

# Using Extended Measurements and Scene Merging for Efficient and Robust Point Cloud Registration

Jacopo Serafin<sup>a,\*</sup>, Giorgio Grisetti<sup>a</sup>

<sup>a</sup>*Department of Computer, Control, and Management Engineering “Antonio Ruberti” at Sapienza University of Rome. Via Ariosto 25, 100185 Rome, Italy.*

---

## Abstract

Point cloud registration is a fundamental building block of many robotic applications. In this paper we describe a system to solve the registration problem, that builds on top of our previous work [1], and that represents an extension to the well known Iterative Closest Point (ICP) algorithm. Our approach combines recent achievements on optimization by using an extended point representation [2] that captures the surface characteristics around the points. Thanks to an effective strategy to search for correspondences, our method can operate on-line and cope with measurements gathered with an heterogeneous set of range and depth sensors. By using an efficient map-merging procedure our approach can quickly update the tracked scene and handle dynamic aspects. We also introduce an approximated variant of our method that runs at twice the speed of our full implementation. Experiments performed on a large publicly available benchmarking dataset show that our approach performs better with respect to other state-of-the art methods. In most of the tests considered, our algorithm has been able to obtain a translational and rotational relative error of respectively  $\sim 1$  cm and  $\sim 1$  degree.

*Keywords:* Pose Tracking, Point Cloud Registration, Iterative Closest Point (ICP)

---

## 1. Introduction

Autonomous mobile robots are used in a broad range of applications, including logistics, mining, delivery tasks in health facilities, search and rescue and many others. These applications rely on the integration of complex subsystems that address specific tasks, such as localization, mapping, object recognition,

---

\*Principal corresponding author

*Email addresses:* [serafin@dis.uniroma1.it](mailto:serafin@dis.uniroma1.it) (Jacopo Serafin),  
[grisetti@dis.uniroma1.it](mailto:grisetti@dis.uniroma1.it) (Giorgio Grisetti)  
*URL:* <http://www.dis.uniroma1.it/~serafin> (Jacopo Serafin),  
<http://www.dis.uniroma1.it/~grisetti> (Giorgio Grisetti)

path planning and so on. The 3D perception layers for a robot, in turn, leverage on multiple approaches to gather a consistent representation of the robot surroundings. Registering point clouds is regarded as an essential enabling technology for these subsystems.

Aligning two point clouds means finding the rotation and the translation that maximizes the consistent overlap between two clouds. To this end, the well known Iterative Closest Point (ICP) principle proposed by Besl and McKay [3] is among the most common techniques. ICP iteratively refines an initial guess of a transformation by interleaving a search for correspondences, and the computation of the transform that minimizes the distance between corresponding points.

Since the introduction of ICP, many variants of increased robustness and accuracy have been proposed [1][4][5][6][7]. A common intuition behind these state of the art algorithms is the idea of registering surface, and not points. At the base of their idea there is the observation that point clouds are sampled representation of continuous surfaces, and when seeking for the transformation one has to minimize the distance between corresponding surface elements rather than between corresponding points. Notably, Chen and Medioni [8] were among the first to introduce a point-to-plane metric that exploits the continuity of the surface. Using a surface based error function leads to a more regular evolution of the optimization. The clouds “slide” onto each other along the tangent direction of the underlying surface, instead of being subject to constraints among samples that are unlikely to represent the same point in the world.

In accordance with this intuition, we recently proposed Normal Iterative Closest Point (NICP) [1], a variant of ICP that can run on-line on a multi-core CPU. The main novelty of NICP is to use a 6D error metric that takes into account the distance between corresponding points *and* corresponding surface normals, and uses the surface curvature as additional cue for data association. NICP leverages on our previous analysis on error functions based on surface properties [2]. In that work we provided experimental evidence that considering the geometric structure around the points enlarges the basin of convergence. NICP is reported to compare favorably to other state-of-the-art methods on a large publicly available benchmarking dataset.

A common application of point cloud registration methods is tracking the position of a moving sensor while constructing local models of the environment. Performing pairwise registration between consecutive measurements leads to an unavoidable drift in the estimate due to error accumulation. To reduce this effect, it is common to register the current frame against a reference model that is augmented each time a successful registration is performed. Whereas this reduces the drift, a naive implementation results in a linear growth of the number of points in the model that renders the execution slower over time. Common strategies to deal with the increasing number of points consist in aggregating the measurements through dense or sparse voxel representations. These approaches, however, do not easily cope with dynamic aspects since removing dynamic parts of the scene requires expensive ray-casting operations.

In this paper, we give a more detailed description of NICP, including the

mathematical derivations that were omitted in our previous works. We aim at providing enough information to the interested readers that want to develop their own implementation of the method. Additionally, we discuss a set of algorithmic enhancements to the original system that reduce the computation time by 50% compared to the implementation in [1]. Being more efficient, this system can be used with multiple sensors [9] or run on low-end computers. Furthermore, we propose a projection-based cloud merging approach that allows to limit the number of points in the scene as new clouds are added and naturally deals with moderate dynamics in the scene. Our merging method is straightforward to implement and directly operates on unorganized point clouds. We validated our approaches on a large standard benchmarking dataset and on real world scenarios by performing both quantitative and qualitative experiments. An implementation of our approach is publicly available on the web at <http://goo.gl/W3qXbE>.

The remainder of this paper is organized as follows: in Sec. 2 we provide an overview of the current state-of-the-art methods for point cloud registration; in Sec. 3 we introduce the general ICP formulation; in Sec. 4 we describe the original NICP algorithm in detail; Sec. 5 describes a speeded up variant of our algorithm; Sec 6 shows comparative experiments on a wide real benchmarking dataset; and, finally, Sec. 7 concludes the paper.

## 2. Related Work

The point-cloud registration problem is object of investigation since more than 20 years, and a wide range of solutions have been proposed. The available methods can be categorized in two main groups: sparse approaches that rely on few meaningful points in the scene, and dense methods that directly operate on the entire set of points. Sparse approaches compute data association based on the local structure of the points (features). It is possible to use these methods also when no prior information about the relative pose between the clouds is available, at the cost of a more complex system. On the other side, dense approaches align two clouds by considering every point, and using simple heuristics to perform the data association. Dense methods are usually faster and easier to implement than sparse approaches and therefore, preferred for tracking purposes. As a drawback, however, they are more sensitive to wrong initial guesses. Our approach belongs to the class of dense algorithms.

The earliest point cloud registration methods were designed to operate with 3D laser range finders. These sensors provide highly accurate data, large field of views and high range at a relatively low frame rate. Performances of a few seconds in registering point clouds were more than acceptable since few scans could cover large regions of the environment.

The Iterative Closest Point (ICP) algorithm [3] is one of the first and most used techniques for registering point clouds. It is an iterative algorithm that refines an initial estimate of the relative transformation. At each iteration, starting from the current transform estimate, the algorithm tries to find pairs of points that most likely represent the same patch of surface in the two clouds.

Once the correspondences are computed, a new and improved transformation is calculated for example through the Horn [10] formula that minimizes the Euclidean distance between corresponding points. Most heuristics rely on the current estimate to perform data association, thus ICP and its variants require multiple iterations to converge. Note that the optimization requires linear time in the number of correspondences, this makes the data association heuristic one the main bottleneck of the entire computation. After its introduction, ICP has been improved in many ways.

As stated in the previous section, the main drawback of the original formulation of ICP is the assumption that the points in the two surfaces are exactly the same. This is not true since the point clouds are obtained by sampling a set of points from the surface observed by the sensor. If the sensor view point changes, the probability that two points in the clouds are the same is very low. This is particularly evident at low sampling resolutions. To cope with this wrong assumption, Chen *et al.* [8] replaced the Euclidean distance error metric used by the original ICP with a more robust point-to-plane criterion. This models the fact that the points measured by the sensors are sampled from a locally continuous and smooth surface.

Similarly, Segal *et al.* [4] developed a probabilistic version of ICP called Generalized-ICP (GICP). GICP takes in account the model of the sensor noise, and utilizes the local continuity of the surface sampled through the cloud. This algorithm is a plane-to-plane variant of ICP that exploits the surface normals to weight each matching correspondence in the objective function. The core idea is to consider the shape of the surface surrounding the point by approximating it with a planar patch. In the optimization, two corresponding patches are aligned onto each other by neglecting the error along their tangent, and penalizing the normal direction. This can be easily implemented, during the minimization step, by forcing the covariance matrix of a measurement to have the shape of a disk aligned with the sampled surface.

Magnusson *et al.* [5] approximated the local structure with a set of Gaussians capturing the statistics of the surface in the neighborhood of a point. This representation is commonly known as the Normal Distribution Transform (NDT). NDT can be seen as a registration algorithm operating between spatial point distributions instead of individual points. Thanks to the more realistic objective function, both NDT and GICP exhibit a substantially more stable convergence behavior. The error metric minimized is still a weighted distance between 3D points. In contrast to NDT and GICP, our method utilizes an extended measurement vector composed by both the point coordinates and its surface normal. This means that our error metric measures a distance in a 6D space. Our method uses an NDT-like representation of the scene, but the statistics are computed directly on an image projection instead of a voxel-grid or a KD-tree.

Within ICP and its variants, the optimization and the correspondence search steps can not be considered independent. In fact, if the optimization is robust to outliers and exhibits a smooth behavior, the chances that it finds a better solution at the subsequent step increases. This allows to obtain an improvement

at each iteration until a good solution is found. Despite NDT and GICP, the authors are unaware of other methods that improve the objective function. Since point clouds are the result of sub-sampling a surface, the local characteristics of this surface play a fundamental role in the optimization. In other words, the objective function has to express some distance between surface samples, and the optimization algorithm has to determine the best alignment between these two set of samples. A surface sample, however, is not fully described just by 3D points, but it should require additional cues like the curvature, the surface normal and, potentially, the direction of the edge. Both NDT and GICP minimize a distance between corresponding points, while they forget additional cues that play an important role in determining the transformation and in rejecting outliers.

Steinbruecker *et al.* [11] developed Dense Visual Odometry (DVO), this method takes advantage of the additional light intensity channel available on modern RGB-D sensors. The idea behind this approach is to compute an image containing the neighborhood of the edges extracted from the corresponding RGB image. Thanks to the depth channel, these edges also correspond to 3D points, and thus they can be straightforwardly reprojected in the image plane. The transformation is found by minimizing the photometric distance of the regions around the edges on the image plane. The objective function clearly minimizes a set of 2D distances, and this further reduces the observability of the transform resulting in a narrow basin of convergence. Since DVO has to process a reduced amount of data, it partly copes with this issue by running at very high frame rates. In fact, the capability to process data at a frame rate of 30 Hz, or greater, makes the assumption of a good initial guess true in most cases. Unfortunately, DVO suffers from the noise and the blur affecting moving RGB cameras, and it is sensitive to illumination conditions. Actually, this is more a limitation of the sensor itself than of the algorithm, however these issues make the approach inadequate to operate in scenarios characterized by poor illumination or robots moving fast. The main advantage coming from the use of the RGB channel is that, in case of poor structure in the 3D data (e.g. when looking at a flat wall), the algorithm is still able to track the camera pose if some texture is present.

Newcomb *et al.* proposed Kinect Fusion (KinFu) [6], which is considered one of the major breakthroughs in dense depth mapping. It is a more complex system that combines both components of mapping and point cloud registration. KinFu utilizes the brute force of the GPU to effectively update the environment representation, thus it can run in real-time despite the computationally heavy environment representation. The camera tracking is a point-to-plane variant of ICP that uses image reprojection to determine the correspondences. This camera tracker, however, easily fails if the sensor is moved too fast resulting in ICP to get lost. As in the case of GICP, our approach shows an increased robustness to these kind of situations.

Point cloud registration is a base building block for many robotic applications, in particular it is widely used in localization and mapping systems. Simultaneous Localization and Mapping (SLAM) is the problem for which a mobile robot can build a map of the surrounding environment, while at the

same time localize itself in that map. In SLAM, maps can be classified as landmark and dense maps. The choice of a particular map representation depends on the type of sensors used, on the characteristics of the environment, and on the estimation algorithm used. Landmark maps [12][13][14] are usually preferred in environments where locally distinguishable features can be identified, and especially when RGB cameras are used. On the other hand, dense representations [15][16][17][18] are commonly used in conjunction with range sensors like lasers or depth cameras.

The acronym SLAM and its first definition was introduced by Durrant-Whyte *et al.* in [19], however the first probabilistic SLAM problem born in a series of papers by Cheeseman and Smith [12][20]. These works demonstrated that landmark estimates are all correlated with each other by the common error of the estimated pose of the vehicle [21].

SLAM algorithms can be classified accordingly to the estimation technique used: filtering or smoothing. Filtering approaches rely on recursive Bayesian estimation, also known as Bayes filtering. Dynamic Bayesian Networks (DBN) are a natural representation to describe filtering processes since they highlights the underlying temporal structure of the problem. The big issue with the general Bayes filter is that in most of the cases it is not computationally tractable. A solution to this problem was given by the introduction of a family of recursive state estimators called Gaussian filters. The most famous and used Gaussian filter approaches developed to solve the SLAM problem are those based on Kalman (KF) and Extended Kalman filters (EKF) [12][22], thin junction trees [23], Information (IF) and Extended Information filters (EIF) [24][25], and particle filters [13][14][16][17][26][27].

Conversely from filtering methods, smoothing approaches estimate the full trajectory of the robot from the full set of measurements [28][29][30]. These approaches address the so called full SLAM problem, and they typically rely on least-squares error minimization techniques. These methods use an alternative representation to DBN known as pose graph. Such representation highlights the underlying spatial structure. More specifically, in graph-based SLAM the poses of the robot are represented by nodes (or vertices), and spatial constraints between poses that result from the robot observations or from odometry measurements are encoded as edges (or factors) between the nodes. Once the graph is constructed the goal is to find the configuration of the robot poses that best satisfies the constraints given by the edges. An important and challenging aspect of this representation is the computation of the data association, known also as loop closure detection. The loop closure problem aims at finding past robot poses (far in time) that are near to its current position (close in space). Identifying when a robot has returned back to a previous location allows to reduce drastically the error on the map, and it increases the robustness of the entire algorithm. The main drawback of the data association is that, even only one incorrect correspondence can lead the algorithm to terrible mapping results. During a loop closure two kinds of errors could occur: false positive matching and missed detected matching. The first one happens when the robot closes a loop with a wrong match, the second occurs when a loop closure is missed

because of a failure in the alignment between the previous mapped area and the current one.

The graph-based SLAM problem is usually decoupled in two different steps: the composition of the graph from the raw measurements (graph generation), and the computation of the most likely configuration of the poses given the edges (graph optimization). These two steps are called respectively front-end and back-end. The former is heavily sensor dependent, while the latter relies on an abstract representation of the data which is sensor agnostic. Smoothing methods represent the current state-of-the-art for solving SLAM problems, and point cloud registration is commonly used as part of front-end systems

During the past years many approaches relying on network-based SLAM have been proposed. The concept of graph SLAM born in the seminal paper of Lu and Milios [28]. Gutmann and Konolige in [31] developed an efficient method for constructing the graph and for detecting loop closures while running an incremental estimation algorithm. The ATLAS framework [32] creates a two-level hierarchy of graphs and employs a KF to build the bottom level. A global optimization approach is then used to align local maps at the second level. Like ATLAS, Estrada *et al.* [33] proposed Hierarchical SLAM as a way for using independent local maps. Olson [34] presented a front-end with outlier rejection based on spectral clustering. For performing data association, statistical approaches such as the joint compatibility test [35], or the  $\chi^2$  error test are often applied. Nüchter *et al.* in [36] construct an integrated SLAM system for 3D mapping. The main goal is to improve the construction of the network, while for the optimization they used a 3D variant of the approach of Lu and Milios [28]. Dellaert *et al.* [30] and Ranganathan *et al.* [37] developed an approach known as Square root smoothing And Mapping (SAM). With respect to EKF methods, it handles non-linearities better and it is faster to compute. Moreover, SAM gives an exactly sparse factorization of the information matrix and it can be used to incrementally acquire 2D and 3D maps. Successively, Kaess *et al.* [38] proposed iSAM, an on-line mapping system. More recently other full graph-based SLAM systems have been proposed like Kintinuous [39], DynamicFusion [40], RTAB-SLAM [41] and ORB-SLAM [42]. Whelan *et al.* [43] presented ElasticFusion, where they use surfels in the graph instead of modelling poses.

### 3. ICP and Point Representations

Registering two point clouds consists in finding an isometry that maximizes the overlap between the two clouds. More formally, let  $\mathcal{P}^r = \{\mathbf{p}_{1:N^r}^r\}$  and  $\mathcal{P}^c = \{\mathbf{p}_{1:N^c}^c\}$  be the two set of points, we want to find the transformation  $\mathbf{T}^*$  that minimizes the distance between corresponding points in the two scenes:

$$\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_c \overbrace{(\mathbf{T} \oplus \mathbf{p}_j^r - \mathbf{p}_i^c)^T \boldsymbol{\Omega}_{ij} (\mathbf{T} \oplus \mathbf{p}_j^r - \mathbf{p}_i^c)}^{\chi_{ij}^2} \underbrace{\phantom{(\mathbf{T} \oplus \mathbf{p}_j^r - \mathbf{p}_i^c)^T \boldsymbol{\Omega}_{ij} (\mathbf{T} \oplus \mathbf{p}_j^r - \mathbf{p}_i^c)}}_{\mathbf{e}_{ij}(\mathbf{T})}. \quad (1)$$

In Eq. 1 the symbols have the following meaning:

- $\mathbf{T}$  is the current estimate of the transformation that maps  $\mathcal{P}^r$  in the reference frame of  $\mathcal{P}^c$ ;
- $\mathbf{\Omega}_{ij}$  is an information matrix that takes into account the noise statistics of sensor and/or surface;
- $\mathcal{C} = \{\langle i, j \rangle_{1:M}\}$ , is a set of correspondences between points in the two clouds. If  $\langle i, j \rangle \in \mathcal{C}$ , means that the point  $\mathbf{p}_j^r$  in the reference cloud  $\mathcal{P}^r$  corresponds to the point  $\mathbf{p}_i^c$  in the current cloud  $\mathcal{P}^c$ ;
- $\oplus$  is the standard composition operator (Smith *et al.* [12]) that applies the transformation  $\mathbf{T}$  to the point  $\mathbf{p}$ . If we use the homogeneous notation for transformations and points,  $\oplus$  reduces to the matrix-vector product.

Clearly the exact correspondences are not known. However, assuming to have a good approximation for the initial guess of the relative transform, it is common to compute an approximation of these correspondences through some heuristic (i.e. nearest neighbor). In its most general formulation, ICP iteratively refines the current estimate  $\mathbf{T}$  by interleaving the search for correspondences (data association) and the solution of Eq. 1 (optimization). Appendix A contains the mathematical derivation of the least-squares problem used to solve Eq. 1. At each new iteration, the data association is recomputed after applying to the cloud  $\mathcal{P}^r$  the most recent transformation  $\mathbf{T}$ .

Over time ICP has been modified and extended to a large number of variants of increased robustness and performance. These differ by the choice of the information matrix  $\mathbf{\Omega}_{ij}$ , or by the heuristic chosen to find the correspondences. Plain ICP uses a diagonal  $\mathbf{\Omega}_{ij}$  potentially scaled with a weight capturing the likelihood that a correspondence is correct. In its most general formulation, GICP models  $\mathbf{\Omega}_{ij}$  in order to incorporate the surface information from both clouds

$$\mathbf{\Omega}_{ij} = (\mathbf{\Sigma}_i^s + \mathbf{R}\mathbf{\Sigma}_j^s\mathbf{R}^T)^{-1} \quad (2)$$

where

$$\mu_n^s = \frac{1}{|\mathcal{V}_n|} \sum_{\mathbf{p}_k \in \mathcal{V}_n} \mathbf{p}_k \quad \mathbf{\Sigma}_n^s = \frac{1}{|\mathcal{V}_n|} \sum_{\mathbf{p}_k \in \mathcal{V}_n} (\mathbf{p}_k - \mu_n)^T (\mathbf{p}_k - \mu_n). \quad (3)$$

with  $\mathcal{V}_n$  being the set of points in the neighborhood of  $\mathbf{p}_n$ . Such a formulation, commonly known as plane-to-plane, increases the overall symmetry of the model. The covariances  $\mathbf{\Sigma}_i^s$  and  $\mathbf{\Sigma}_j^s$  are forced to have a disk shape and to lie on the surface from where  $\mathbf{p}_i^c$  and  $\mathbf{p}_j^r$  were sampled ( $\mathbf{\Sigma}_i^s$  and  $\mathbf{\Sigma}_j^s$  with a small eigenvalue along the normal direction). When carrying on the minimization in Eq. 1, GICP replaces the points with their means  $\mu_i^s$  and  $\mu_j^s$ , and the information matrix  $\mathbf{\Omega}_{ij}$  with the result obtained by applying Eq. 2.

Note that this formulation of GICP requires at each iteration of the algorithm a double matrix multiplication, and a matrix inversion, for each correspondence. This yield a significant increase in computation time, in particular when the number of correspondences is in the order of tens of thousands, like in the case



of dense point clouds. As stated by Segal *et al.* in [4], when the information matrix of the reference surface  $\Sigma_j^s$  is neglected, that means  $\Omega_{ij} = (\Sigma_i^s)^{-1}$ , we obtain a limiting case of GICP known as point-to-plane. Albeit slightly less accurate, this special case of GICP saves computation time since  $\Omega_{ij}$  stays fixed during the entire alignment. This makes the approach particularly suitable when dealing with high frequency dense stereo sensors like the Asus Xtion, or the Microsoft Kinect. Our method is a variant of a point-to-plane GICP, that extends the measurements of the error function  $\mathbf{e}_{ij}$  with the surface normal components.

In all cases here described the error vector  $\mathbf{T} \oplus \mathbf{p}_j^r - \mathbf{p}_i^c$  is computed as the difference between two 3D points, and lives in  $\mathbb{R}^3$ . Also, consider that two matching points in two different clouds are unlikely to be exactly the same point in space. This introduces an arbitrary error while minimizing the Euclidean distance. In this paper we introduce a technique to reduce this error. This is done by reformulating the correspondence between points as a partial overlap between small surfaces centered in the points, and applying an error metric that takes into account this new aspect.

### 3.1. ICP Probabilistic Formulation

Given the correspondences, we can model the point cloud registration as a maximum-a-posteriori (MAP) estimation problem. In particular, the alignment can be seen as the estimation of the position  $\mathbf{x}$  of a robot given a set of measurements of the surrounding points. In this scenario, one of the cloud is fixed and it represents the global world, while the other it is our current measurement  $\mathbf{z}$ . More formally, we want to estimate the state  $\mathbf{x}^*$  that maximizes the probability  $p(\mathbf{x}|\mathbf{z})$ :

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}} \{p(\mathbf{x}|\mathbf{z})\} \quad (4)$$

$$\text{(Bayes Theorem)} = \underset{\mathbf{x}}{\operatorname{argmax}} \{p(\mathbf{z}|\mathbf{x})p(\mathbf{x})\} \quad (5)$$

$$\text{(Measurement Independence)} = \underset{\mathbf{x}}{\operatorname{argmax}} \left\{ p(\mathbf{x}) \prod_i^N p(\mathbf{z}_i|\mathbf{x}) \right\} \quad (6)$$

$$= \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ -\log \left( p(\mathbf{x}) \prod_i^N p(\mathbf{z}_i|\mathbf{x}) \right) \right\}, \quad (7)$$

where the last equality comes from the fact that maximizing the posterior is the same as minimizing the negative log-posterior.

Let now  $\mathbf{h}_i(\cdot)$  be a known function called *observation* or *measurement model* that, given a state  $\mathbf{x}$ , returns the prediction  $\mathbf{z}_i'$  of the measurement as if the robot is located in  $\mathbf{x}$ . Assuming the measurement noise  $\epsilon_i$  to be zero-mean Normally distributed with information matrix  $\Omega_i$ , we can rewrite in a more explicit form the likelihood of the measurement:

$$p(\mathbf{z}_i|\mathbf{x}) \propto \exp\left(-\frac{1}{2}(\mathbf{h}_i(\mathbf{x}) - \mathbf{z}_i)^T \Omega_i (\mathbf{h}_i(\mathbf{x}) - \mathbf{z}_i)\right), \quad (8)$$

and the prior:

$$p(\mathbf{x}) \propto \exp\left(-\frac{1}{2}(\mathbf{h}_0(\mathbf{x}) - \mathbf{z}_0)^T \boldsymbol{\Omega}_0 (\mathbf{h}_0(\mathbf{x}) - \mathbf{z}_0)\right), \quad (9)$$

for some given function  $\mathbf{h}_0(\cdot)$ , prior mean  $\mathbf{z}_0$  and information matrix  $\boldsymbol{\Omega}_0$ . In our case  $\mathbf{z}'_i = \mathbf{h}_i(\mathbf{x}) = \mathbf{T}^{-1}(\mathbf{x}) \oplus \mathbf{z}_i$

At this point, the maximum-a-posteriori probability encoded in Eq. 7 can be manipulated as follows:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ -\log \left( p(\mathbf{x}) \prod_i^N p(\mathbf{z}_i | \mathbf{x}) \right) \right\} \quad (10)$$

$$= \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ \sum_i^N (\mathbf{z}'_i - \mathbf{z}_i)^T \boldsymbol{\Omega}_i (\mathbf{z}'_i - \mathbf{z}_i) \right\} \quad (11)$$

$$= \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ \sum_i^N \mathbf{e}_i(\mathbf{x})^T \boldsymbol{\Omega}_i \mathbf{e}_i(\mathbf{x}) \right\}. \quad (12)$$

Note that the squared  $l_2$ -norm cost function derived in Eq. 11 is correct only if the measurement noise is Normally distributed. If such assumption changes, and for example the noise follows a Laplace distribution, the cost function will be the  $l_1$ -norm. The reader might notice the similarity of the last equation with the general definition of a least-squares problem described by Eq. 41. For a detailed derivation of the solution of a least-squares problem see Appendix A.

#### 4. Normal ICP

Our method is a variant of ICP that deviates from the general scheme presented in Section 3. Instead of considering only the Euclidean distance between points, we exploit the local properties of the surface, characterized by the surface normals and the local curvature, both in the search for correspondences, and in the computation of the best alignment. In addition to that, to reduce the drift occurring when performing pairwise alignment of consecutive measurements, we construct a global model of the scene by integrating new point clouds.

Figure 1 illustrates the different stages of our system that are resumed below:

- our algorithm computes a Cartesian representation of the 3D point cloud from the raw input data, shall it be a 3D scan, a depth image or a combination of them. See Section 4.1;
- subsequently, our method adds to each point of the cloud the properties of the surrounding surface, namely normal and curvature (see Section 4.2);
- as in a traditional ICP our approach iterates the following steps to refine an initial transform  $\mathbf{T}$ :

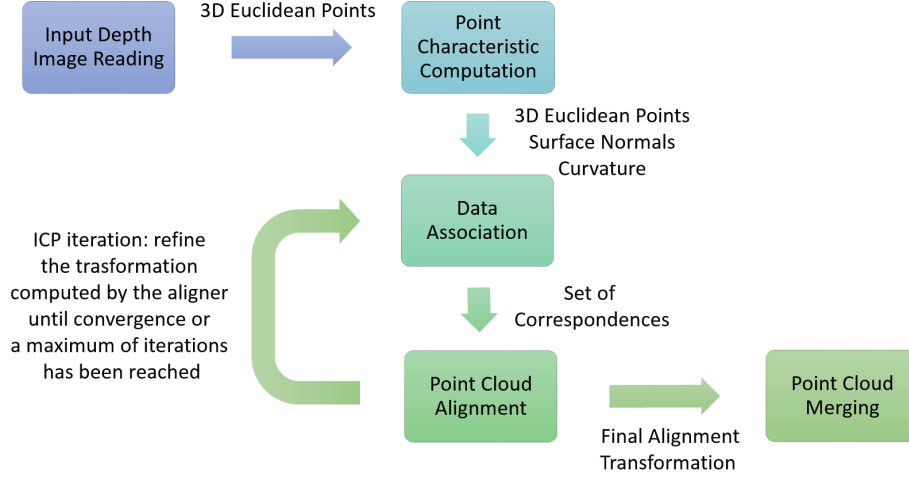


Figure 1: Data flow of our approach NIPC.

- search for correspondences performed using a time efficient projection criterion (ICP approaches using this kind of correspondence search are also known as projective-ICP methods)(Section 4.3);
- computation of the transformation that minimizes the difference between the corresponding points and their normals (Section 4.4). This differs from previous variants of ICP that minimize the distance between the euclidean positions of corresponding points. With respect to common ICP implementation metrics, as demonstrated in our previous work [2], our least squares formulation enlarges the basin of convergence;
- once convergence is reached, or the iterations terminate, our method merges the current cloud into an existing model: the scene (see Section 4.5). In this phase, elements belonging to dynamic objects or inconsistencies are eliminated, and nearby points are merged to keep the size of the cloud tractable.

#### 4.1. Projecting a 3D Point Cloud onto a Range Image and Vice-Versa

Typically, 3D sensors provide an indirect measure of the cloud. As an example, depth cameras generate images where the value of the pixel  $(u, v)$  has the depth  $d$  of the object closest to the observer, and lying on a ray passing through that pixel. These images are normally called depth or range images. To extract a 3D point cloud, it is necessary to apply a function that depends on the intrinsic camera parameters. Similarly, a 3D laser provides for each point an azimuth  $\theta$ , elevation  $\phi$  and the range  $d$  measured at that elevation. Normally, both the azimuth and the elevation angles are discretized, for this reason it is possible to see a 3D scan as a range image where  $(\theta, \phi)$  are the coordinates of a

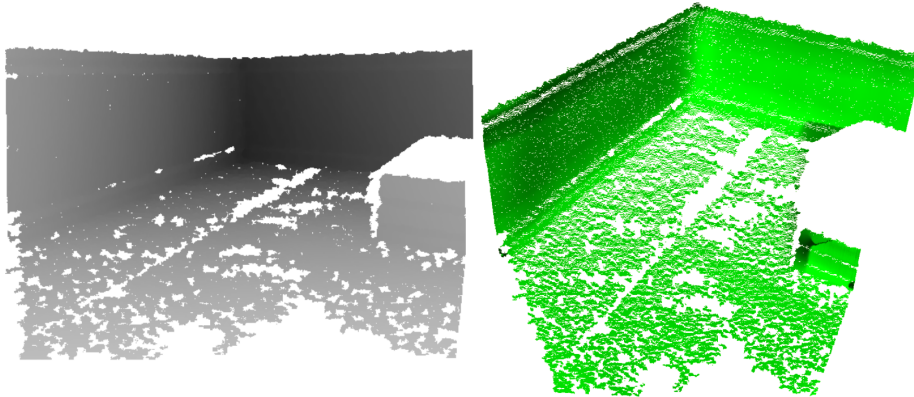


Figure 2: Example of the effect of applying the function  $\pi^{-1}$  (right) to the input raw measurement (left).

pixel on a spherical surface. The value  $d$  of the pixel is its depth. Thus, without loss of generality, we can define a projection function  $\mathbf{s} = \pi(\mathbf{p})$  that maps a point  $\mathbf{p}$  from the Cartesian to polar coordinates. The point in measurement space  $\mathbf{s}$  can be either a  $(u, v, d)$  or  $(\theta, \phi, d)$  triplet depending on the type of sensor used. Let then  $\pi^{-1}(\mathbf{s})$  be the inverse of a projection function that maps a raw sensor measurement to a point in the Cartesian space.

The very first step of our method is to apply the function  $\pi^{-1}$  to the raw measurements to compute a Cartesian representation of the point cloud. This is done once each time a new measurement becomes available. Fig. 2 shows an example result of this step. In case one uses multiple sensors this procedure is performed individually for each of them, producing one Cartesian cloud for each sensor.

#### 4.2. Computing Local Surface Statistics and Normals

A point measurement obtained by a range sensor is a sample of a piece-wise continuous surface. This is the core idea of the point-to-plane and plane-to-plane metrics used by Chen *et al.* [8] and Segal *et al.* [4].

We locally characterize the surface around a point  $\mathbf{p}_i$  with its surface normal  $\mathbf{n}_i$  and curvature  $\sigma_i$ . In order to compute the normal, we extract the covariance matrix of the Gaussian distribution  $\mathcal{N}_i^s(\mu_i^s, \Sigma_i^s)$  of all the points that lie in a sphere of radius  $R$  centered in the query point  $\mathbf{p}_i$ . In our evaluation, we found experimentally that 10 cm is a good value for  $R$  in indoor environments. If the surface is well defined, meaning that it is reasonably flat, it can be approximated by a plane, and only one eigenvalue of the covariance will be substantially smaller than the other two. The surface normal is selected as the eigenvector associated to the smallest eigenvalue and, if necessary, reoriented towards the observer point of view.

Likewise GICP, for each point  $\mathbf{p}_i$  we compute the mean  $\mu_i^s$  and the covariance  $\Sigma_i^s$  by using Eq. 3. To determine the set  $\mathcal{V}_i$  of points inside the sphere, a standard

implementation of the above algorithm would require expensive queries on a KD-tree data structure where the cloud is stored. To speed up the calculation, we adopt an approach based on integral images, described in [44], that allows us to compute Eq. 3 in constant time. Once the parameters of the Gaussian are evaluated, we get its eigenvalue decomposition as follows:

$$\Sigma_i^s = \mathbf{R} \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \mathbf{R}^T. \quad (13)$$

In the previous equation  $\lambda_{1:3}$  are the eigenvalues of  $\Sigma_i^s$  in ascending order, and  $\mathbf{R}$  is the matrix of eigenvectors that represent the axes of the ellipsoid that approximates the point distribution. We use the curvature  $\sigma_i = \lambda_1 / (\lambda_1 + \lambda_2 + \lambda_3) \in [0, 1]$  to discriminate how well the surface is fitted by a plane (see [45] for more details). The smaller it is  $\sigma$ , the flatter is the surface around the point.

In the real case, due to the sensor noise, even surfaces that are perfectly planar will not have a perfect 0 curvature (or in other words the smallest eigenvalue null). To reduce the effect of this noise, when needed, we modify the covariance matrix  $\Sigma_i^s$  by forcing a “disc” shape. This can be achieved by modifying the length of the axis of the ellipsoid in the following way:

$$\Sigma_i^s \leftarrow \mathbf{R} \begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{R}^T \quad (14)$$

where  $\epsilon$  is a small coefficient, in our experiments  $\epsilon$  is equal to 0.001. If the surface is not well approximated by a local plane (the curvature is high), we do not modify  $\Sigma_i^s$ . This technique has been first introduced by GICP.

After this step, each point  $\mathbf{p}_i$  belonging to the cloud is augmented with its own surface characteristics  $\langle \mu_i^s, \Sigma_i^s, \sigma_i \rangle$ . Figure 3 (left image) shows an example of its typical outcome.

In case one uses multiple sensors, our approach adds the surface normal and the curvature of the points individually for each point cloud. This operation is carried out in the reference frame of the sensor. Let  $\mathcal{P}_{1:k}^c$  be these clouds. By knowing the pose of each sensor  $\mathbf{K}_{1:k}$  with respect to *the reference frame of the robot* (or more in general a common reference frame), our method computes a global cloud  $\mathcal{P}^c$  in such frame as the union of the points in  $\mathcal{P}_{1:k}^c$ , after applying the corresponding transform:

$$\mathcal{P}^c = \bigcup_{\mathbf{K}_i \in \mathcal{S}} \mathbf{K}_i \oplus \mathcal{P}_i^c \quad (15)$$

with  $\mathcal{S}$  being the set of sensors used.

#### 4.3. Projection Based Correspondence Finding

Similar to [6] and [11] we search for the correspondences by using a projection criterion. In particular, we project the reference cloud on a depth image whose

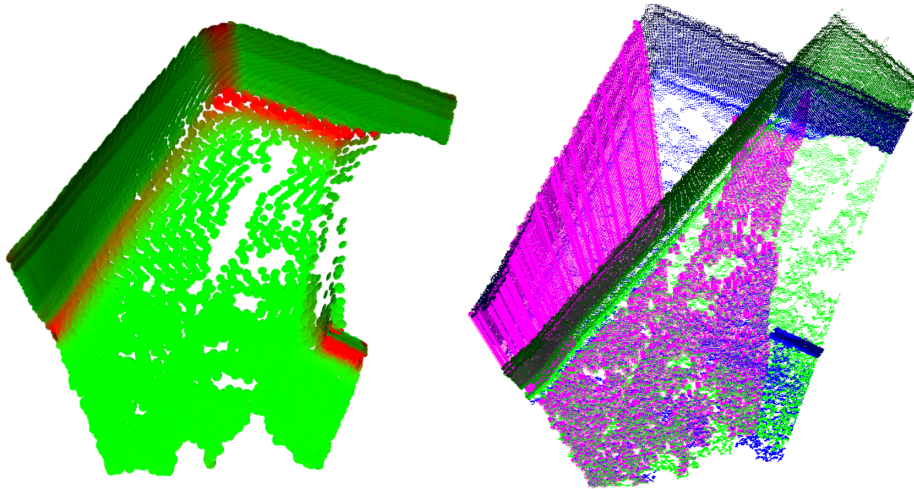


Figure 3: Left: example of the effect of extracting the surface statistics. According to their curvature  $\sigma$ , green ellipsoids correspond to points lying on flat regions while red ones to corners. Right: example of the output generated after evaluating the data association. Correspondences are shown as violet lines connecting matching points in the blue and green clouds.

viewpoint is the current estimate of the transformation, and points that fall in the same pixel coordinates that have compatible normals and curvature are labeled as a correspondence. Note that this projection operation does not need to be performed for the current cloud, since we have already the corresponding depth image. While the approach by itself is straightforward, it is necessary to design an efficient implementation due to the potentially large amount of data the algorithm has to manipulate. We describe now a procedure that reduces the memory movements and brings the clouds manipulation to the minimum. The first assumption we make is that the point clouds are stored in arrays not necessarily ordered in any way. To describe our procedure we first introduce the concept of *index image*. Given a projection model  $\pi(\mathbb{R}^3) \rightarrow \mathbb{R}^3$ , an index image  $\mathcal{I}$  is matrix where the element of coordinates  $(u, v)$  contains the *index* of the point  $\mathbf{p}_i$  in the array such that  $\pi(\mathbf{p}_i) \rightarrow (u, v, d)^T$ . If more than one point falls in the same pixel, we store the index of the point closest to the observer and having a normal oriented towards the center of projection (the point of view).

Now, Let  $\mathcal{I}(\pi, \mathcal{P})$  be an index image computed from the cloud  $\mathcal{P}$  using the projection function  $\pi$ . At the beginning of the registration process we compute the current index image by projecting all points of the current cloud  $\mathcal{P}^c$ :

$$\mathcal{I}^c = \mathcal{I}(\pi, \mathcal{P}^c). \quad (16)$$

Since our optimization procedure keeps fixed the current cloud,  $\mathcal{I}^c$  does not move during the entire alignment, thus we do not need to recompute it at every iteration.

Conversely, we need to calculate the index image  $\mathcal{I}^r$  of the reference cloud  $\mathcal{P}^r$  at each iteration. Note that, by taking approximately 30% of the entire

registration time, together with the correspondence search heuristic this step represents an important bottleneck during the alignment. However, this is necessary since, at each iteration, we have to re-map the reference cloud in the frame of the current cloud using the most recent transform  $\mathbf{T}$ :

$$\mathcal{I}^r = \mathcal{I}(\pi, \mathbf{T} \oplus \mathcal{P}^r). \quad (17)$$

Here  $\oplus$  applies the transformation  $\mathbf{T}$  to the whole cloud  $\mathcal{P}^r$ .

The reference cloud  $\mathcal{P}^r$  can be large, thus this reprojection is an expensive step. An easy optimization we perform consists in clipping  $\mathcal{P}^r$  around the current location once before starting the iterations.

Let  $\mathcal{I}_{u,v}$  be the value of the pixel of coordinates  $(u, v)$  in the index image  $\mathcal{I}$ . At this point, from  $\mathcal{I}^r$  and  $\mathcal{I}^c$  we generate a candidate correspondence for each pixel coordinates  $(u, v)$  as  $\langle i, j \rangle_{u,v} = \langle \mathcal{I}_{u,v}^c, \mathcal{I}_{u,v}^r \rangle$ . A candidate correspondence  $\langle i, j \rangle$  between the points  $\mathbf{p}_i^c$  and  $\mathbf{p}_j^r$  is rejected if one of the following constraints is verified:

- either  $\mathbf{p}_i^c$  or  $\mathbf{p}_j^r$  do not have a well defined normal;
- the distance between the point in the current cloud and the reprojected point in the reference cloud is greater than a threshold:

$$\|\mathbf{T} \oplus \mathbf{p}_j^r - \mathbf{p}_i^c\| > \epsilon_d; \quad (18)$$

- the magnitude of the log ratio of the curvatures of the points is larger than a threshold:

$$|\log \sigma_i^c - \log \sigma_j^r| > \epsilon_\sigma; \quad (19)$$

- the angle between the normal of the current point and the reprojected normal of the reference point is greater than a threshold:

$$\mathbf{n}_i^c \cdot (\mathbf{T} \oplus \mathbf{n}_j^r) < \epsilon_n. \quad (20)$$

In our evaluation, we found experimentally that good values for  $\epsilon_d$ ,  $\epsilon_n$  and  $\epsilon_\sigma$  are respectively 0.5 m, 0.95 and 1.3. Figure 3 (right image) illustrates an example of the output of the correspondence selection. By using index images, we avoid copying points, normals and covariance matrices in temporary structures, thus increasing the whole speed of the algorithm.

In case of multiple sensors this procedure is repeated independently for each of them, producing two index images for each sensor  $\mathcal{I}_k^c$  and  $\mathcal{I}_k^r$ , by considering the relative pose of the sensor on the mobile base. This leads to the following formulation for Eq. 16 and Eq. 17:

$$\mathcal{I}_k^c = \mathcal{I}(\pi, \mathbf{K}_k^{-1} \oplus \mathcal{P}^c) \quad \mathcal{I}_k^r = \mathcal{I}(\pi, \mathbf{T} \oplus \mathbf{K}_k^{-1} \oplus \mathcal{P}^r). \quad (21)$$

The remaining operations remain the same.

#### 4.4. Computing the Relative Transform

Once we have evaluated a set of correspondence pairs  $\mathcal{C} = \langle i, j \rangle_{1:M}$ , we compute the relative transformation between the two frames by using an iterative least squares formulation (see Appendix A for the mathematical derivation of the least-squares problem). Recall that each  $i^{\text{th}}$  point in a cloud contains the following information: the Cartesian coordinates  $\mathbf{p}_i$ , the surface curvature  $\sigma_i$ , the surface normal  $\mathbf{n}_i$  and the covariance matrix  $\Sigma_i^s$ .

Let  $\tilde{\mathbf{p}}$  be a point with normal  $\tilde{\mathbf{p}} = (\mathbf{p}^T \ \mathbf{n}^T)^T$  and  $\mathbf{T}$  a transformation matrix parametrized by using a rotation matrix  $\mathbf{R}$  and a translation vector  $\mathbf{t}$ . The  $\oplus$  operator on points with normals can be defined as:

$$\tilde{\mathbf{p}}' = \mathbf{T} \oplus \tilde{\mathbf{p}} = \begin{pmatrix} \mathbf{R}\mathbf{p} + \mathbf{t} \\ \mathbf{R}\mathbf{n} \end{pmatrix} \quad (22)$$

This means that, given a correspondence pair and the current transform  $\mathbf{T}$ , the error  $\mathbf{e}_{ij}(\mathbf{T})$  is a 6D dimensional vector

$$\mathbf{e}_{ij}(\mathbf{T}) = \mathbf{T} \oplus \tilde{\mathbf{p}}_j^r - \tilde{\mathbf{p}}_i^c. \quad (23)$$

Substituting Eq. 23 in Eq. 1 leads to the following objective function:

$$\sum_{\mathcal{C}} \mathbf{e}_{ij}(\mathbf{T})^T \tilde{\mathbf{\Omega}}_{ij} \mathbf{e}_{ij}(\mathbf{T}). \quad (24)$$

Here  $\tilde{\mathbf{\Omega}}_{ij}$  is a  $6 \times 6$  information matrix whose goal is to scale the different components of the error. The ideal behavior we want to obtain from that matrix is to rotate corresponding points so that their normals align, while at the same time, the distance along the normal direction is penalized. In addition to this, it also neglects the distance along the plane tangents. With this in mind, we impose the translational and normal components to be independent between each other, and we select  $\tilde{\mathbf{\Omega}}_{ij}$  as follows:

$$\tilde{\mathbf{\Omega}}_{ij} = \begin{pmatrix} \mathbf{\Omega}_i^s & \mathbf{0} \\ \mathbf{0} & \mathbf{\Omega}_i^n \end{pmatrix}. \quad (25)$$

Here  $\mathbf{\Omega}_i^s = \Sigma_i^{s-1}$  is the surface information matrix around the current point  $\mathbf{p}_i^c$ , and  $\mathbf{\Omega}_i^n$  is the information matrix of the normal. If the curvature is small enough we force  $\mathbf{\Omega}_i^n$  to have a disk shape as follows:

$$\mathbf{\Omega}_i^n \leftarrow \mathbf{R} \begin{pmatrix} \frac{1}{\epsilon} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{R}^T, \quad (26)$$

otherwise we impose  $\mathbf{\Omega}_i^n$  to the identity. By generating such information matrices, a correspondence pair is minimized by allowing the points to slide onto each other along the tangential direction of the surface, and rotating them so that their normals align. The reader might notice that setting  $\mathbf{\Omega}_i^n$  to zero makes our



objective function identical to point-to-plane GICP. We refer the reader to our previous work [1] where we investigated the effects of the weight of the normals in the registration.

To reduce the effect of outliers we further robustify the error function by clamping the norm of the  $\chi_{ij}^2$  of each point to a maximum value. This method is known as “winsorization”. To this end we compute a scaling factor  $\gamma_{ij}$  for each information matrix  $\tilde{\mathbf{\Omega}}_{ij}$  as

$$\gamma_{ij} = \begin{cases} 1 & \text{if } \chi_{ij}^2 < K \\ \frac{K}{\chi_{ij}^2} & \text{otherwise} \end{cases}, \quad (27)$$

where  $K$  is a thresholding value of the  $\chi_{ij}^2$  error that discriminates when a pair of corresponding points  $i$  and  $j$  is considered an outlier or not.

Our method minimizes Eq. 24 by using a local parametrization of the perturbation in the following form:  $\Delta \mathbf{T} = (\Delta t_x \ \Delta t_y \ \Delta t_z \ \Delta q_x \ \Delta q_y \ \Delta q_z)^T$ . It is composed by the translation vector  $\Delta \mathbf{t} = (\Delta t_x, \Delta t_y, \Delta t_z)^T$  and the imaginary part of the normalized quaternion  $\Delta \mathbf{q} = (\Delta q_x, \Delta q_y, \Delta q_z)^T$ . In order to smooth the convergence of the whole system we use a damped Gauss-Newton algorithm. This prevents the solution to take too large steps that might be caused by non-linearities or wrong correspondences. More formally, at each iteration our approach solves the following linear system:

$$(\mathbf{H} + \lambda \mathbf{I}) \Delta \mathbf{T} = \mathbf{b}. \quad (28)$$

Here,  $\mathbf{H} = \sum \mathbf{J}_{ij}^T \tilde{\mathbf{\Omega}}_{ij} \mathbf{J}_{ij}$  is the approximated Hessian and  $\mathbf{b} = \sum \mathbf{J}_{ij}^T \tilde{\mathbf{\Omega}}_{ij} \mathbf{e}_{ij}(\mathbf{T})$  is the vector of residuals (more details can be found in our previous works [1][2]). Once we computed the perturbation  $\Delta \mathbf{T}$  from Eq. 28, we refine the current transformation as:

$$\mathbf{T} \leftarrow \Delta \mathbf{T} \oplus \mathbf{T}. \quad (29)$$

The Jacobian  $\mathbf{J}_{ij}$  is calculated by evaluating the derivative of Eq. 23 in  $\Delta \mathbf{T} = \mathbf{0}$ :

$$\mathbf{J}_{ij} = \left. \frac{\partial \mathbf{e}_{ij}(\Delta \mathbf{T} \oplus \mathbf{T})}{\partial \Delta \mathbf{T}} \right|_{\Delta \mathbf{T}=\mathbf{0}} = \begin{pmatrix} \mathbf{I} & -2[\mathbf{T} \oplus \mathbf{p}_j^*]_{\times} \\ \mathbf{0} & -2[\mathbf{T} \oplus \mathbf{n}_j^*]_{\times} \end{pmatrix} \quad (30)$$

where  $[\mathbf{v}]_{\times}$  is the cross product matrix of the vector  $\mathbf{v}$  (see Appendix B and Appendix C for the mathematical derivation of the Jacobian  $\mathbf{J}_{ij}$ ). Note that, it is possible to construct efficiently the linear system in Eq. 28 by exploiting the block structure of the Jacobian, and its substantial sparsity.

In case of multiple sensors, we carry on a single optimization step based on all correspondences found in all sensor frames. The aim of this optimization step is to compute the transform of the robot origin that minimizes all correspondences. Since all points in the clouds are expressed in a common frame, we do not need to take care of the sensor transformations  $\mathbf{K}_{1:k}$  that have been already embedded in the points.

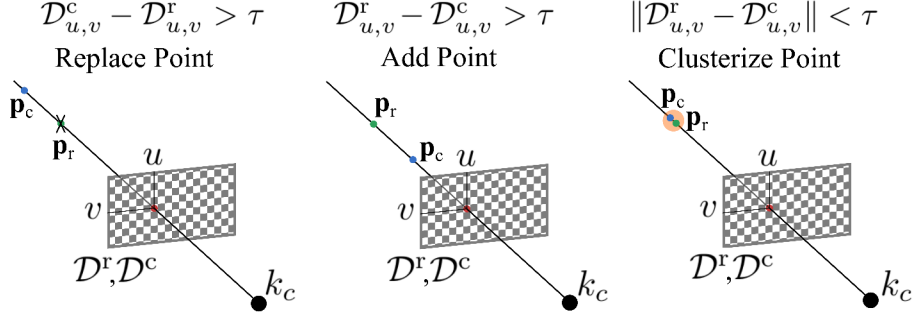


Figure 4: Graphical example showing the three possible cases considered by our method during the clustering step of the merging process. In the image,  $k_c$  represents the origin of the current point cloud.

#### 4.5. Point Cloud Merging

To gather a map, or track the pose of a robot, it is common to perform several pairwise alignments. Each registration introduces a small error and the map can rapidly become inconsistent. To limit this drift, performing incremental alignments on the same model has been reported to be very effective. After registering each new measurement, a local map is augmented with the new aligned data. This is at the base of KinFu, and many successful 2D mapping approaches. The model generated in this way is typically locally consistent.

A naive implementation of this strategy would result in constant growth of the points in the reference model, and since the performance of the algorithm depends directly on the number of points, we might expect a linear increase in per-frame computation as the time passes. Note that, many of these points are samples of the surface very close in space. To keep the process tractable, and enhance the quality of the model, it is common practice to apply some decimation or aggregation technique. Furthermore, each point is a measurement that comes with its own error that depends by the sensor. In case of depth cameras typically this error grows with the distance measured. Based on the above considerations, while doing this decimation, it is frequent to refine the point statistics.

In our current implementation we keep for each point the coefficients of a 1D normal distribution that describes the isotropic uncertainty of the measurements in the space. In principle, one could represent the full 3D distribution of a point noise, however, our experiments have shown that an isotropic noise provides the best trade off between computation and accuracy.

The aim of a merging procedure is to fuse the points of two clouds to get a unique consistent model, and at the same time to refine the statistics of the points. The new model should have a regular density of measurements. A merging procedure therefore consists of two components:

- *clustering*, where we partition the input clouds in sets of points that will contribute to a single one in the output cloud;

- *update*, where all statistics of the new point are fused to a new one.

Optionally, if the scene is dynamic we might want also to *remove* points from the model.

In this section we propose a projection based approach for clustering that leverages on the index-image data structure presented before. This model allows us to efficiently group the points into sets and it handles occlusion. We omit the case of multiple sensor as it is a straightforward extension of the single sensor case.

We assume to have two aligned clouds:  $\mathcal{P}^r$  and  $\mathcal{P}^c$ , and to know the relative transform  $\mathbf{T}$  between them. We can then compute a view of the reference as if it was observed from the origin of  $\mathcal{P}^c$ . This is done already during the computation of the correspondences through Eq. 17. In addition to the index image  $\mathcal{I}^r$ , we also compute the range images  $\mathcal{D}^r$  and  $\mathcal{D}^c$  for both  $\mathcal{P}^r$  and  $\mathcal{P}^c$ . This operation naturally handles occlusions. Let  $\mathcal{D}_{u,v}$  be the value of the pixel of coordinates  $(u, v)$  in the index image  $\mathcal{D}$ . By selecting a distance threshold parameter  $\tau$ , we scan the two range images pixel by pixel and based on the depth comparison we perform the following operations:

- if  $\mathcal{D}_{u,v}^c - \mathcal{D}_{u,v}^r > \tau$ , the new ray crosses an existing element of the reference surface. In absence of noise, this results in *replacing* the point  $\mathcal{D}_{u,v}^r$  with the point  $\mathcal{D}_{u,v}^c$ . In other words, if we see through a point in the reference cloud we replace it with the corresponding transformed point in the current cloud;
- if  $\mathcal{D}_{u,v}^r - \mathcal{D}_{u,v}^c > \tau$ , the new ray ends much before a point in the previous cloud. This results in the new point to be added to the reference model as it might be due to a new object entering in the scene;
- if  $\|\mathcal{D}_{u,v}^r - \mathcal{D}_{u,v}^c\| < \tau$ , the two points are close, so they are likely to belong to the same surface, and they end up in the same cluster.

Figure 4 illustrates these three cases.

Please note that, compared to traditional clustering approaches that rely on voxelization, this method makes use of the free space and naturally handles the removal of old points in the scene. In the worst of the cases we have a number of clusters equal to the number of pixels in the image.

Once we are done with the clustering, we update the noise statistics of each point and we adjust its estimate by using an information filter as follows:

$$\mathbf{\Omega}_{u,v} = \sum_{i=1}^N \mathbf{\Omega}_i^m \quad \mu_{u,v} = \mathbf{\Omega}_{u,v}^{-1} \sum_{i=1}^N \mathbf{\Omega}_i^m \cdot \mathbf{p}_i \quad (31)$$

where  $\mathbf{\Omega}_i^m$  is the inverse of the covariance matrix of the  $i$ -th point  $\mathbf{p}_i$  falling in the cluster  $(u, v)$ . We replace then the estimate of each point in the reference cloud with the mean of the cluster.

After updating the point positions, we need to recompute the normals by applying the procedure described in Section 4.2. Note that only the normals

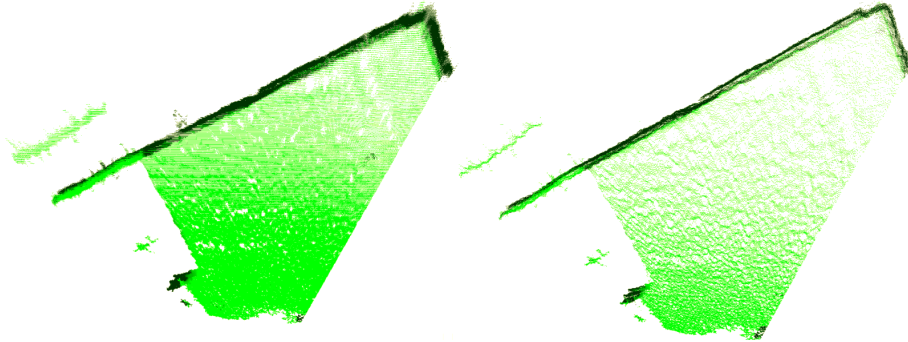


Figure 5: Example of merging. Note the difference in the thickness of the walls due to sensor uncertainty before the merging (left), and after (right).

within the area of the cloud where points have been inserted or modified are recomputed. Fig. 5 highlights how the merging helps to remove artifacts (thick wall) generated by a naive accumulation of the point clouds.

## 5. Speeded Up NICP

Thanks to its parallel implementation, NICP is able to execute in real-time. However, this can not be ensured when NICP is used in conjunction with low-end computers or with multiple sensors.

In the remainder of this section we highlight the main bottlenecks of the procedure described before, and how we addressed them to further enhance the processing time performance.

### 5.1. Computation of the Normals and the Point Statistics

The computation of the surface statistics (Section 4.2), requires the solution of a third degree polynomial for each pixel to calculate the eigenvalue decomposition of the covariance.

A faster but less accurate alternative way, is to compute the normals directly in the depth image, as the normalized cross product of points projecting onto adjacent pixels (see Fig. 6). Namely, given the point  $\mathbf{p}_{u,v}$  that projects in the pixel  $(u, v)$ , we compute its normal direction as follows:

$$\mathbf{n}'_{u,v} = (\mathbf{p}_{u+\Delta,v} - \mathbf{p}_{u-\Delta,v}) \times (\mathbf{p}_{u,v+\Delta} - \mathbf{p}_{u,v-\Delta}) \quad \mathbf{n}_{u,v} = \frac{\mathbf{n}'_{u,v}}{\|\mathbf{n}'_{u,v}\|}. \quad (32)$$

Here,  $\Delta$  is an offset on the image of a few pixels. This procedure is substantially faster, however leads to poor quality normals, that improve as  $\Delta$  increases. Typical values of  $\Delta$  range from 2 to 5 pixels. To further smooth the normal we apply a block filtering to the normals.

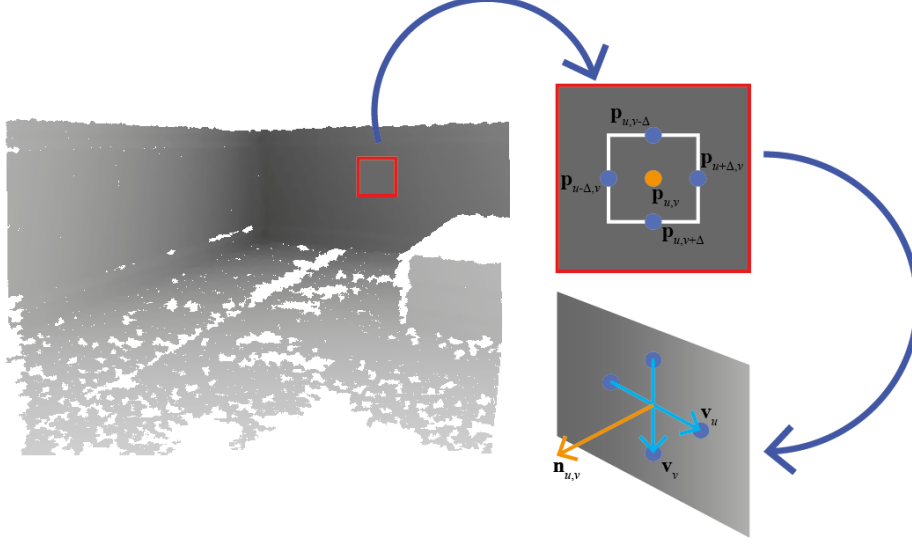


Figure 6: Graphical example of cross product surface normal computation. The red square on the top right is a magnification of a part of the range image shown on the left. We select a square region on the depth image, around the query point  $\mathbf{p}_{u,v}$ , whose neighbor points  $\mathbf{p}_{u+\Delta,v}$ ,  $\mathbf{p}_{u-\Delta,v}$ ,  $\mathbf{p}_{u,v+\Delta}$  and  $\mathbf{p}_{u,v-\Delta}$  are used to compute the normal via cross product (bottom right image  $\mathbf{v}_u \times \mathbf{v}_v$ ).

### 5.2. Computing the Relative Transform

During the alignment the reference cloud is “moved” at each iteration. Since the number of points in the reference augment over time, the computation increases accordingly. An alternative is to perform the optimization by computing the transform  $\mathbf{T}$  that maps the current cloud onto the reference as follows:

$$\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{\mathbf{c}} (\mathbf{T} \oplus \mathbf{p}_i^c - \mathbf{p}_j^r)^T \Omega_{ij} (\mathbf{T} \oplus \mathbf{p}_i^c - \mathbf{p}_j^r). \quad (33)$$

This approach, combined with our projection based data association, has the major drawback of “fixing” the reference projection. If the offset is large, the current cloud might be dragged out of the field of view of the reference during the optimization. To cope with this problem, we use a projection function in the correspondence search that has a broader field of view with respect to the original one. In this way, we capture in the reference projection a larger portion of the scene, and during the optimization the current projection is more likely to stay in the frustum used to render the reference. With this slightly different objective function to be minimized, the Jacobian  $\mathbf{J}_{ij}$  becomes:

$$\mathbf{J}_{ij} = \frac{\partial \mathbf{e}_{ij}(\Delta \mathbf{T} \oplus \mathbf{T})}{\partial \Delta \mathbf{T}} \Big|_{\Delta \mathbf{T}=\mathbf{0}} = \begin{pmatrix} \mathbf{I} & -2[\mathbf{T} \oplus \mathbf{p}_j^c]_{\times} \\ \mathbf{0} & -2[\mathbf{T} \oplus \mathbf{n}_j^c]_{\times} \end{pmatrix} \quad (34)$$

Additionally, our enhanced version performs a set of pyramidal alignments at increasing resolutions rather than operating at a fixed one. This has the

benefit of enlarging the convergence basin for data association as the pixels become bigger, and to speed up the computation. Accuracy is preserved when performing the alignment at higher resolutions. The reader will notice that this can be done by simply modifying the parameters of the projection function  $\pi$ .

### 5.3. Point Cloud Merging

The update strategy used in the merging procedure requires to keep an information filter per point, and then to recompute the statistics in the observed regions.

To speed up the computation, for each point in a cloud we keep a mono dimensional information  $\omega_{u,v}$  representing the uncertainty of both normal and point. The update of the point and its normal is carried out in an information-like form as follows:

$$\omega_{u,v} = \sum_{i=1}^N \Omega_i^m \quad \mu_{u,v}^{\mathbf{p}} = \omega_{u,v}^{-1} \sum_{i=1}^N \omega_i \cdot \mathbf{p}_i \quad \mu_{u,v}^{\mathbf{n}} = \omega_{u,v}^{-1} \sum_{i=1}^N \omega_i \cdot \mathbf{n}_i \quad (35)$$

The normals are scaled to unit norm after computing the means.

### 5.4. Optimizing Memory Accesses

From a hardware point of view, the main bottleneck of the procedure lies in the scattered memory accesses induced by the double indirection through index images. We observed a 20% increase in performance when reordering the points before each alignment. Our point clouds are represented as unordered arrays. During all procedures, the access to the arrays is either linear or follows the order of the pixels in the index image. An alignment operation is likely to access the pixels in these two orders. To guarantee more “regular” accesses, before each alignment we sort the points in the reference cloud so that the first block of the array are the points in the index image used for computing the correspondences, in the order set by the index image.

## 6. Experiments

In our previous work [2], we performed a convergence study of the error function of NICP. More recently, in [1], we also evaluated a non incremental version of NICP, thus without merging, against some of the current state-of-the-art methods. In this paper we aim at comparing a incremental implementation of NICP that integrates the merging procedure described before. For the remainder of this work, we will refer to the speeded up variant of NICP introduced in Sec. 5 with the name of SNICP. In addition to this, we also show some qualitative results of our tracking algorithm working both in a environment with dynamic objects, and with multiple sensors.

### 6.1. Incremental Tracking

We compared NICP and SNICP with point-to-plane GICP, NDT and KinFu using a large depth camera standard benchmarking dataset developed by Pomerleau *et al.* [46] (ETH Kinect). To allow for a fair comparison, we added to GICP and NDT the same components that allows NICP to perform incremental registrations, and we kept the merging parameters fixed to the same value for all of them in all experiments. SNICP and KinFu come with their own merging systems. We used the KinFu implementation provided by the Point Cloud Library (PCL) [47], while for NDT we considered the ROS [48] package suggested by the authors, and available on the web. Since point-to-plane GICP is a special case of our algorithm, where the error of the normals in the optimization stage is neglected, and the correspondences are selected based only on point distance, we used our own GICP implementation. Note that our implementation of GICP benefits of all data structures and of surrounding algorithms that are used in NICP, namely the extraction of the statistics and the calculation of the correspondences. For NDT and KinFu we used the default parameters found in their implementations. In the case of GICP, SNICP and NICP, instead, we used the values indicated in the previous sections and we forced them to run all the iterations independently by the  $\chi^2$  value obtained. In particular, the number of iterations has been set to 10 for both GICP and NICP, while SNICP was configured to perform 3 iterations at three different levels of dimension of the input depth images (1/4, 1/2 and full size). An aggregated list of parameters used in the experiments for NICP can be found on the website linked before.

To measure the performance of an algorithm we used the benchmarking tools of Sturm *et al.* [49], and we computed the Relative Pose Error (RPE). The RPE measures the pairwise alignment error between successive poses, and it is one of the most common metrics used for the evaluation of visual odometry or camera tracking systems. More formally, given the groundtruth transform between two consecutive point clouds  $\mathbf{T}_{gt}$ , and the transform  $\mathbf{T}$  calculated by one of the algorithms considered in the comparison, the RPE computes the translational and rotational difference of the offset transform  $\Delta\mathbf{T}_o = \mathbf{T}_{gt}^{-1}\mathbf{T}$ . In particular, the translational error  $t_e$  is computed as the module of the translation vector  $\mathbf{t}_o$  of  $\Delta\mathbf{T}_o$ . The rotational error  $R_e$ , instead, is taken as the rotation angle of the rotation matrix  $\mathbf{R}_o$  associated to  $\Delta\mathbf{T}_o$ . In the ideal case when the transform  $\mathbf{T}$  is exactly equal to  $\mathbf{T}_{gt}$ , the transformation offset  $\mathbf{T}_o$  would be the identity matrix, and thus both the translational and rotational errors equal to zero. All tests have been performed on a i7-3630QM, over a single core, running at 2.4 GHz and with an nVidia GeForce GT 650M graphics card.

Table 1: Mean and standard deviation of the point cloud registration time for each algorithm over all the sequences of the ETH Kinect dataset.

	NDT	GICP	KinFu	SNICP	NICP
<b>Mean</b>	183 ms	37 ms	52 ms	18 ms	38 ms
<b>Std. Dev.</b>	17 ms	4 ms	4 ms	2 ms	6 ms

As shown in Table 1, in terms of processing time, NDT was the slowest algorithm in computing a single registration. This is mainly due to the fact that it does not scale very well when the number of points in the scene increases. KinFu took about 50 ms to process each new point cloud, but this low frame rate is probably caused by the low-end graphics card in our system. GICP and NICP were able to execute close to real-time, while SNICP resulted to be the fastest processing clouds at a rate of almost 60Hz. More precisely, SNICP is more than  $\sim 50\%$  faster than NICP.

The dataset is composed by several sequences of depth and RGB images acquired with a RGB-D camera. For benchmarking purposes, the ground-truth is available. It covers 3 different environments of increasing complexity (low, high, medium), with 3 types of motions (rotational, translational, fly) at 3 different speeds (slow, medium, fast). Sequences recorded with high camera motions allow to test the robustness of the algorithms to poor initial guesses. In fact, a big camera velocity implies an increasing average distance between two processed frames.

Table 2 reports the results obtained by processing all the sequences in the dataset with NDT, GICP, KinFu, SNICP and NICP algorithms. For each approach and for each test, we processed all the images in sequence and we generated the estimated trajectories. Note that, since this dataset is recorded with a high frame rate, the RPE is computed on poses with a difference in time of 0.25 seconds. Green cells in the table highlight the best mean result for each specific test among all the compared algorithms. As the reader can see NICP, or its time efficient variant SNICP, are in most of the cases more accurate with respect to the other methods. The surface normals, in conjunction with a new merging method to incrementally align point clouds, give a major contribute to increase the overall accuracy and robustness of our approach, thus leading to better camera tracking results. The reader might notice that in some tests, in particular the fast ones, both the translational and rotational errors are quite large. This is caused by the fact that, in the experiments, the algorithms lost the tracking due to the large displacements between the clouds.

Table 2 also shows that SNICP, despite sacrificing a little of accuracy in favor of computation time efficiency, obtains results comparable with those of NICP. Nowadays, due to project or environmental constraints, many robotics applications have to rely only on low-end hardware or they need to use multiple sensors simultaneously. In these cases, it is impossible to use methods like KinFu, or even NICP, since they require at least a mid-range laptop to reach near real-time performance. Consider also that in the case of KinFu, a NVIDIA graphics card is mandatory to run the algorithm, indeed it requires CUDA parallel computing. These results highlight the fact that, by exploiting low computational time methods, SNICP offers a very good alternative in this kind of situations.

## 6.2. Tracking in Environments with Dynamic Objects

In this test case we run our tracking approach inside a office like environment. While registering, both a chair and a person moved inside the field of view of



Table 2: Mean and standard deviation of the relative translational and rotational error for all the sequences of the ETH Kinect dataset. Green cells in the table highlight the best mean result for each specific test among all the compared algorithms (NDT, GICP, KinFu, SNICP and NIPC).

Sequence	Mean / Standard Deviation Translational Error [m]					Mean / Standard Deviation Rational Error [deg°]				
	NDT	GICP	KinFu	SNICP	NIPC	NDT	GICP	KinFu	SNICP	NIPC
high-fast-f	0.343/0.566	0.094/0.114	0.337/0.436	0.178/0.105	<b>0.090/0.106</b>	17.0/30.6	<b>6.1/4.3</b>	7.3/6.6	9.0/7.2	6.5/4.6
high-fast-r	0.409/0.821	0.419/0.493	0.624/0.546	<b>0.060/0.029</b>	0.195/0.209	25.8/29.1	22.7/17.4	18.1/10.2	20.0/9.8	<b>15.5/9.0</b>
high-fast-t	0.083/0.159	0.052/0.045	0.092/0.124	0.091/0.070	<b>0.048/0.023</b>	5.0/5.1	4.3/3.1	4.2/2.2	<b>4.1/2.0</b>	4.2/2.1
high-med-f	0.103/0.214	0.026/0.019	0.036/0.045	0.051/0.045	<b>0.025/0.018</b>	5.1/7.5	2.5/1.5	<b>2.4/1.4</b>	3.2/2.0	2.5/1.7
high-med-r	0.106/0.344	0.058/0.096	0.182/0.281	<b>0.021/0.011</b>	0.051/0.080	6.5/16.3	3.8/2.6	4.5/4.2	3.7/2.4	<b>3.6/2.4</b>
high-med-t	0.036/0.043	0.014/0.008	<b>0.013/0.010</b>	0.024/0.019	0.014/0.009	1.9/1.5	<b>1.3/0.8</b>	1.4/0.8	1.5/1.0	1.4/0.8
high-slow-f	0.058/0.186	0.010/0.007	0.016/0.050	0.012/0.008	<b>0.009/0.007</b>	2.8/10.5	<b>1.0/0.6</b>	<b>1.0/0.5</b>	1.2/0.8	<b>1.0/0.6</b>
high-slow-r	0.071/0.185	0.033/0.060	0.071/0.153	<b>0.012/0.009</b>	0.035/0.060	4.7/15.8	1.9/1.9	1.9/2.0	<b>1.4/1.1</b>	1.7/1.3
high-slow-t	0.017/0.031	0.007/0.004	<b>0.006/0.003</b>	0.009/0.005	0.007/0.004	1.0/0.6	<b>0.7/0.4</b>	<b>0.7/0.4</b>	<b>0.7/0.4</b>	<b>0.7/0.4</b>
low-fast-f	0.354/0.494	<b>0.130/0.172</b>	0.308/0.312	0.143/0.094	0.138/0.183	12.1/23.1	6.5/6.0	7.3/9.0	<b>6.2/4.7</b>	6.5/5.6
low-fast-r	0.340/0.750	0.119/0.239	0.299/0.376	<b>0.049/0.030</b>	0.122/0.174	20.1/25.4	15.2/11.5	14.4/7.7	14.4/7.8	<b>14.2/8.0</b>
low-fast-t	0.237/0.233	<b>0.058/0.028</b>	0.097/0.125	0.122/0.084	0.059/0.030	7.3/6.9	5.3/2.7	5.5/3.0	5.3/2.6	<b>5.2/2.6</b>
low-med-f	0.181/0.372	0.025/0.022	0.107/0.178	0.042/0.040	<b>0.023/0.019</b>	4.8/9.5	<b>2.2/1.5</b>	3.5/8.0	2.3/1.4	<b>2.2/1.3</b>
low-med-r	0.092/0.159	0.030/0.043	0.140/0.218	<b>0.020/0.012</b>	0.028/0.046	3.5/3.6	2.9/2.1	3.0/2.3	2.9/2.2	<b>2.8/2.5</b>
low-med-t	0.068/0.156	<b>0.026/0.014</b>	0.071/0.093	0.032/0.020	<b>0.026/0.014</b>	3.8/9.1	<b>2.5/1.5</b>	<b>2.5/1.5</b>	<b>2.5/1.5</b>	<b>2.5/1.5</b>
low-slow-f	0.052/0.111	<b>0.012/0.016</b>	<b>0.012/0.014</b>	0.024/0.026	<b>0.012/0.014</b>	2.4/8.2	1.3/0.8	<b>1.2/0.7</b>	1.3/0.7	1.3/0.9
low-slow-r	0.060/0.108	0.033/0.068	0.087/0.155	<b>0.014/0.013</b>	0.031/0.064	2.3/2.7	1.5/1.4	1.6/2.5	1.6/1.9	<b>1.3/1.3</b>
low-slow-t	0.020/0.022	<b>0.010/0.007</b>	0.021/0.066	0.015/0.013	<b>0.010/0.007</b>	1.0/0.6	<b>0.9/0.5</b>	<b>0.9/0.5</b>	<b>0.9/0.5</b>	<b>0.9/0.5</b>
med-fast-f	0.309/0.380	0.180/0.245	0.438/0.453	0.189/0.103	<b>0.117/0.131</b>	15.6/19.2	11.9/18.4	9.0/8.1	10.7/7.8	<b>8.2/6.7</b>
med-fast-r	0.248/0.620	0.318/0.478	0.430/0.457	<b>0.055/0.031</b>	0.158/0.202	17.7/17.1	18.2/14.0	15.3/8.9	15.8/8.0	<b>14.1/7.2</b>
med-fast-t	0.075/0.134	0.042/0.022	0.078/0.139	0.069/0.053	<b>0.040/0.020</b>	3.7/2.8	<b>3.3/1.8</b>	3.4/1.9	<b>3.3/1.8</b>	<b>3.3/1.8</b>
med-med-f	0.162/0.448	0.021/0.017	0.043/0.085	0.041/0.038	<b>0.019/0.013</b>	8.2/24.3	<b>2.1/1.2</b>	2.8/4.5	2.5/1.8	<b>2.1/1.1</b>
med-med-r	0.060/0.083	0.053/0.086	0.145/0.249	<b>0.023/0.013</b>	0.041/0.058	3.8/2.9	3.8/2.8	3.5/2.8	<b>3.2/2.4</b>	<b>3.2/2.2</b>
med-med-t	0.028/0.037	<b>0.012/0.006</b>	<b>0.012/0.007</b>	0.017/0.011	<b>0.012/0.006</b>	1.5/1.1	<b>1.3/0.7</b>	<b>1.3/0.7</b>	<b>1.3/0.7</b>	<b>1.3/0.7</b>
med-slow-f	0.053/0.171	0.012/0.008	0.014/0.038	0.015/0.014	<b>0.010/0.006</b>	2.2/5.9	<b>1.3/0.9</b>	<b>1.3/1.0</b>	1.4/1.1	<b>1.3/0.9</b>
med-slow-r	0.037/0.151	0.014/0.029	0.031/0.094	<b>0.009/0.008</b>	0.014/0.029	1.5/2.9	1.0/0.7	1.1/2.0	1.0/0.7	<b>0.9/0.7</b>
med-slow-t	0.031/0.118	<b>0.007/0.004</b>	0.027/0.080	0.009/0.005	<b>0.007/0.004</b>	1.3/2.4	<b>0.8/0.4</b>	<b>0.8/0.5</b>	<b>0.8/0.4</b>	<b>0.8/0.4</b>

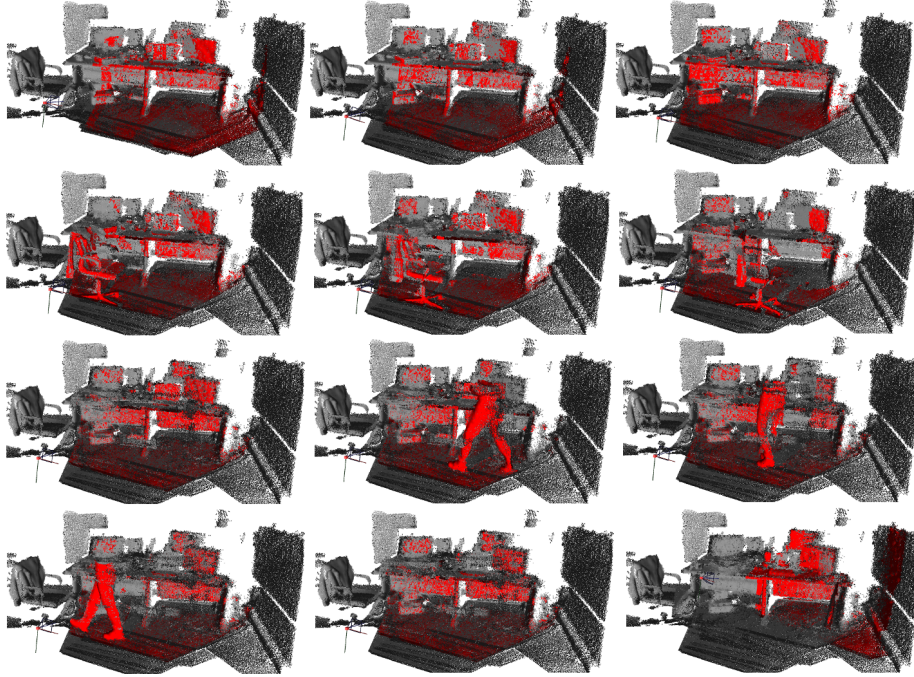


Figure 7: Example of camera tracking in a office like environment with dynamic objects. From left to right, and from top to bottom: a temporal sequence of snapshot acquired during the tracking. Red points belong to the last depth image registered. Despite both a chair and a person moved within the field of view of the camera, our algorithm is able to track the sensor pose and remove the dynamic objects.

the camera. Despite these dynamic objects in the scene, our algorithm has been able to track the pose of the sensor, and at the same time remove these elements from the point cloud. Fig. 7 illustrates a temporal sequence of snapshot acquired during the tracking. Red points belong to the last depth image registered.

### 6.3. Multiple Sensor Tracking

In this experiment we tested our tracking method on a dataset acquired in the Catacombs of Priscilla in Rome during the ROVINA project, and available at the link <http://www.rovina-project.eu/research/datasets>.

The robot was equipped with 2 Asus Xtions mounted on the front. More specifically, assume the  $x$  axis of the robot reference frame pointing forward, and the  $z$  axis going upward. The left and the right Xtions were mounted at a distance of 30 cm between each other, and with a pan angle of respectively  $\pi/6$  rad and  $-\pi/6$  rad. These kind of catacombs are composed by an underground network of very narrow corridors. While exploring the tunnels, multiple sensors are fundamental to help the robot to avoid the possibility of blindness on turns. Fig. 8 shows the result of the tracking on a part of the dataset, the

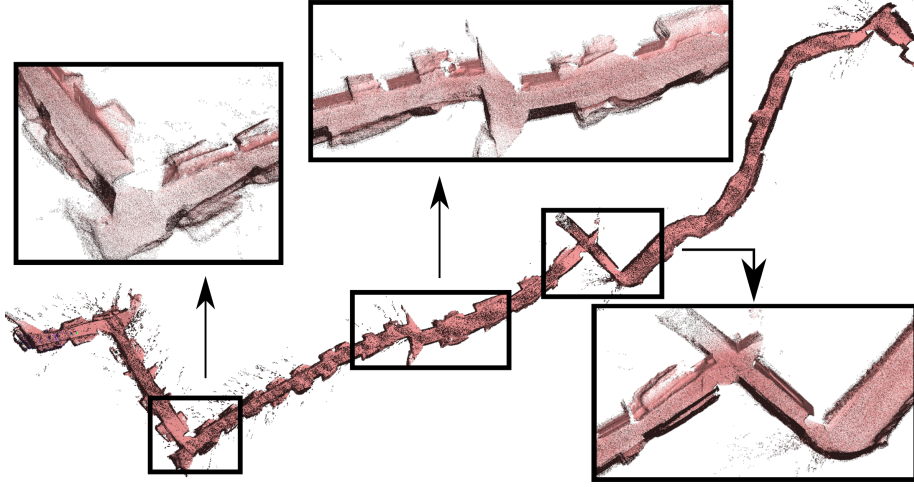


Figure 8: Multiple sensor tracking in the catacombs of Priscilla in Rome. In this specific case 2 Asus Xtions have been used.

magnifications highlight the high quality of the point clouds generated. No surface reconstruction has been applied. To give an idea of the size of the map, consider that the whole dataset has a dimension of  $\sim 100 \times 50$  meters, and the figure depicts a portion of  $\sim 35 \times 10$  meters.

#### 6.4. Complexity Analysis

In this section we provide an analysis of the computational complexity of the algorithms treated in this paper.

The surface normal computation approach plays an important role during an alignment. Indeed, methods using Singular Value Decomposition (SVD) on the neighborhood of the points have  $\mathcal{O}(k \cdot d^3)$  complexity, where  $k$  is the number of neighboring points, and  $d$  is the dimension of the matrix on which the SVD is computed. In addition to this, it must also be taken into account the complexity associated to the computation of the covariance matrix of each neighborhood. When using KD-trees, this means an additional  $\mathcal{O}(p \cdot k \cdot \log(p))$ , where  $p$  is the number of points in the cloud, plus the KD-tree generation time. Using integral images, instead, adds a  $\mathcal{O}(n \cdot m)$  complexity related to the integral image construction, with  $(n, m)$  being the dimension of the integral image itself. Once generated, the covariance matrix can be computed in constant time  $\mathcal{O}(1)$ . NICP uses integral image based normal computation through SVD. Using a cross product based method, as done in SNICP, leads to a complexity  $\mathcal{O}(k)$ , thus to a relevant computation time reduction at the cost of less accurate surface normals.

Another part that must be analyzed is the correspondence search method. By using a projective criteria the complexity is  $\mathcal{O}(n \cdot m)$ , where  $(n, m)$  is the dimension of the image where the points are projected. Using KD-trees,

instead, leads to a complexity of  $\mathcal{O}(q \cdot \log(p))$ , with  $q$  and  $p$  being respectively the number of queries in the KD-tree, and the number of points in the cloud. Additionally, as in the case of the surface normal computation, we need to add also the KD-tree construction time. Correspondences estimated using KD-trees are usually slower to compute, but in general more accurate. In our approach we use projective correspondence search.

Finally, the last part to be analyzed is the approach used for computing the relative transformation between the two clouds. The algorithms that solve this problem can be divided in two main classes: direct and iterative approaches. Direct methods like the Horn formula computes the relative translation and rotation between two cloud in one step. Unfortunately, such solutions can be applied only under the assumption that the correspondences are known. Since this is not the general case, we must use iterative methods. All iterative methods have linear complexity in the number of measurements.

## 7. Conclusions

In this paper we presented in detail a novel variant of the Iterative Closest Point (ICP) algorithm to incrementally register point clouds. Our method extends the measurement vector with surface normals information and it uses a projective criterion to find correspondences. We discussed all the relevant steps needed for the implementation of this system, and we also provided the mathematical derivations in the appendices of this paper. Additionally, we introduced a method for point cloud merging that allows to decimate the points on a cloud, while taking into account the sensor intrinsic error of the points. Also, we extended the method to handle dynamic objects and we provided a variant that can process data with a rate of  $\sim 60\text{Hz}$ , on a single CPU core. Experiments on a large standard benchmarking dataset show that our algorithm offers better results, and higher robustness, with respect to other state-of-the-art methods.

## Acknowledgments

The research leading to these results has been partly funded from the European Commission under FP7-600890-ROVINA.

## Appendix A. Least-Squares Problem

The least-squares method is a standard technique used to compute approximated solutions of over determined systems (i.e. sets of equations where there are more equations than unknowns variables). In particular, the term “Least square” means that the final solution minimizes the sum of the squares of each error term. With this in mind, it is possible to define the error  $\mathbf{e}_i$  as the difference between the current measurement coming from the sensor, and a predicted one:

$$\mathbf{e}_i(\mathbf{x}) = \mathbf{z}'_i - \mathbf{z}_i = \mathbf{h}_i(\mathbf{x}) - \mathbf{z}_i \quad (36)$$

where  $\mathbf{h}_i$  is the function that maps the state vector  $\mathbf{x}$  to the predicted measurement  $\mathbf{z}'_i$ . Note that, in general,  $\mathbf{h}_i(\mathbf{x})$  is a non-linear function of the state. However, it is possible to approximate it in the neighborhood of a linearization point  $\check{\mathbf{x}}$  using its Taylor expansion:

$$\mathbf{h}_i(\check{\mathbf{x}} + \Delta\mathbf{x}) \simeq \mathbf{h}_i(\check{\mathbf{x}}) + \left. \frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\check{\mathbf{x}}} \cdot \Delta\mathbf{x} \quad (37)$$

where we call *Jacobian matrix*  $\mathbf{J}_i$ :

$$\mathbf{J}_i = \left. \frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\check{\mathbf{x}}} \quad (38)$$

Now, assuming the error to be zero mean and normally distributed with an *information matrix*  $\Omega_i$ , the squared error of a measurement depends only on the state and it is a scalar value:

$$e_i(\mathbf{x}) = \mathbf{e}_i(\mathbf{x})^T \Omega_i \mathbf{e}_i(\mathbf{x}). \quad (39)$$

Given a set of  $N$  measurements, the objective is to find the state  $\mathbf{x}^*$  that minimizes the error of all measurements:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ \sum_i^N \mathbf{e}_i(\mathbf{x})^T \Omega_i \mathbf{e}_i(\mathbf{x}) \right\} \quad (40)$$

Let be

$$F(\mathbf{x}) = \sum_{i=1}^N \mathbf{e}_i(\mathbf{x})^T \Omega_i \mathbf{e}_i(\mathbf{x}) = \sum_{i=1}^N e_i(\mathbf{x}), \quad (41)$$

and assuming that an acceptable initial guess  $\check{\mathbf{x}}$  of the optimum solution is known, it is possible to rewrite one of the summands in the previous equation by using the Taylor approximation in Eq. 37:

$$\begin{aligned} e_i(\check{\mathbf{x}} + \Delta\mathbf{x}) &= (\mathbf{h}_i(\check{\mathbf{x}} + \Delta\mathbf{x}) - \mathbf{z}_i)^T \Omega_i (\mathbf{h}_i(\check{\mathbf{x}} + \Delta\mathbf{x}) - \mathbf{z}_i) \\ &\simeq (\mathbf{J}_i \Delta\mathbf{x} + \underbrace{\mathbf{h}_i(\check{\mathbf{x}}) - \mathbf{z}_i}_{\mathbf{e}_i})^T \Omega_i (\mathbf{J}_i \Delta\mathbf{x} + \underbrace{\mathbf{h}_i(\check{\mathbf{x}}) - \mathbf{z}_i}_{\mathbf{e}_i}) \\ &= (\mathbf{J}_i \Delta\mathbf{x} + \mathbf{e}_i)^T \Omega_i (\mathbf{J}_i \Delta\mathbf{x} + \mathbf{e}_i) \\ &= \Delta\mathbf{x}^T \mathbf{J}_i^T \Omega_i \mathbf{J}_i \Delta\mathbf{x} + 2\mathbf{e}_i^T \Omega_i \mathbf{J}_i \Delta\mathbf{x} + \mathbf{e}_i^T \Omega_i \mathbf{e}_i \\ &= \Delta\mathbf{x}^T \mathbf{H}_i \Delta\mathbf{x} + 2\mathbf{b}_i \Delta\mathbf{x} + \mathbf{e}_i^T \Omega_i \mathbf{e}_i, \end{aligned} \quad (42)$$

where  $\mathbf{H}_i = \mathbf{J}_i^T \Omega_i \mathbf{J}_i$  and  $\mathbf{b}_i = \mathbf{e}_i^T \Omega_i \mathbf{J}_i$ .

Substituting Eq. 42 in Eq. 41 we obtain:

$$\begin{aligned} F(\check{\mathbf{x}} + \Delta\mathbf{x}) &\simeq \sum_{i=1}^N \Delta\mathbf{x}^T \mathbf{H}_i \Delta\mathbf{x} + 2\mathbf{b}_i \Delta\mathbf{x} + \mathbf{e}_i^T \Omega_i \mathbf{e}_i \\ &= \Delta\mathbf{x}^T \left[ \sum_{i=1}^N \mathbf{H}_i \right] \Delta\mathbf{x} + 2 \left[ \sum_{i=1}^N \mathbf{b}_i \right] \Delta\mathbf{x} + \left[ \sum_{i=1}^N \mathbf{e}_i^T \Omega_i \mathbf{e}_i \right] \\ &= \Delta\mathbf{x}^T \mathbf{H} \Delta\mathbf{x} + 2\mathbf{b} \Delta\mathbf{x} + c, \end{aligned} \quad (43)$$

with  $c = \sum_{i=1}^N \mathbf{e}_i^T \Omega_i \mathbf{e}_i$ .

The last equation is the expression of the objective function  $F(\mathbf{x})$  under a linear approximation of  $\mathbf{h}_i(\mathbf{x})$ , in the neighborhood of the initial guess  $\check{\mathbf{x}}$ . In other words, if the initial estimate is fixed, the value of the function can be approximated by a quadratic form with respect to the increments  $\Delta\mathbf{x}$ . At this point, it is possible to minimize this quadratic equation and therefore compute the optimal increment  $\Delta\mathbf{x}^*$  that applied to the current estimate leads to an improved solution  $\mathbf{x}^*$ :

$$\mathbf{x}^* = \check{\mathbf{x}} + \Delta\mathbf{x}^*. \quad (44)$$

The optimal increment  $\Delta\mathbf{x}^*$  is computed by imposing the derivative of the function  $F(\check{\mathbf{x}} + \Delta\mathbf{x})$  (Eq. 43) to be equal to zero, and then solve the corresponding equation:

$$\frac{\partial(\Delta\mathbf{x}^T \mathbf{H} \Delta\mathbf{x} + 2\mathbf{b} \Delta\mathbf{x} + c)}{\partial \Delta\mathbf{x}} = 2\mathbf{H} \Delta\mathbf{x} + 2\mathbf{b} = 0. \quad (45)$$

This leads to find the solution of the following linear system:

$$\mathbf{H} \Delta\mathbf{x}^* = -\mathbf{b}, \quad (46)$$

where the matrix  $\mathbf{H}$  is commonly known as *Hessian matrix*.

If the model function would be linear, the solution would be found in just one step. Since, as said before, this is not the general case, it is necessary to iterate the procedure until an acceptable solution is found. Several methods that solve this problem exist, among these we recall the Gauss-Newton and the Levenberg-Marquardt minimization algorithms.

## Appendix B. Quaternion Based Rotation Matrix Derivative

Consider a 3D rotation defined through a quaternion  $\mathbf{q} = (q_x, q_y, q_z, q_w)^T$ , then we can also define the associated rotation matrix  $\mathbf{R}(\mathbf{q})$  as a function of the quaternion's parameters:

$$\mathbf{R}(\mathbf{q}) = \begin{pmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_x q_y - 2q_z q_w & 2q_x q_z + 2q_y q_w \\ 2q_x q_y + 2q_z q_w & 1 - 2q_x^2 - 2q_z^2 & 2q_y q_z - 2q_x q_w \\ 2q_x q_z - 2q_y q_w & 2q_y q_z + 2q_x q_w & 1 - 2q_x^2 - 2q_y^2 \end{pmatrix} \quad (47)$$

Now, assuming that  $\mathbf{q}$  is *normalized*, that means  $|\mathbf{q}| = 1$ , we can rewrite the previous matrix as follow:

$$\mathbf{R}(\mathbf{q}) = \begin{pmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_x q_y - 2q_z \hat{q}_w & 2q_x q_z + 2q_y \hat{q}_w \\ 2q_x q_y + 2q_z \hat{q}_w & 1 - 2q_x^2 - 2q_z^2 & 2q_y q_z - 2q_x \hat{q}_w \\ 2q_x q_z - 2q_y \hat{q}_w & 2q_y q_z + 2q_x \hat{q}_w & 1 - 2q_x^2 - 2q_y^2 \end{pmatrix} \quad (48)$$

where  $\hat{q}_w = \sqrt{1 - q_x^2 - q_y^2 - q_z^2}$ .

The partial derivatives of the rotation matrix  $\mathbf{R}(\mathbf{q})$  (Eq. 48), evaluated in  $\mathbf{q} = (0, 0, 0, 1)^T = \mathbf{0}$ , have the following form:

$$\left. \frac{\partial \mathbf{R}(\mathbf{q})}{\partial q_x} \right|_{\mathbf{q}=\mathbf{0}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -2 \\ 0 & 2 & 0 \end{pmatrix} = \mathbf{S}_x \quad (49)$$

$$\left. \frac{\partial \mathbf{R}(\mathbf{q})}{\partial q_y} \right|_{\mathbf{q}=\mathbf{0}} = \begin{pmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \\ -2 & 0 & 0 \end{pmatrix} = \mathbf{S}_y \quad (50)$$

$$\left. \frac{\partial \mathbf{R}(\mathbf{q})}{\partial q_z} \right|_{\mathbf{q}=\mathbf{0}} = \begin{pmatrix} 0 & -2 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \mathbf{S}_z. \quad (51)$$

Finally, the derivative of the rotation of a vector  $\mathbf{v} = (v_x, v_y, v_z)^T$ , in  $\mathbf{q} = (0, 0, 0, 1)^T = \mathbf{0}$ , can be written as:

$$\mathbf{S}(\mathbf{v}) = (\mathbf{S}_x \cdot \mathbf{v} | \mathbf{S}_y \cdot \mathbf{v} | \mathbf{S}_z \cdot \mathbf{v}) = \begin{pmatrix} 0 & 2v_z & -2v_y \\ -2v_z & 0 & 2v_x \\ 2v_y & -2v_x & 0 \end{pmatrix} = -2[\mathbf{v}]_{\times}. \quad (52)$$

### Appendix C. Jacobian Derivation

Defining a generic point with normal as a 6D vector  $\mathbf{p} = (\mathbf{c}^T, \mathbf{n}^T)^T = (c_x, c_y, c_z, n_x, n_y, n_z)^T$  composed by its Cartesian coordinates  $\mathbf{c} = (c_x, c_y, c_z)^T$  and the associated surface normal components  $\mathbf{n} = (n_x, n_y, n_z)^T$ , and recalling the definition of the  $\oplus$  operator described in Eq. 22, the perturbation of the error between two corresponding points  $\mathbf{p}$  and  $\mathbf{p}'$  is:

$$\mathbf{e}(\mathbf{x} \oplus \Delta \mathbf{x}) = \mathbf{e}(\mathbf{t} \oplus \Delta \mathbf{t}, \mathbf{R} \oplus \mathbf{R}(\Delta \mathbf{q})) = \begin{pmatrix} \mathbf{R}(\Delta \mathbf{q}) \cdot \hat{\mathbf{c}} + \Delta \mathbf{t} \\ \mathbf{R}(\Delta \mathbf{q}) \cdot \hat{\mathbf{n}} \end{pmatrix} - \begin{pmatrix} \mathbf{c}' \\ \mathbf{n}' \end{pmatrix} \quad (53)$$

with  $\hat{\mathbf{p}} = \mathbf{T}(\mathbf{x}) \oplus \mathbf{p}$ .

The  $6 \times 6$  *Jacobian matrix*  $\mathbf{J}$  is computed deriving the previous equation with respect to  $\Delta \mathbf{t} = (\Delta t_x, \Delta t_y, \Delta t_z)^T$  and  $\Delta \mathbf{q} = (\Delta q_x, \Delta q_y, \Delta q_z)^T$ , and then evaluating the derivative in  $(\Delta \mathbf{t} = 0, \Delta \mathbf{q} = 0)$ :

$$\begin{aligned} \mathbf{J} &= \left( \left. \frac{\partial \mathbf{e}(\mathbf{t} + \Delta \mathbf{t}, \mathbf{R} + \mathbf{R}(\Delta \mathbf{q}))}{\partial \Delta \mathbf{t}} \right| \left. \frac{\partial \mathbf{e}(\mathbf{t} + \Delta \mathbf{t}, \mathbf{R} + \mathbf{R}(\Delta \mathbf{q}))}{\partial \Delta \mathbf{q}} \right) \right) \Big|_{\Delta \mathbf{t}=\mathbf{0}, \Delta \mathbf{q}=\mathbf{0}} \\ &= \begin{pmatrix} \mathbf{I}_{3 \times 3} & \frac{\partial \mathbf{R}(\Delta \mathbf{q})}{\partial \Delta q_x} \cdot \hat{\mathbf{c}} & \frac{\partial \mathbf{R}(\Delta \mathbf{q})}{\partial \Delta q_y} \cdot \hat{\mathbf{c}} & \frac{\partial \mathbf{R}(\Delta \mathbf{q})}{\partial \Delta q_z} \cdot \hat{\mathbf{c}} \\ \mathbf{0}_{3 \times 3} & \frac{\partial \mathbf{R}(\Delta \mathbf{q})}{\partial \Delta q_x} \cdot \hat{\mathbf{n}} & \frac{\partial \mathbf{R}(\Delta \mathbf{q})}{\partial \Delta q_y} \cdot \hat{\mathbf{n}} & \frac{\partial \mathbf{R}(\Delta \mathbf{q})}{\partial \Delta q_z} \cdot \hat{\mathbf{n}} \end{pmatrix} \Big|_{\Delta \mathbf{t}=\mathbf{0}, \Delta \mathbf{q}=\mathbf{0}} \\ &= \begin{pmatrix} \mathbf{I}_{3 \times 3} & -2[\hat{\mathbf{c}}]_{\times} \\ \mathbf{0}_{3 \times 3} & -2[\hat{\mathbf{n}}]_{\times} \end{pmatrix} \end{aligned} \quad (54)$$

where  $[\mathbf{v}]_{\times}$  represents the cross product matrix as defined in Eq. 52.

The reader might notice that the left block of the Jacobian matrix has a  $3 \times 3$  matrix of zeros in the bottom, this is because the normal, unlike the point coordinates, is not translated.

## References

- [1] J. Serafin, G. Grisetti, Nicp: Dense normal based point cloud registration, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Hamburg, Germany, 2015, pp. 742–749.
- [2] J. Serafin, G. Grisetti, Using augmented measurements to improve the convergence of ICP, in: Simulation, Modeling, and Programming for Autonomous Robots (SIMPAN), Springer, 2014.
- [3] P. J. Besl, N. D. McKay, A method for registration of 3-D shapes, IEEE Transactions on Pattern Analysis and Machine Intelligence.
- [4] A. V. Segal, D. Haehnel, S. Thrun, Generalized-ICP, in: Proc. of Robotics: Science and Systems (RSS), 2009.
- [5] M. Magnusson, T. Duckett, A. J. Lilienthal, Scan registration for autonomous mining vehicles using 3D-NDT, Journal on Field Robotics.
- [6] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, A. Fitzgibbon, KinectFusion: Real-time dense surface mapping and tracking, in: Proc. of the Int. Symposium on Mixed and Augmented Reality (ISMAR), 2011.
- [7] F. Pomerleau, F. Colas, R. Siegwart, S. Magnenat, Comparing ICP variants on real-world data sets.
- [8] J. Chen, G. Medioni, Object modeling by registration of multiple range images, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 1991.
- [9] G. Blais, M. D. Levine, Registering multiview range data to create 3D computer objects, Pattern Analysis and Machine Intelligence, IEEE Transactions on 17 (8) (1995) 820–824.
- [10] B. K. Horn, H. M. Hilden, S. Negahdaripour, Closed-form solution of absolute orientation using orthonormal matrices, Journal of the Optical Society of America.
- [11] F. Steinbrücker, J. Sturm, D. Cremers, Real-time visual odometry from dense rgb-d images, in: Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on, IEEE, 2011, pp. 719–722.
- [12] R. Smith, M. Self, P. Cheeseman, Estimating uncertain spatial relationships in robotics, Autonomous Robot Vehicles.
- [13] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, Fastslam: A factored solution to the simultaneous localization and mapping problem, in: In Proceedings of the AAAI National Conference on Artificial Intelligence, 2002.



- [14] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges, in: In Proc. of the Int. Conf. on Artificial Intelligence (IJCAI, 2003.
- [15] R. Triebel, P. Pfaff, W. Burgard, Multi-level surface maps for outdoor terrain mapping and loop closing, in: 2006 IEEE/RSJ international conference on intelligent robots and systems, IEEE, 2006, pp. 2276–2282.
- [16] D. Hähnel, W. Burgard, D. Fox, S. Thrun, An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements, in: Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on, 2003.
- [17] G. Grisetti, C. Stachniss, W. Burgard, Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling, in: Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, 2005.
- [18] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, W. Burgard, Octomap: An efficient probabilistic 3d mapping framework based on octrees, *Autonomous Robots* 34 (3) (2013) 189–206.
- [19] H. Durrant-Whyte, D. Rye, E. Nebot, Localization of autonomous guided vehicles, in: Robotics Research, 1996.
- [20] R. C. Smith, P. Cheeseman, On the representation and estimation of spatial uncertainty, *The international journal of Robotics Research*.
- [21] J. J. Leonard, H. F. Durrant-Whyte, Simultaneous map building and localization for an autonomous mobile robot, in: Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on, 1991.
- [22] J. A. Castellanos, J. Montiel, J. Neira, J. D. Tardós, The spmap: A probabilistic framework for simultaneous localization and map building, *IEEE Transactions on Robotics and Automation* 15 (5) (1999) 948–952.
- [23] M. A. Paskin, Thin junction tree filters for simultaneous localization and mapping, in: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03), 2003.
- [24] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, H. Durrant-Whyte, Simultaneous localization and mapping with sparse extended information filters, *The International Journal of Robotics Research*.
- [25] R. M. Eustice, H. Singh, J. J. Leonard, Exactly sparse delayed-state filters, in: Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, 2005.

- [26] K. P. Murphy, et al., Bayesian map learning in dynamic environments., in: NIPS, 1999.
- [27] G. Grisetti, C. Stachniss, W. Burgard, Improved techniques for grid mapping with rao-blackwellized particle filters, *IEEE transactions on Robotics* 23 (1) (2007) 34–46.
- [28] F. Lu, E. Milios, Globally consistent range scan alignment for environment mapping, *Autonomous robots*.
- [29] E. Olson, J. Leonard, S. Teller, Fast iterative alignment of pose graphs with poor initial estimates, in: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, 2006.
- [30] F. Dellaert, M. Kaess, Square root sam: Simultaneous localization and mapping via square root information smoothing, *Int. J. Rob. Res.*
- [31] J.-S. Gutmann, K. Konolige, Incremental mapping of large cyclic environments, in: *Computational Intelligence in Robotics and Automation, 1999. CIRA'99. Proceedings. 1999 IEEE International Symposium on*, 1999.
- [32] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, S. Teller, An atlas framework for scalable mapping, in: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, 2003.
- [33] C. Estrada, J. Neira, J. D. Tardós, Hierarchical slam: real-time accurate mapping of large environments, *Robotics, IEEE Transactions on*.
- [34] E. B. Olson, Robust and efficient robotic mapping.
- [35] J. Neira, J. D. Tardós, Data association in stochastic mapping using the joint compatibility test, *IEEE Transactions on robotics and automation* 17 (6) (2001) 890–897.
- [36] A. Nüchter, K. Lingemann, J. Hertzberg, H. Surmann, 6d slam with approximate data association, in: *Advanced Robotics, 2005. ICAR'05. Proceedings., 12th International Conference on*, 2005.
- [37] A. Ranganathan, M. Kaess, F. Dellaert, Loopy SAM, in: *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, 2007.
- [38] M. Kaess, A. Ranganathan, F. Dellaert, isam: Fast incremental smoothing and mapping with efficient data association, in: *Robotics and Automation, 2007 IEEE International Conference on*, 2007.
- [39] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, J. McDonald, Kintinuous: Spatially extended KinectFusion, in: *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia, 2012.

- [40] R. A. Newcombe, D. Fox, S. M. Seitz, Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 343–352.
- [41] M. Labbé, F. Michaud, Online global loop closure detection for large-scale multi-session graph-based slam, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2014, pp. 2661–2666.
- [42] R. Mur-Artal, J. Montiel, J. D. Tardós, Orb-slam: a versatile and accurate monocular slam system, IEEE Transactions on Robotics 31 (5) (2015) 1147–1163.
- [43] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, A. J. Davison, Elasticfusion: Dense slam without a pose graph, Proc. Robotics: Science and Systems, Rome, Italy.
- [44] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, N. Navab, Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images, in: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE, 2012, pp. 2684–2689.
- [45] M. Pauly, M. Gross, L. P. Kobbelt, Efficient simplification of point-sampled surfaces, in: Proceedings of the conference on Visualization’02, IEEE Computer Society, 2002, pp. 163–170.
- [46] F. Pomerleau, S. Magnenat, F. Colas, M. Liu, R. Siegwart, Tracking a depth camera: Parameter exploration for fast ICP, in: Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, IEEE, 2011, pp. 3824–3829.
- [47] R. B. Rusu, S. Cousins, 3d is here: Point cloud library (pcl), in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), 2011.
- [48] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, Ros: an open-source robot operating system, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA) Workshop on Open Source Software, 2009.
- [49] J. Sturm, N. Engelhard, F. Endres, W. Burgard, D. Cremers, A benchmark for the evaluation of rgb-d slam systems, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2012.