixed focus using a sequence of images taken from a chessboard with known dimensions. In order to improve the calibration results, the chessboard should rotate and move, in order to occupy the entire image size.

After all the images are obtained, the corners of the chessboard are extracted and the re-projection error is minimized to obtain the the intrinsic parameters. The results of this calibration are more accurate if the corners of the chessboard are well defined in the image, so the chessboard should have an appropriate size. Also, the chessboard poses should be enough and should be well distributed spatially.

In the end, the accuracy of the calibration should be measured for new images, with the re-projection error. This value should be as low as possible and, as a rule of thumb, a value less than 0.01 is acceptable.

In ROS, this calibration is easily obtained with the *cameracalibrator.py*, which includes a graphical interface, and provides feedback about the corner detection and the state of the calibration. The interface is shown on Figure 3.6. In this system, this data is first saved into a ROS *camera_info* file. Then, this file is also saved in each capture in the parameters file.

3.4 Camera Extrinsic Calibration

The extrinsic parameters of the camera the position and orientation of the camera in the robot. In this case, the camera was mounted statically on top of the PTU, just like the

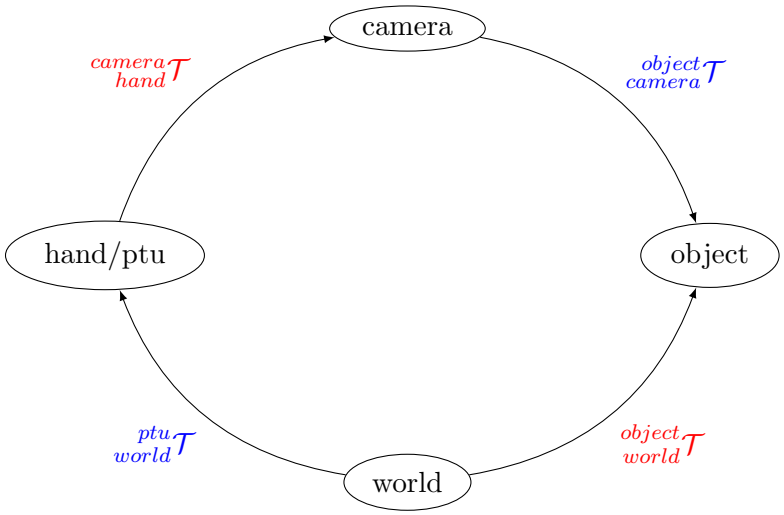


Figure 3.7: Hand-in-eye transformation graph

laser scanner. This calibration that determines this extrinsic parameters is known as the eye-in-hand calibration, described in [2].

This calibration relies on a static calibration object, whose pose can be estimated in the camera frame. Hence, four coordinate frames and four transformation exist. The four frames are the *camera* frame, the *world* frame, the *PTU* frame and the *object*. The four transformations are the extrinsic transformation of the camera, or the *PTU* to the *camera* transformation ${}^{camera}_{ptu}\mathcal{T}$, which is static and unknown, the *camera* to *object* transformation ${}^{object}_{camera}\mathcal{T}$, which is obtained by the object pose estimation algorithm, the *world* to *PTU*, which is known and, finally, the *world* to *object* transformation, which is static and unknown. The overall transformation graph is shown is in Figure 3.7, with the unknown transformations in red and the known transformations in green.

The inspection of the transformation graph determines an equality, because there are two possible ways to transverse the graph from one node to another, which yields the Equation (3.15). This equality is the base of this optimization: ${}^{camera}_{ptu}\mathcal{T}$ can be obtained from multiple pairs of synchronized ${}^{object}_{world}\mathcal{T}$ and ${}^{object}_{camera}\mathcal{T}$ transformations.

$$ {}^{object}_{world}\mathcal{T} = {}^{ptu}_{world}\mathcal{T} \cdot {}^{camera}_{ptu}\mathcal{T} \cdot {}^{object}_{camera}\mathcal{T} \tag{3.15} $$

In this work, the object used for detection was an ArUco marker, which is comprised of a pattern which can be detected and also allows for precise pose estimation. One of the biggest advantages over other markers is that the implementation for detection and pose estimation is already implemented in the ROS package *aruco_detect*. The calibration is also implemented in the ROS package *visp_hand2eye_calibration*, as a node that receives multiple transformations in the topics */world_effector* and */camera_object*, which correspond respectively to the ${}^{ptu}_{world}\mathcal{T}$ and ${}^{object}_{camera}\mathcal{T}$ transformations. To publish the transformations on this topics, a node was developed, the *hand2eye_simple_client*, which publishes both the transformations synchronously at the keypress of the user. The control of the PTU was also manual.

Bibliography

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

[2] Radu Horaud and Fadi Dornaika. "Hand-Eye Calibration". In: 14 (June 1995), pp. 195–210.

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

[4] Sagi Katz, Ayellet Tal, and Ronen Basri. "Direct visibility of point sets". In: 26 (July 2007).

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

[6] 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`.