SCUOLA DI DOTTORATO
UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

Dipartimento di / Department of

Informatica, Sistemistica e Comunicazione

Dottorato di Ricerca in / PhD program Informatica Ciclo / Cycle XXIX

Curriculum in (se presente / if it is) _____

Robust Point Clouds Registration

Cognome / Surname Fontana Nome / Name Simone

Matricola / Registration number 712638

Tutore / Tutor: dott. Giuseppe Vizzari

Cotutore / Co-tutor: _____
(se presente / if there is one)

Supervisor: prof. Domenico G. Sorrenti
(se presente / if there is one)

Coordinatore / Coordinator: prof. Stefania Bandini

ANNO ACCADEMICO / ACADEMIC YEAR 2015/2016

Robust Point Clouds Registration

Simone Fontana

August 2017

Abstract

Point clouds registration is a very well studied problem, with many different and efficient solutions. Nevertheless, the approaches in the literature rely heavily on a good initialization and on a good set of parameters. These approaches could be roughly divided into two categories: those based on features and the so-called closest-point-based. The first category aims at aligning two point clouds by first detecting some salient points, the keypoints, and calculating their descriptors so that they can be compared, in the same way it is usually done with 2D images. On the other hand, the latter category approximates correspondences by iteratively choosing the closest point, without the need for any kind of feature. The most important algorithm in this category is Iterative Closest Point (ICP). Most other algorithms are variants of ICP, and so is one of the proposed approaches.

In this work we introduce two novel solutions to point clouds registration.

The first one is a variant of ICP, with a different data association policy, derived from a probabilistic model. The experiments show that it is very effective at aligning a sparse point cloud with a dense one, one of the issues we faced in this work. On the other hand, it showed very good results also on standard alignment problems, often better than those of other popular state of the art algorithms. We show that, for the most common approaches, the quality of the result is heavily dependent on some parameters that, thus, need to be carefully calibrated before the algorithms could be used in real applications. Moreover, a new calibration is usually required when facing a new scenario. For this reasons we propose this innovative technique, that aims, besides at being capable of aligning two generic point clouds, independently from their density, at being more robust w.r.t. wrong parameter sets.

The second technique we developed is a global point cloud registration algorithm. ICP-like techniques requires, in order to converge to the right solution, an initial estimate of the transformation between the two point clouds. Without a proper initial guess, the algorithm would remain stuck in a local minima. On the other hand, feature-based techniques do not require any initial estimate but are not applicable to sparse point clouds, because

they do not contain enough information to extract meaningful descriptors. The approach we developed combines the advantages of both approaches. It is based on a soft-computing technique, Particle Swarm Optimization, that is known for being able to escape from local optima. The combination of the two new techniques is an algorithm capable of aligning any kind of point cloud, without the need of any initial estimate of the transformation.

Contents

Chapter 1

Introduction

A point cloud is simply a set of 3D points, *i.e.*, a set of points in the space. These points could be used to define the shape of a single object or to represent a big environment, such as a building. For this reason, point clouds are heavily used in robotics. For example, they can be used as a map for a localization system or they can be analysed to extract semantic information.

Point clouds are becoming increasingly popular nowadays. While in the past the production of point clouds was reserved to very expensive robotic platforms or measurement systems, today also cheaper alternatives exist. The forefather of these cheaper sensors is the Microsoft Kinect, the first cheap and easily available RGB-D sensor. RGB-D stands for RGB-Depth, that is, the sensor is capable of producing together both a standard RGB image and an aligned depth image. From the depth image, given the calibration data of the sensor, we are able to produce a point cloud.

However, when dealing with point clouds, a very common problem arises. Suppose that a robot needs to build a representation of an environment; typically a single point cloud would not be enough, given the limited field of view and usable range of the sensors. Therefore, the robot would have to produce different point clouds from different points of view and then fuse them together. The different point clouds, indeed, will have each its own reference frame. Thus, there is no naïve way of aligning them, because no global reference frame exists. This is the so-called point clouds registration problem: we want to find the transformation that best aligns a point cloud with another one, *i.e.*, we want to express the coordinates of the points in one point cloud w.r.t. to the reference system of the other one. Of course, the two clouds must have some degree of overlap.

One simple solution is to use some kind of external sensor that returns the displacement between the two poses where the clouds have been produced. This could be accomplished, for example, using odometry measurements,

when using a wheeled robotic platform. As an alternative (or in addition) also an inertial measurement unit could be used. However, usually these solutions are not precise enough and, thus, have to be integrated with other techniques. Moreover there are situations where no other measurement, besides the point clouds, is available, *e.g.*, when fusing two maps produced by two different robots at different times.

Point clouds registration is a very well studied problem, with many different and efficient solutions. The approaches in the literature could be roughly divided into two main categories: those based on features, and the so-called closest-point based solutions. The first category aims at aligning two point clouds by first detecting some salient points, the keypoints, and calculating some kind of descriptors. The descriptors of the keypoints can then be matched with those of another point cloud, so to find correspondences in a way similar to what is done with feature matching for 2D images, *e.g.*, in mosaicing for building panorama images.

This class of techniques usually does not require any other information, besides the point clouds. For this reason it is said to be capable of aligning point clouds globally: no matter the initial displacement, the algorithms should be able to find a proper alignment. In practice, however, the results usually are not really accurate. Therefore, feature-based registration is usually used to produce a rough alignment, that will be refined later using other kinds of techniques.

On the contrary, closest-point based approaches do not have any explicit feature matching step. They simply greedily approximate the correspondences by iteratively associating to each point in a point cloud the closest point in the other one. The most important algorithm in this category is Iterative Closest Point (ICP). Most other algorithms are variants of ICP, and so is also one of the approaches proposed in this work. Given its closest-point based association policy, this class of algorithms is usually able to converge to a meaningful solution only when the two point clouds are already quite close to the final alignment: they need to be already roughly aligned. However, given the appropriate initial conditions, ICP-like algorithms usually produce very precise results. Therefore, the two categories are not mutually exclusive: feature based registration could be used to produce a rough alignment that then is refined using a variant of ICP.

Our approach is derived directly from a probabilistic model and is basically a substitute for ICP, with better performances in terms of quality of the result and with less sensitivity to parameters. Notably, the result of our technique is much more independent to a fine parameters' tuning. One of the problem of ICP and its variants is that they depend heavily on one or more parameters. We will show that the results can be heavily affected by just

very small changes in these parameters. This is a very undesirable behaviour for an algorithm that will be used on robotics platforms: if a parameter has to be very finely tuned in order for the algorithm to work, it means that it would have to be re-calibrated for every different scenario. Of course, if a recalibration is needed even for small changes and, moreover, there is no automatic way of doing this calibration, the robot cannot really be called autonomous. On the contrary, a low sensibility to the parameters is a very desirable behaviour and is what we achieved with the proposed algorithm.

The execution time of our proposal, w.r.t. ICP, increases greatly, so it is not suitable for real-time applications. This is not a big problem, considering that there are lot of applications where point clouds registration is performed off-line.

Moreover, closest-point based algorithms usually need a good initial guess in order to converge to the right solution. This initial guess could be provided using some kind of feature-based technique, but this is not always possible. For example, when dealing with sparse point clouds, *i.e.*, clouds with very few points per unit of volume, feature-based techniques usually are not useful. This is due to the fact that 3D feature descriptors usually require the normal of the surface on which the keypoint lies. Very sparse point clouds do not represent a surface in a way enough informative. Practically, calculating the normal to a point on a surface requires a set of neighbouring points lying on the same surface. This neighbourhood, in a sparse point cloud, could easily be empty! As it will be clear in the following chapters, one of the objective of this work is to deal with any kind of point clouds. This includes registering a very sparse point cloud with a *standard* dense point cloud.

In which situations could be useful to align a sparse point cloud with a dense one? Suppose, for example, that a robot produced a map of an environment with a sensor producing dense point clouds. Moreover, suppose that this map is then used for the localization of another robot, using a different kind of sensor, thus maybe producing sparser point clouds. This kind of problem is an example of dense-sparse registration that state-of-the-art algorithms often struggle to solve and that cannot take advantage of feature-based registration. We will show how our probabilistic approach can be successfully applied to this scenario too.

However, the proposed probabilistic approach, similarly to ICP, is still a fine-registration technique. For this reason we developed also an algorithm aimed at roughly aligning two point clouds, without any assumption on their initial displacement or their densities and that, thus, does not use feature descriptors. It can be applied literately ot any possible kind of point cloud, as it requires only the coordinates of the points. This novel algorithm finds the best alignment of two point clouds solving an optimization problem using

a soft-computing technique: Particle Swarm Optimization. We will see that defining the *best alignment* is per-se a very hard challenge. Since the real point associations between the two point clouds are unknown (and, actually, could even not exist), defining a metric that quantifies the quality of an alignment was not easy and required many experiments. Indeed, this metric has to work globally, *i.e.*, independently from the initial displacement of the point clouds, the minimum value of the metric must correspond to the best alignment. Moreover, also the *shape* of the function is important: if the metric has many strong local minima, the optimization algorithm will struggle to find the global minimum. This two requisites should hold for any kind of point clouds. We show how our PSO-based algorithm is capable of aligning two generic point clouds, without the need of any initial guess.

Chapter 2 is a summary of the state-of-the-art in point clouds registration, covering both feature-based registration and, more deeply, closest-point based registration algorithms. Chapter 3 describes various datasets used to perform the experiments presented in this work. Chapter 4 introduces our probabilistic point clouds registration algorithm and shows its performances on many kind of datasets. Chapter 5 expands the approach presented in the preceding chapter, introducing an automatic termination criterion for the algorithm. Chapter 6 analyses the proposed probabilistic approach and the most common point clouds registration algorithms, showing their sensitivity to various parameters. We show that our algorithm has a very low parameters' sensitivity. Chapter 7 describes our PSO-based global point clouds registration algorithm and shows its very good performances on various datasets.

Chapter 2

Related Work

2.1 Point Clouds Registration

What is a Point Cloud?

A point cloud is simply a set of points. However, since many different kind of sensor can produce a point cloud, they can differ on what they contain, besides 3D coordinates of the set of points.

A very popular class of sensors, capable of producing point clouds, are the LiDARs. As a simplification, LiDARs are composed of an emitter, that emits a laser ray, and a receiver, that receives the light emitted and reflected by obstacles in the space. The distance of an object from the sensor can be measured using the return time of the emitted light. In some cases, a point cloud produced with a LiDAR, depending to the sensor used, could contain also the intensity of the reflected light, besides the coordinates in the scene.

Another important class of sensors, capable of producing point clouds, are the RGB-D sensors. A very popular members of this class is the Microsoft Kinect, [1]. The peculiarity of this kind of sensors is that they can produce both an RGB image and a depth image. Therefore, the produced point clouds will contain also color (RGB) information, besides 3D points. While there exist other RGB-D sensors, the two aforementioned have acquired high importance recently, mainly because of their very low price, compared to other sensors. Therefore, they made producing point clouds very easy and cheap, and boosted the research in the field of point clouds processing and 3D reconstruction, [2].

Lastly, a point cloud can be produced also using a stereo pair or even a single moving camera, using structure from motion or photogrammetry techniques, [3].

The density of the point clouds is a crucial characteristic. With *density*

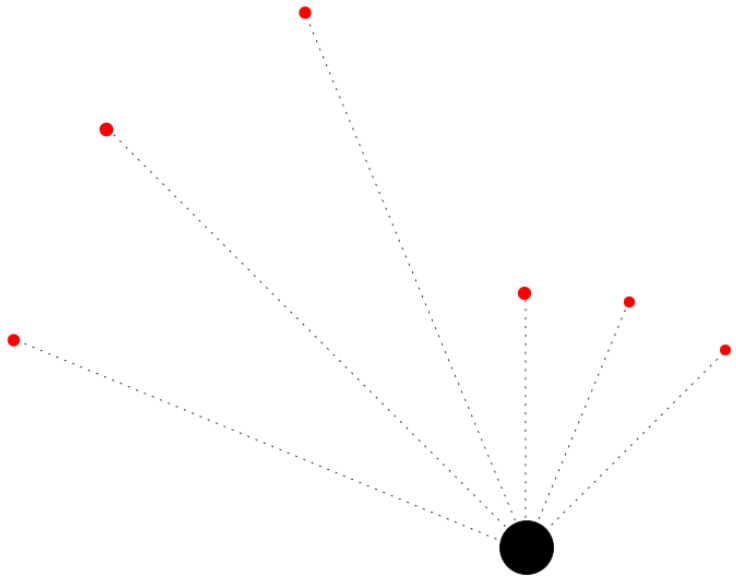


Figure 2.1: A scheme of how a LiDAR works. Points more distant from the sensor are less dense, even though their angular distance is the same.

we refer to the distance between closest points in a cloud. A point cloud could be dense, when the points are very close, or sparse instead. These characteristics make sense mainly when comparing two point cloud: we could say, for example, that a point cloud is denser than another when it contains more points per unit of volume. Moreover, it could happen that the density of a point cloud is not constant, but decreases when moving farther from the sensor. This is usually true, for example, for LiDAR sensors, that measure points at a fixed angular resolution. Therefore, points more distant from the sensor will be more spaced, Figure 2.1.

Dense point clouds are produced, for example, by a Kinect, that, in its first version, has a field of view of 57° horizontally and 43° vertically, with a maximum of 307200 measured points. As a comparison, a Velodyne VLP-16, a LiDAR sensor, has an horizontal field of view of 360°, and 30° vertically, with 54000 measured points. Therefore, it has a much wider field of view, but with less points and thus, the produced point clouds are less dense. Figure 2.2 is an example of these differences.

There are situations where only a sparse point cloud can be obtained. For example visual-inertial systems usually produce a map of features, each with its own position in the space. Usually this set of features is not very dense, with the density depending on the particular scene, the texture of the scene and the feature detector used. Nevertheless, it can be treated as a sparse point cloud, in order, for example, to fuse it with other point clouds. The

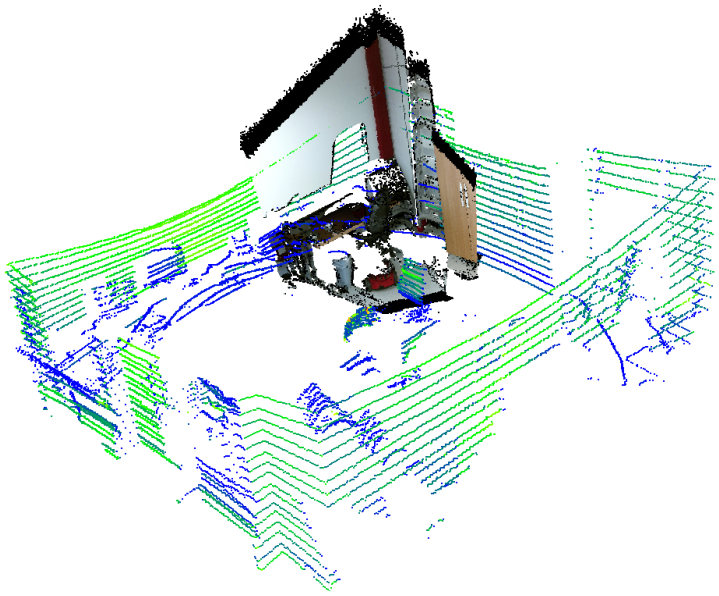


Figure 2.2: Two point clouds: one very dense, produced with a Kinect 2, the other, sparser, produced with a Velodyne VLP-16.

other point cloud could be another feature map from the same or a similar sensor, or a completely different kind of point cloud, such as a dense point cloud produced with an RGB-D sensor.

Sparse point clouds have some peculiarities. Most important, usually calculating accurate surface descriptors is impossible. Surface descriptors are used to locally describe the surfaces represented in a point cloud. Almost all the descriptors in the literature use the Surface Normal, that is the normal to the surface in the neighbourhood of a point, [4, 5, 6, 7, 8]. To calculate it, the surface on which the point lies is approximated using the position of a number of points in its neighbourhood. This is not usually possible in a sparse point cloud, because the neighbourhood of a point often is simply empty! That is we do not have enough points nearby to accurately describe the surface. This is a very important drawback, because it makes registering sparse point clouds much harder, since many global registration techniques use some kind of feature descriptor.

What is Point Clouds Registration?

Suppose that we have two different point clouds that have at least a part in common, *i.e.*, they partially overlap. These two point clouds could have been generated from the same moving sensor at different times, or they could have

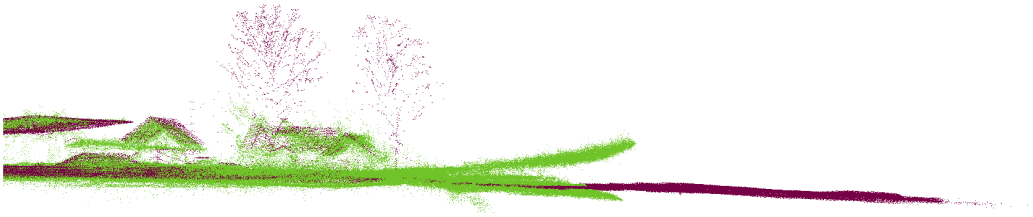


Figure 2.3: A detail from the Linkoping dataset, [13]. One of the point clouds is heavily distorted and thus cannot be aligned perfectly.

been generated by different sensors. Their sparseness can consequently be very different. Nevertheless, the setup used to produce them is not important for our specific problem.

These point clouds have each its own local reference frame, that is, their points have coordinates expressed in two different reference frames, usually with the origin on the position of the sensor at the time of the acquisition. If we want to express all the coordinates in the same reference frame, we have to solve the so-called *Point Clouds Registration* problem, *i.e.*, we have to find a transformation that aligns the first point cloud (usually called the *source* point cloud), with the second (usually called the *target* point cloud). In this work we will only deal with the problem of finding a rigid transformation between two point clouds, that means find a rotation and a translation. Of course using non-rigid transformations could lead to a more precise alignment, [9], but the problem would become much harder to solve and also prone to over-fitting. The rigid transformation assumption is heavily used in the literature, [10, 11, 12], because usually leads to good results, while greatly simplifying the problem: if the two point clouds are not heavily distorted, the result is usually good enough. In Figure 2.3 there is an example of alignment that would benefit from the use of non-rigid transformations. One of the point clouds, the one produced with a camera, is heavily distorted and thus cannot be aligned perfectly.

Why is Point Clouds Registration important?

There are different situations in which *Point Clouds Registration* can be useful.

Localization

While navigating thorough a known environment, that is, a mapped environment, a robot needs to understand where it is with respect to the map.

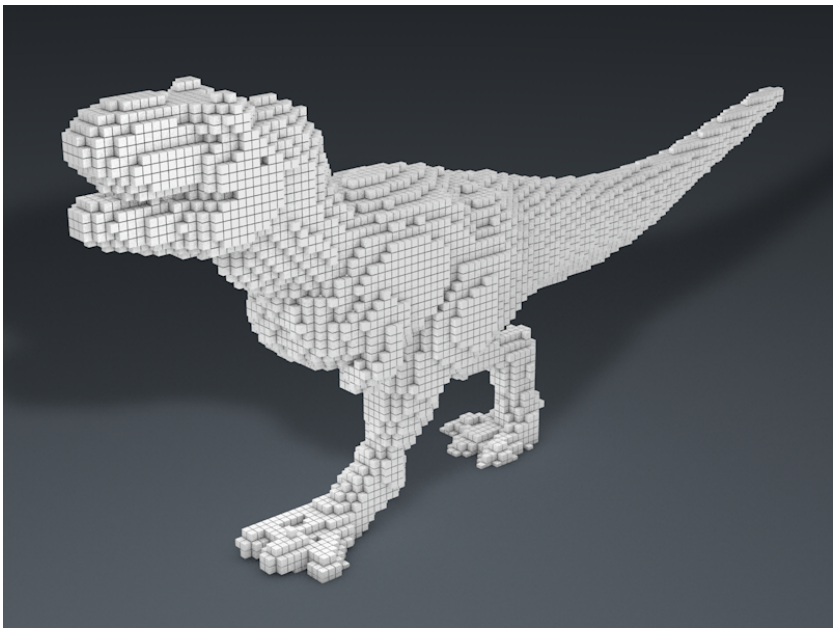


Figure 2.4: An example of Voxel map, from www.remotion4d.net

This is the problem of self-localization, often called just localization.

Suppose that the map is some kind of point cloud, or that it can be converted into a point cloud; this is the case, for example, of octomaps, [14], voxel maps (a kind of 3D grid map, Figure 2.4) or surface maps (Figure 2.5). Moreover, suppose that the robot is equipped with a sensor capable of producing point clouds. The sensor does not have to be the same used for mapping and it could even produce a different kind of point cloud. For example, consider the use of a sensor that produces dense point clouds used to navigate through a previously mapped environment, mapped with a sensor producing sparse point clouds. Independently from the setup and the sensors used, at each time step we would have two point clouds: the map, and the scene as currently seen by the robot. These two point clouds would have each its own local reference frame. In order to understand where the robot is w.r.t. the map, we could align the current point cloud with the map, that is solving the localization problem as a point clouds registration problem.

Since most point clouds registration techniques are local optimization algorithms, in order to solve the localization problem we need a rough initial guess. Usually this is not a problem, since this guess is usually supplied by some kind of odometry. Nevertheless, in Section 7.3 we will see how an accurate registration can be obtained even in absence of any kind of initial guess.

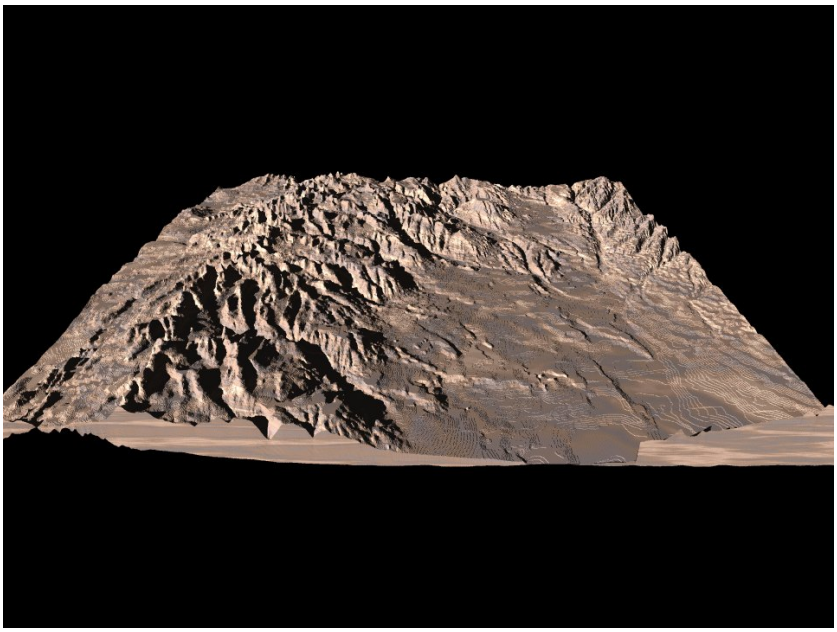


Figure 2.5: An example of Digital Surface Map, from www.terrainmap.com

Figure 2.6 presents an example of localization using point clouds registration. Suppose we have a point cloud of a corridor produced with a LiDAR and a robot equipped with a Kinect. By aligning the two point clouds we can estimate the pose of the robot in the corridor.

Mapping

Suppose we want to create a map of an environment. Usually one point cloud is not enough to represent a complex environment: the field of view and range of the sensors are limited and, moreover, there can be obstructions that make impossible to observe and then represent a whole environment from just a single point of view. Fusing different point clouds, taken in different places, is usually necessary. Thus, mapping can also be seen as an instance of a point clouds registration problem.

Moreover, we could have two maps produced by two different robots that we would like to merge. This is called multi-robot mapping and can be done for different reasons: for example because the two robots mapped two different parts of the environment (of course with some degree of overlap, otherwise the registration would not be possible), or because they used two different sensors with different peculiarities, like a visual-inertial system and a LiDAR.

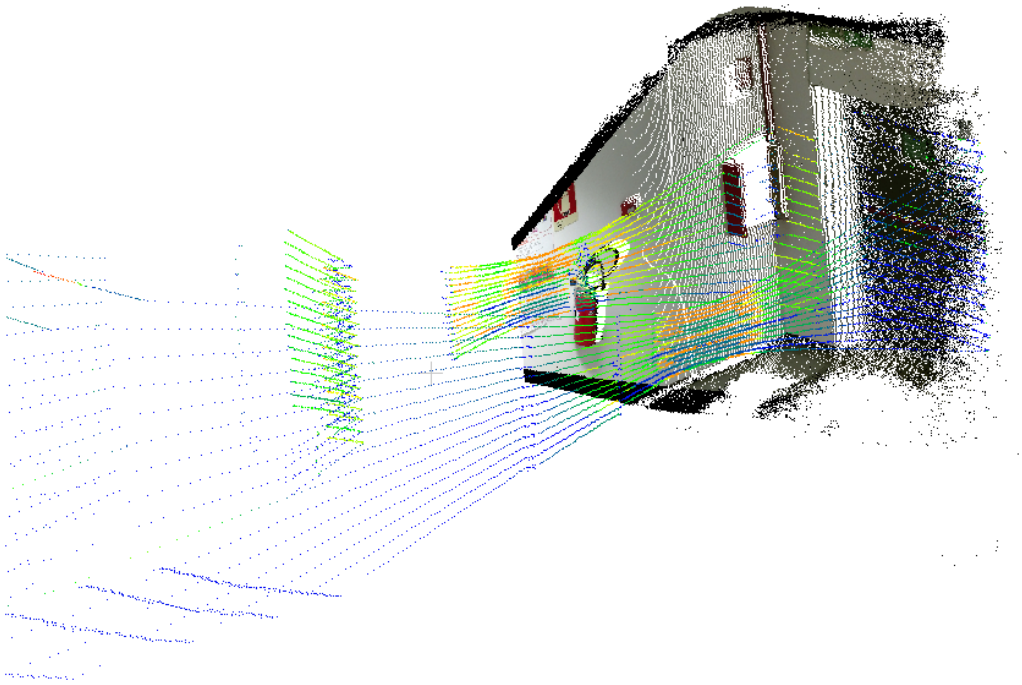


Figure 2.6: An example of localization using point clouds registration.

3D Reconstruction

The problem of 3D reconstruction is very similar to the mapping problem, but it has its own peculiarities. While with *mapping* usually we refer to the process of building a representation of a potentially dynamic environment, with 3D Reconstruction usually we refer to the process of building a 3D representation of an object in some *controlled* setup (be it a very small object, like a cup, or very large, like a whole building). The setup of the two processes is usually very different. Mapping is usually performed by a robot, often in a real open world. This poses several difficulties, like dealing with moving objects or with a high level of uncertainty in the measures. On the other hand, 3D reconstruction is often performed in more controlled conditions, where there are no moving objects, where the best point of view can be freely chosen, a great amount of measures can be gathered and the processing is usually performed off-line.

Regardless of these differences, the same consideration made for the mapping problem apply also to 3D Reconstruction.

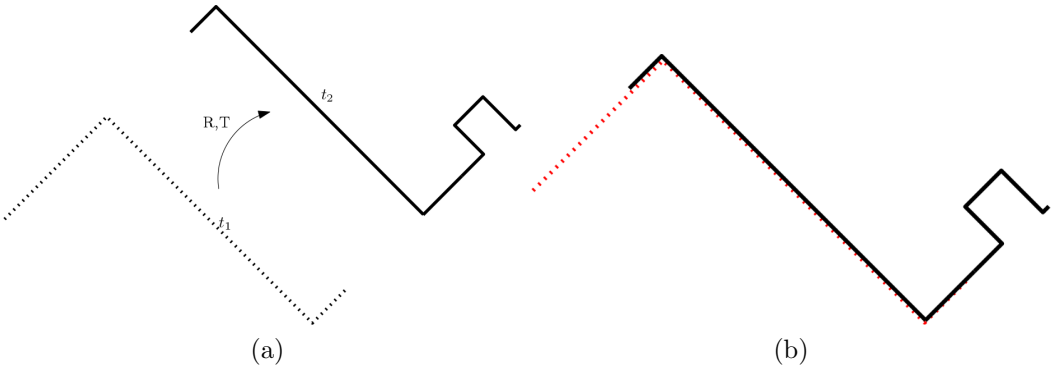


Figure 2.7: A schematic on how the odometry can be estimated using point clouds registration.

Odometry

Odometry is the process of estimating the pose of a robot over the time, by means of some sensor data. The difference between odometry and localization is that the first is usually a dead reckoning process. This means that the pose is not estimated w.r.t. an absolute reference system, but w.r.t. the pose of the robot at some previous time, usually measuring or calculating the displacement between the two poses. This has a very high impact on the uncertainty of the pose estimate, because over time this uncertainty will grow without any bound. Indeed, without some absolute measure, the measurement errors accumulate over time without any bound.

Nevertheless odometry estimation is still a very important part of a robotic system and is usually used as an input for many localization or mapping algorithms.

Many different kinds of odometry exist. The most common, by far, is odometry based on measuring the distance covered by the wheels of the robot (if available, of course). Another very popular alternative, for robots without wheels or in addition to wheel odometry, is based on IMU (Intertial Measurement Units).

What has Point Clouds Registration to do with odometry? Point Clouds Registration can play an important role in Laser Odometry: estimating the distance traveled by the robot by estimating the rototranslation between two consecutive laser scans or point clouds, Figure 2.7. Actually this problem, well known as scan-matching when referring to the data produced by single scanning plane LIDARs, is a problem of point clouds registration! One very important peculiarity of this application is that it needs to be strictly real-time and it cannot be done off-line (like 3D Reconstruction, for example).

2.2 Techniques for Point Clouds Registration

Feature-based registration

One large category of point clouds registration techniques are those exploiting some kind of geometric features, which are, basically, a representation of salient points of the underlining surface.

3D features are used in a similar way to what is done with 2D features extracted from images. First of all, salient points are detected in the point cloud. With "salient" points we mean points that are easy recognizable: it is the same concept behind corner extraction in computer vision. This kind of points are usually called *keypoints*. A naive way of extracting keypoints is to randomly select points or uniformly sub-sample the cloud. Of course these methods give no guarantee that the chosen points will be easily detectable in other point clouds, so other selection methods have been developed. Harris3D, [15], is one of those and was directly inspired by the popular Harris corner detector for 2D pictures, [16]. Other examples of keypoints detector,s specifically designed for point clouds and not derived from computer vision techniques, are the Intrinsic Shape Signature, [17], and NARF, [7]. A complete review of 3D keypoints detector is beyond the scope of this work. For an extensive review, see the work by Tombari *et al.*, [18].

Once the keypoints have been detected, we need, for each one, a descriptor, such that they can be matched in other point clouds. Examples of descriptors are PFH, [19], and their faster variant FPFH, [20], or angular-invariant features, [8]. Moreover, Sehgal *et al.* developed an approach, derived from the computer vision world, for point clouds registration that uses SIFT features extracted from a 2D image generated from the point cloud [21]. These are just example of the many 3D descriptors available. For a comparison of the various alternatives, see the work by Alexandre *et al.*, [22].

One of the main limiting factors to the use of 3D descriptors is that almost all require, in order to be computed, the normal to the surface on which the keypoint lies. This is not a problem for most point clouds, but it makes them not applicable to sparse point clouds. These kind of point clouds, indeed, usually does not carry enough information to accurately represent the underling surface. Therefore the normal to the surface cannot be reliably extracted. This is the reason because we will not use features descriptors in this work: we wanted to develop a technique as general as possible, that could be applied to the registration of any kind of point cloud.

Once descriptors have been computed in both the source and the target point clouds, they need to be matched. The easiest solution is to associate each keypoint in a point cloud to the closest, in the space of the descriptors,

in the other one. Of course the resulting set of associations will contain many outliers, that need to be filtered out. A popular technique for outliers rejection is RANSAC, [23]. A popular way to estimate the rototranslation from a set of noisy associations is to solve a least squares problem, for example using the SVD decomposition.

There are two main drawbacks to feature-based point clouds registration. First of all, it is usually a slow process: both keypoints and descriptors extraction are computationally expensive. This makes the technique not suitable for real-time applications. Secondly, and most important, the resulting alignment usually is not very accurate. For this reason feature-based registration is usually used not *per se*, but to estimate an initial guess that will be refined later on with other techniques (such as ICP).

Closest point-based registration

Iterative Closest Point

There is another relevant category of registration algorithms, whose most important member is Iterative Closest Point (ICP). This category is based on an approach completely different from feature-based registration. Instead than looking for keypoints and calculating their descriptors, so that they can be matched between the clouds, it greedily approximates the correspondences simply by looking for the closest point.

ICP was originally developed independently by Besl and McKay [24], Chen and Medioni [25], and Zhang [10]. Although its first introduction was in 1991, it is still the *de facto* standard for point clouds registration. ICP assumes that the two point clouds are already roughly aligned and aims at finding the rigid transformation, *i.e.*, a rototranslation, that best refines the alignment. The aim of ICP is to align the source cloud with the target cloud and, to do so, it repeats the following steps until convergence:

1. For each point x_j in the source cloud finds the closest point y_k in the target cloud.

2. Finds the best values for R and T (rotation and translation) that minimize the sum of squared errors in Equation (2.1).

3. Transforms the source cloud using R and T.

$$\sum_j \|Rx_j + T - y_k\|^2 \tag{2.1}$$

The algorithm may end, *e.g.*, after a predefined number of iterations, or when the sum of the residuals defined by Equation 2.1 becomes smaller than a certain threshold or when the difference of the rototranslations between two consecutive steps becomes smaller than a threshold.

ICP is very effective at aligning two point clouds that are already roughly aligned. Nevertheless, it needs an initial guess that has to be accurate enough, because the optimization algorithm it uses is local. Thus, in case the initial guess is very wrong, the algorithm could converge to the wrong solution or, rarely, not converge at all.

Usually, the minimization problem that constitutes each ICP iteration is solved in closed-form using the SVD decomposition. This is due to the fact that the problem can be represented as the linear system in Equation (2.2), where X is the matrix whose rows contain the points in the source point cloud and Y the matrix containing the points in the target point cloud, ordered so that the point in the i-th row of X corresponds to the i-th row of Y.

$$R \cdot X + T = Y \tag{2.2}$$

Since the linear system in Equation (2.2) is overdetermined and almost surely inconsistent, due to wrong data associations and noise in the point clouds, simple algorithms (such as Gaussian elimination) for linear system solving cannot be used. Instead, algorithms that provide a least squares solution have to be used. The SVD decomposition is the most used in this field, but other techniques exist. Instead than solving the problem in closed form, also generic non-linear optimization algorithms could be used, such as Levenberg-Marquard, [26, 27]. Although they are suited to the problem, the closed form solution is much faster and thus is usually preferred.

Many different variants of ICP have been proposed; usually they aim at speeding up the algorithm or at improving the quality of the result. For an extensive review and comparison of ICP variants, see the work of Pomerlau *et al.*, [28]. In this work we will refer to the basic ICP algorithm (also called Point-to-Point ICP) as implemented in the PCL library [29], because it is still the most used version and it is easily available for comparisons.

Generalized ICP

A variant of ICP that is worth mentioning is Generalized ICP (G-ICP) [30]. G-ICP modifies the standard ICP algorithm by attaching a probabilistic model to Equation (2.1), while keeping the other steps unchanged (including, notably, the closest distance based data association). The main idea is that there are two underlying set of points, $\hat{X}$ and $\hat{Y}$, used to generate X and Y.

Specifically:

$$x_i \sim \mathcal{N}(\hat{x}, \Sigma_i^X)$$
$$y_i \sim \mathcal{N}(\hat{y}, \Sigma_i^Y)$$

Therefore, points in the two point clouds to align, X and Y are assumed to be drawn from independent Gaussians. The distance $d_i^{(\mathbf{T})}$ between two points x_i and y_i, once aligned with a rigid transformation $\mathbf{T}$, is thus also drawn from a Gaussian with mean and covariance given by:

$$d_i^{(\mathbf{T})} \sim \mathcal{N}(\mathbf{T}\hat{x}_i - \hat{y}_i, \mathbf{T} \cdot \Sigma_i^X \cdot \mathbf{T}^T + \Sigma_i^Y) \tag{2.3}$$

assuming that the points association problem has already been solved, so that $\hat{x}_i$ corresponds to $\hat{y}_i$.

It follows that, applying MLE, the best transformation $\mathbf{T}^*$ can be computed using

$$\mathbf{T}^* = \arg\max_T \prod_i p(d_i^{(\mathbf{T})}) \tag{2.4}$$

By simplifying the above, Equation (2.1) is modified into Equation (2.5), where $\mathbf{T}$ is the rototranslation we want to estimate.

$$\sum_i (\mathbf{T}x_i - y_i)^T \cdot (\Sigma_i^Y + \mathbf{T}\Sigma_i^X\mathbf{T}^T)^{-1} \cdot (\mathbf{T}x_i - y_i) \tag{2.5}$$

By incorporating the covariances into the error function, G-ICP usually leads to better results, but at the expense of computation time. Indeed, Equation (2.5) cannot be expressed as a linear system and therefore has to be solved using a generic non linear optimization algorithm, such as Levenberg-Marquard. Given the good results usually obtained, G-ICP is one of the algorithm we will compare our approach to.

Normal-Distribution Transform

Normal-Distribution Transform is a method for representing and aligning point clouds, firstly introduced for 2D scans by Biber *et al.*, [12], and then extended also to 3D point clouds by Magnusson, [31]. Instead than looking for point correspondences, it uses a completely different approach, using a probabilistic representation for the point clouds.

First of all, a regular grid is fitted onto a point cloud. For each cell that contains at least three points, the points inside it are represented with

a Gaussian distribution, with mean and covariance calculated accordingly. Given the points $x_{i..n}$ in a cell, the mean and covariance of the Gaussian are

$$\mu = \frac{1}{n}\sum_i x_i$$

$$\Sigma = \frac{1}{n}\sum_i (x_i - \mu)(x_i - \mu)^T$$

This distribution $\mathcal{N}(\mu, \Sigma)$ represents, for each cell, the probability of finding a point for each position inside that cell.

To align another cloud to the first one, first it is transformed using an initial estimate for the rotation and translation. Each point in the second point cloud, once transformed with an initial guess, will lie in a particular cell of the first cloud. Therefore, for each of these points, a score is calculated, evaluating the distribution of the cell it lies in. A new improved estimate for the transformation is calculated trying to minimize the sum of all the scores, then these steps are repeated until convergence. The score is calculated according to Equation (2.6), where y_i are the points in the second point cloud, $\mathbf{T}$ is the current estimate of the roto-translation and μ_i and Σ_i are the mean and covariance of the cell in which y_i lies.

$$score(\mathbf{T}) = \sum_i e^{\left(\frac{-(\mathbf{T}\cdot y_i - \mu)^T \cdot \Sigma_i^{-1} \cdot (\mathbf{T}\cdot y_i - \mu)}{2}\right)} \tag{2.6}$$

Besides the differences regarding point correspondences, NDT and ICP are also similar because both are iterative algorithms that start with a rough initial guess for the transformation and try to improve it, by minimizing some error metric. NDT too will be used as comparison algorithm in this work.

2.3 Dense-sparse registration

The algorithms described so far have been designed to align two point clouds, without any reference to their density or to the difference of density between them. This is probably due to the fact that, commonly, point clouds registration techniques are used to align point clouds coming from the same sensor.

On the other hand, there are important situations where it would be useful to align point clouds with different densities. For example when mapping an environment with multiple robots, it could be useful to merge the maps in a unique global map, in which all the robots could localize. Localizing in a map produced with a sensor, for example a LiDAR, using a robot equipped

with another kind of sensor, for example a Kinect, is another instance of dense-sparse registration.

Eventually, also the problem of calibrating two different sensors on the same robot can be reconducted to a dense-sparse registration problem.

The problem of aligning a sparse point cloud with a dense one is, to our knowledge, little explored. Most of the works, indeed, tackle the problem of aligning two generic point clouds, without any reference to their density or to the density difference between the two. Although most of these techniques could be used to solve our problem, they produce less-than-optimal results, as we will show in the following sections.

For example, although ICP was not designed to explicitly solve the dense-sparse registration problem, it could be used anyway. Nevertheless, when used in such situations, it has some disadvantages. In case the two point clouds were produced using different kind of sensors, a problem, common also to the standard registration problem, is worsened: a point in a point cloud will never exactly correspond to another single point in the other. Instead, the hypothetical correspondent for a point could lie in between multiple points of the other point clouds. This problem exists also for registration problems between two point clouds coming from the same sensor, but is worsened when the sensors are different because they will have different scanning patterns. In this situation, for sure the standard point-to-point data association will be sub-optimal. This is a big drawback and thus the algorithm we propose in this work will also solve this issue by introducing a new kind of data association.

On the other hand, as already noted, feature based registration techniques could simply be inapplicable to very sparse point clouds, such those produced with visual-inertial systems.

2.4 Conclusions

Point clouds registration have become really popular in recent years, partially thanks to new sensors (mainly the Microsoft Kinect) that made producing 3D point clouds a cheap process, while in the past it was reserved to very expensive sensors. A complete review of the field is beyond the scope of this work. The selection of algorithms briefly presented here have been chosen because they are the de-facto standards for point clouds registration and are those widely used. For this reason they will be used to make comparisons against the algorithms we propose in this work.

Chapter 3

The Datasets

We used many different datasets to test the performances of various point clouds registration algorithms. Since they will be used later in many different chapters, to avoid repetitions, this chapter summarizes and describes them. These datasets have been chosen, among the many available, for a reason: they cover many different use cases and situations. We wanted our algorithms to be as general as possible and thus we had to test them in the whole spectrum of possible situations. For this reason too, we decided to record some novel datasets: for the specific use case of dense-sparse registration, we could not find any suitable set of data.

3.1 The Bremen Dataset

The *Bremen Dataset*, [32], is composed of eleven 3D scans taken in the city center of Bremen, in Germany. These were recorded with a Riegl VZ-400 laser scanner and contain also thermal information from a Optris PI IR camera. However, thermal information will not be used in this work. A ground truth, produced using odometry and refined with 6D SLAM, is available.

The Bremen dataset is a very big and challenging dataset for global registration because the point clouds are severely misaligned, as can be seen in Figure 3.1.

3.2 The Hannover Dataset

The *Hannover Dataset*, [33], is composed of almost one thousand 3D scans recorded at the University Campus of Hannover. Like the previous one, also this dataset represents an outdoor environment. However, it is much less challenging because is much more structured and usually there is a great

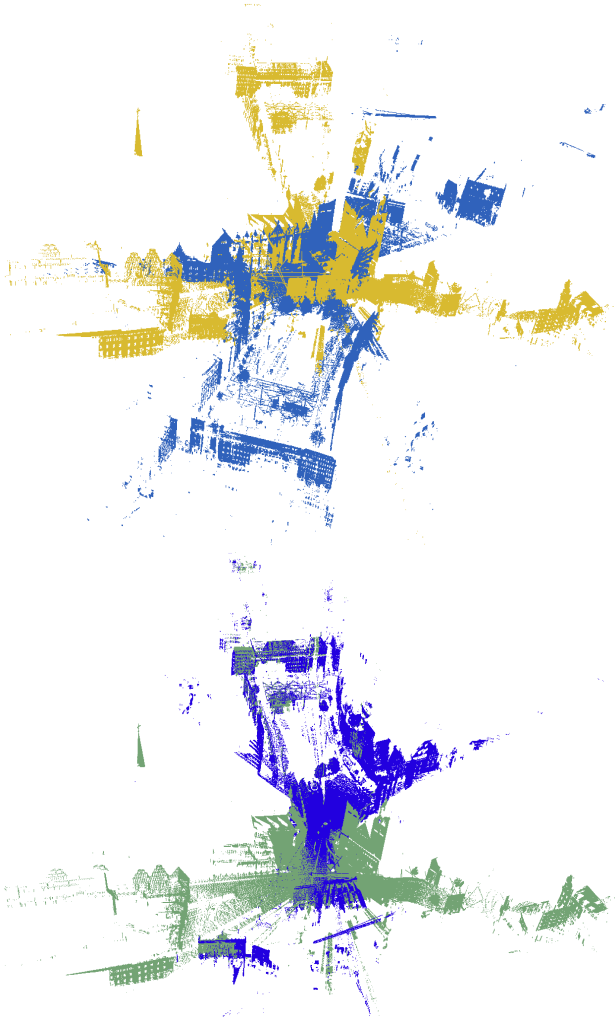


Figure 3.1: Two point clouds from the Bremen dataset. Before and after the alignment using the ground truth.

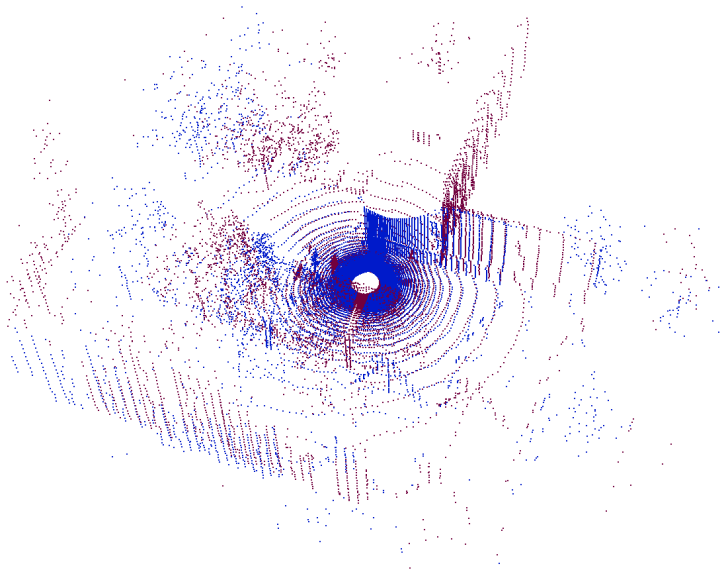


Figure 3.2: Two point clouds from the Hannover dataset, before the alignment.

amount of overlap between the point clouds, see Figure 3.2. For this dataset a ground truth provided by the odometry is available.

3.3 The Standford Bunny Dataset

The *Standford Bunny Dataset* is a very popular set of point clouds representing a bunny from different point of views, Figure 3.3. It was recorded by the Standford University Computer Graphics Laboratory, [34, 32], using a Cyberware 3030 MS scanner. Since this dataset poses relatively few challenges to modern point clouds registration algorithms, it was not used for the comparisons, but only to test the termination criteria of our probabilistic approach.

3.4 The Linköping Dataset

The *Linköping Dataset*, [13], contains data provided to us by the UASTech Laboratory of the Linköping University, in the context of the SHERPA European project (http://www.sherpa-project.eu). The dataset has been acquired with an aerial robot and represents a large rural area, with trees and some buildings, Figure 3.4. It is composed of two point clouds: the first one is relatively sparse (it is composed of 235486 points) and has been produced

Figure 3.3: The Standford Bunny Dataset

with a LiDAR, while the second one is denser (506742 points) and has been produced using photogrammetry with images from a camera.

Both sensors were mounted on the same robot, however, due to noise and distortion in the cloud produced with the camera, the two point clouds cannot be aligned perfectly applying a rigid transformation, Figure 4.3. Therefore, this dataset represents a very challenging test bench.

For this dataset, we precisely aligned the two point clouds manually, in order to have a ground truth to use to quantitatively compare the various techniques.

3.5 The Office and Corridor Datasets

The *Office* and the *Corridor* datasets have been recorded by us, [13]. They represent the typical office environment, with desks, chairs and computers. They are each composed of two point clouds, one recorded with a Velodyne VLP-16, the other with a Kinect 2.

These datasets are an example of sparse-dense registration. The Kinect 2 produces very dense, although noisy, point clouds and has a relatively narrow field of view (43° vertically and 57° horizontally). The Velodyne VLP-16 is a laser scanner that has sixteen scanning planes and an horizontal field of view of 360°. The produced point clouds are definitely less dense than those

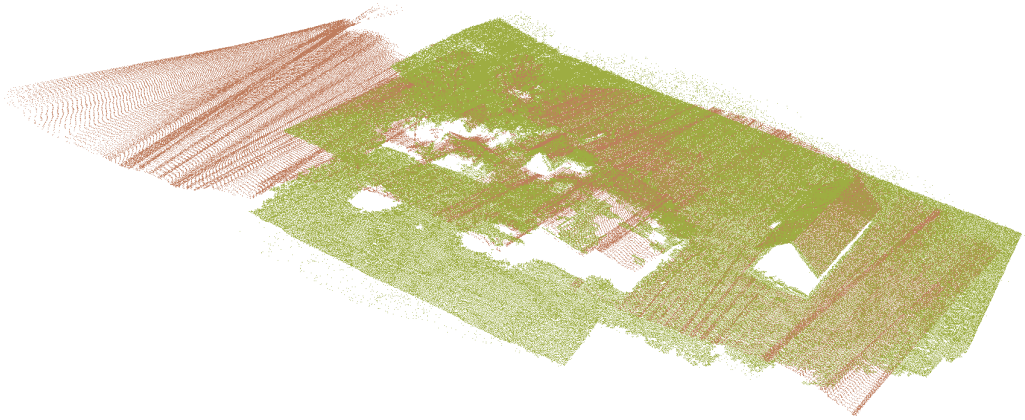


Figure 3.4: The Linköping dataset.

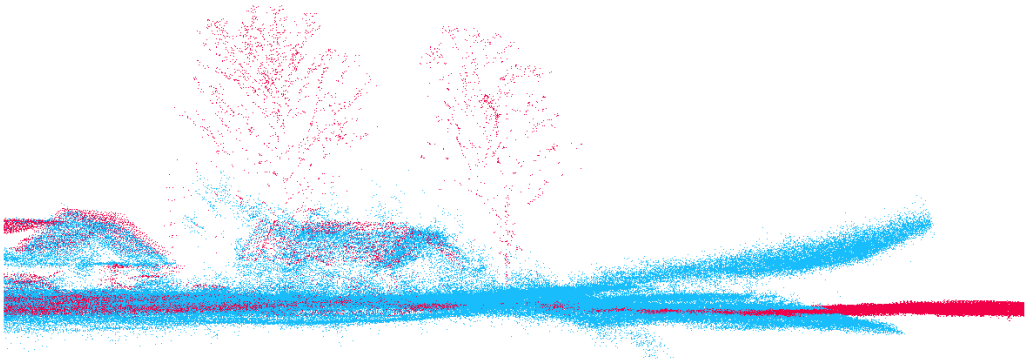
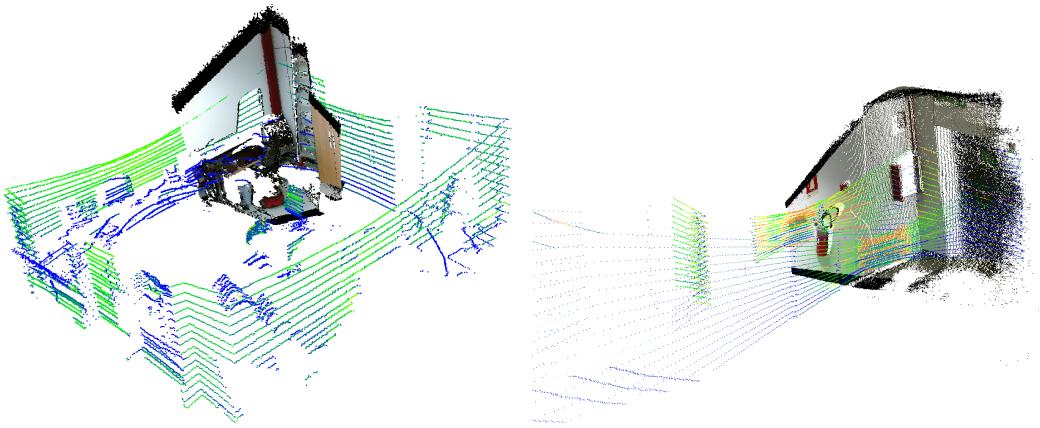


Figure 3.5: A detail of the two point clouds from the Linköping dataset. Notice the high amount of distortion in the cloud produced with the camera.



(a) Office (b) Corridor

Figure 3.6: The point clouds from the *Office* and *Corridor* datasets. The RGB-D point cloud has been produced with a Kinect 2, the other one with a Velodyne VLP-16.

produced with a Kinect 2, but are also less affected by noise. Thus, the point clouds produced with the VLP-16 are the sparse point clouds and represent a wider view of the environment, while those produced by the Kinect 2 are the dense point clouds and represent a more detailed and narrower view. To better understand the difference in density, consider that, for the Office dataset, although narrower, the cloud produced with the Kinect contains 215793 points. On the other hand those produced with the VLP-16 contains only 26848 points.

For this datasets too the ground truth have been produced by manually aligning the clouds.

Chapter 4

Point clouds registration with probabilistic data association

4.1 Overview

In this chapter we present a novel approach for introducing robustness in a point clouds registration algorithm. Although our approach was originally motivated by the problem of aligning a dense point cloud with a sparse one, it can also be used as a substitute of ICP for generic point clouds registration. For this reason, in the following sections, we won't make any reference to sparse and dense clouds, but we will use the usual ICP notation (target and source point clouds). Our main contribution is a new data association policy, which we called Probabilistic Data Association, because it was derived by applying statistical inference techniques on a fully probabilistic model. The final result is an algorithm similar to ICP, but more robust w.r.t. noise and outliers.

The main difference between our proposal and the standard ICP data association policy is that ICP assigns to each point in the source point cloud the closest one in the target point cloud. Instead, we associate to each point in the source point cloud, a sets of points in the target point cloud. These points can be chosen in two different ways: either we associate a certain fixed number of closest points, or all the points under a certain distance. Also a combination of both criteria is possible: we could associate the n closest points only if they are closer than a threshold.

The associations are then weighted and used as error terms in a minimization problem.

As already said, this choice was motivated by using inference on a probabilistic model, however it has also an intuitive explanation. A point in the

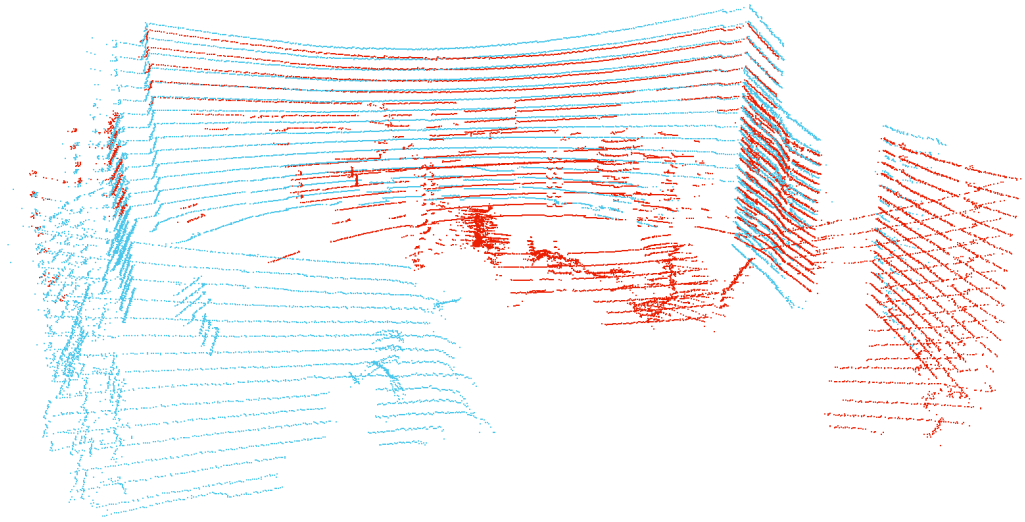


Figure 4.1: A data association policy based on a single neighbour tries to align the scanning line. On the other hand, the best alignment could have the lines not aligned because they correspond to different points in the scene.

source point cloud won't have an exact correspondence in the target. This is due to several reasons: for example the two point clouds could have been generated using different sensors with different scanning patterns or densities. Even if they were recorded using the same setup, the measured points in a scene as seen from a point of view won't necessary be the same measured from another point of view. Figure 4.1 shows an example of this situation. A data association policy based on a single neighbour tries to align the scanning line. On the other hand, the best alignment could have the lines not aligned because they correspond to different points in the scene. For this reasons, it seems reasonable that the correspondence for a point will not be exactly a point in another cloud, but it will lie somewhere in the middle of a set of points. The weights of the associations represents exactly this intuition: they say where in the set of points the correspondence lies.

4.2 Model Definition

Problem statement

Suppose that we have two point clouds taken from the same scene, X and Y, composed of points $x_1, ..., x_n$ and $y_1, ..., y_m$. These may have been acquired with different sensors (e.g., a camera and a laser scanner) or with the same sensor at different times; they may also have very different densities, such

as a dense, laser scanner-produced, point cloud and a sparse keypoints map. Our approach is completely agnostic w.r.t. these characteristics. We want to recover the rigid transformation between the two point clouds, i.e., the roto-translation that best aligns X with Y. The limitation to only rigid transformations has, of course, some effects on the quality of the result; indeed it is a simplification often used in the literature and usually, as long as the two point clouds are not heavily distorted, leads to good results.

For each point y_k in Y, we may define

$$y_k = Rx_j + T \tag{4.1}$$

where x_j is the point in the point cloud X corresponding to point y_k in the point cloud Y and R and T are, respectively, the rotation and translation that align X with Y. In practice, the true point associations are unknown and Equation (4.1) is never exactly holding, because of noise. Uncertainty due to sensor noise is often treated as a random variable, *e.g.*, an additive white Gaussian noise added to a deterministic value. We argue that association ambiguity is also a source of uncertainty and therefore it should also be treated as a random variable.

Probabilistic model

In order to reason about sensor noise and data association uncertainty simultaneously, we define a probabilistic model. Probabilistic models are attractive because:

1. they are interpretable and extensible;
2. they allow well-behaving optimization criteria, *i.e.*, the negative log-likelihood;

A probabilistic model is defined by a series of statements about the distribution of the variables involved in the model. For example, we define:

$$p(y_k|x_j, a_{jk} = 1) \sim \mathcal{N}(Rx_j + T, \Sigma) \tag{4.2}$$

where $a_{jk} = 1$ if point x_j corresponds to point y_k. This means that, if we were certain that point x_j in X corresponds to point y_k in Y, then y_k would follow a multi-variate normal distribution with mean $Rx_j + T$ and covariance Σ. To complete the model we must place a prior distribution over a_{jk}. Suppose we have a set C_k of candidate points from X corresponding to y_k. Then, we may define

$$P(a_{jk}) = \left\{ \begin{array}{cc} |C_k|^{-1} & j \in C_k \\ 0 & j \notin C_k \end{array} \right\} \tag{4.3}$$

This means that, a priori, all the points in C_k are equally likely to be associated to y_k.

In practice we prefer a slightly more sophisticated model that could account for outliers produced by sensor noise. For this reason, instead of a Gaussian, a t-distribution is more appropriate. This is due to the *shape* of a t-distribution: depending on its degree of freedom, it gives a higher probability to the tails of the distribution. On the contrary, in a Gaussian, the tails have an almost negligible probability. By giving a slightly higher probability to the tails we are able to implicitly take into account for outliers. We redefine the model as follows:

$$p(y_k|x_j, a_{jk} = 1) \sim \mathcal{T}(Rx_j + T, \Sigma, \nu) \tag{4.4}$$

where $\mathcal{T}$ denotes the family of multi-variate t-distributions and ν represents its degree of freedoms. Equation (4.4) states that y_k conditioned to being the correspondent of x_j is t-distributed. A t-distribution is a heavy-tailed distribution and ν controls the weight of the tails. For $\nu \to \infty$ the t-distribution reduces to a Gaussian. For finite ν it assigns a non-negligible probability to the tails, thus implicitly taking into account for outliers, without the need to pre-filter them or to treat them as a special case.

It is convenient to re-parametrize the model. Namely, the t-distribution in Equation (4.4) is equivalent to

$$p(y_k|x_j, a_{jk} = 1, w_k) \sim \mathcal{N}(Rx + T, \frac{\Sigma}{w_k}) \tag{4.5a}$$

$$w_k \sim \Gamma(\frac{\nu}{2}, \frac{\nu}{2}) \tag{4.5b}$$

where $\Gamma(a, b)$ denotes a Gamma distribution with shape a and rate b, [35]. The convolution of Equation (4.5a) and Equation (4.5b) produces Equation (4.4). The weight w_k is an auxiliary variable that arises from the parametrization.

It is known that a t-distribution can be characterized as a mixture of Normal distributions with Gamma mixing weights, [36]. Suppose that there exists a random variable A, with $A \sim \mathcal{N}(0, \Theta^{-1})$, thus with mean 0 and variance Θ^{-1}. Moreover, suppose that Θ follows a Gamma distribution with both shape and rate equal to the same value α. Since it follows a Gamma distribution, there is uncertainty on the variance and thus is more appropriate to write that $A|\Theta \sim \mathcal{N}(0, \Theta^{-1})$. The unconditioned distribution of A will then follows a t-distribution with 2α degree of freedoms. In our case, $\alpha = \frac{\nu}{2}$.

This particular parametrization is convenient because, if we knew a_{jk} and w_k, then the negative log-likelihood would be a quadratic function of x_j and we could run a non-linear least-squares solver.

4.3 Point Clouds Registration as an EM Problem

The model defined can be used to recover the rigid transformation between the two point clouds using an optimization algorithm. It needs a rough initial guess and it iteratively improves it, simultaneously estimating the transformation and the point associations.

Expectation-maximization (EM), [37], is a procedure that can be used to iteratively estimate the marginal log-likelihood of the data, given a set of parameters. Each EM iteration consists of two steps: the E-step and the M-step. The E-step effectively estimates the values of the hidden variables by evaluating expectations, while the M-step updates the parameters in order to decrease the expected negative log-likelihood. EM is well-suited for models containing latent variables, such as ours. In our case the latent variables are the auxiliary weights w_k. The negative log-likelihood we want to minimize with EM is given by

$$l(x) = -\ln \sum_{k=1}^{m} \sum_{j=1}^{n} \int_{w_k} p(y_k|x_{C_k}, a_{jk}, w_k) p(a_{jk}) p(w_k) dw_k \tag{4.6}$$

where x_{C_k} is the set of all the points x_j such that $j \in C_k$ and p denotes the probability density function implied by Equations (4.5a) and (4.5b).

Minimizing the negative log-likelihood directly is difficult due to non-convexity. It is much easier to minimize a convex upper bound on Equation (4.6). Let q_k be an arbitrary probability density function of a_{jk} and w_k. Then, applying Jensen's inequality, [38], leads to

$$l(x) \leq \sum_{k=1}^{m} b_k(x_{C_k}, q_k) \tag{4.7}$$

where

$$\begin{aligned} b_k(x_{C_k}, q_k) = -\sum_{j=1}^{n} \int_{w_k} q_k(a_{jk}, w_k) ln(p(y_k|x_{C_k}, a_{jk}, w_k)) d_{w_k} + \\ + \sum_{j=1}^{n} \int_{w_k} q_k(a_{jk}, w_k) ln(\frac{q_k(a_{jk}, w_k)}{p(a_{jk}) p(w_k)}) dw_k \end{aligned} \tag{4.8}$$

The inequality in Equation (4.7) defines an upper bound on $l(x)$. If $q_k(a_{jk}, w_k) = p(a_{jk}, w_k|x_{C_k}, y_k)$, then the bound becomes tight and the inequality becomes

an equality. If we evaluate the expectations in Equation (4.8) and retain only the terms involving the points we obtain

$$b_k(x_{C_k}, q_k) = \frac{1}{2} \sum_{j \in C_k} \rho_{jk} r_k^2(x_j) \tag{4.9}$$

where

$$r_k^2(x_j) = \|y_k - (Rx_j + T)\|^2 \tag{4.10}$$

is the squared Euclidean norm of the residual, and

$$\rho_{jk} = \sum_j \int_{w_k} \delta(a_{jk}, j) w_k q(a_{jk}, w_k) dw_k \tag{4.11}$$

is the residual weight. Hence for a fixed q_k, b_k is a quadratic-composite function of the points' positions in the source cloud. It is also a local function, depending only on the candidate corresponding points for the k-th point in the target cloud.

EM minimizes the upper bound coordinate-wise. The E-step minimizes the bound with respect to q and computes the residual weights, hence it is equivalent to a re-weighting step. The M-step updates the objective function along a descent direction. Therefore, applying EM is equivalent to solving an iteratively re-weighted, non-linear least-squares problem.

1. The E-step: For fixed x_j, the minimum of b_k occurs when

$$q_k(a_k, w_k) = p(a_{jk}, w_k | x_{C_k}, y_k) \tag{4.12}$$

In other words, minimizing the upper bound with respect to q_k is the same as computing the joint posterior distribution over all the a_{jk} and w_k , given x_j. Due to conjugacy, the posterior has the same mathematical form as the prior, *i.e.*, multinomial Gamma. Specifically,

$$P(a_{jk} | x_{C_k}, y_k) \propto t(y_k, Rx_j + T, \Sigma, \nu) \tag{4.13a}$$

$$w_k | a_{jk}, x_{C_k}, y_k \sim \mathcal{G}\left(\frac{\nu + d}{2}, \frac{\nu + r_k^2(x_j)}{2}\right) \tag{4.13b}$$

where t is the density function of the multi-variate t-distribution, and the proportionality sign implies a normalization constant such that $\sum_{j \in C_k} P(a_{jk} | x_{C_k}, y_k) = 1$.

Evaluating the expectation in Equation (4.11) yields

$$\rho_{jk} = P(a_{jk} | X_{C_k}, y_k) E[w_k | x_{C_k}, y_k, a_{jk}] \tag{4.14}$$

where

$$E[w_k|x_{C_k}, y_k, a_{jk}] = \frac{\nu + d}{\nu + r_k^2(x_j)} \tag{4.15}$$

follows from the properties of the Gamma distribution.

2. The M-step: There are several different ways of updating the rotation and translation in our model. A trust-region method such as Levenberg-Marquardt, [39], solves a sequence of quadratic sub-problems in the form

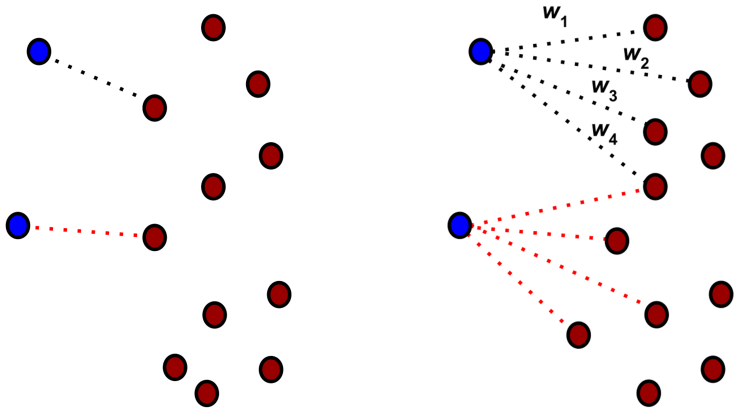
$$\min_{\Delta x_j} \sum_{k,j \in C_k} \rho_{jk} \, ||y_k - (Rx_j + T)||^2 \tag{4.16}$$

The solution to a sub-problem is an update step which is applied to the current estimate of the rotation and translation between the clouds.

4.4 Implementation

Essentially, our approach differs from ICP in the data association. In ICP each point in the source point cloud is associated only with a point in the target point cloud. On the other hand, the proposed algorithm associates a point in the source point cloud with a set of points in the target cloud. Moreover, the set of associations are not changed at every iteration, but remain the same for the whole duration of the algorithm. What is changed, instead, are the weights of the associations, that are updated during the *expectation* phase of the EM algorithm. The two different data association methods are depicted in Figure 4.2.

The candidate points to be associated may be found in different ways, for example by nearest neighbours search or by feature matching. We found that, for the problem of sparse-dense registration, and given a reasonable initial hypothesis on the transformation, the nearest neighbours search proved to be good enough, while remaining very fast to compute. In contrast, feature extraction and matching is usually a slow process. Feature-based data association is not an option for sparse-dense registration, since sparse point clouds usually do not contain enough information to extract discriminative geometric features. However, our approach can potentially accommodate feature matching as prior information. For instance, in Equation (4.3), the prior association probabilities can be scaled according to a non-negative feature similarity metric, thus assigning a higher prior probability to similar (in the space of the descriptors) points. Moreover, feature matching could even replace the closest points based data association entirely. Our algorithm would remain mostly unchanged: it just needs some kind of distance,



(a) ICP Data Association

(b) Probabilistic Data Association

Figure 4.2: The two different data association policies

it does not matter whether it is a distance in the euclidean space or in the descriptors space.

Let us now suppose to proceed with the nearest neighbour policy. For each point x_j in the source point cloud, we look for the n nearest points, $y_0, ..., y_n$, in the target point cloud. For each of these points y_k, with $0 \leq k \leq n$, we define an error term given by

$$\|y_k - (Rx_j + T)\|^2 \tag{4.17}$$

Equation (4.17) represents the squared error between the point y_k in the target point cloud and the associated point x_j from the source point cloud, transformed using the current estimate of the rototranslation.

Our point cloud registration algorithm is composed of an optimization problem, whose error terms are calculated according to Equation (4.17) and that is then solved using a suitable method (such as Levenberg-Marquardt). However, given a set of points associated to x_j, not every corresponding error terms should have the same weight. Intuitively, we want to give more importance to the associations that are in accordance with the current estimate of the transformation and a lower importance to the others. Thus, using the model described earlier, the weight of the error term $\|y_k - (Rx_j + T)\|^2$ is given by

$$w_{kj} \propto e^{-\frac{\|y_k - (Rx_j + T)\|^2}{2}} \tag{4.18}$$

where the proportionality implies a normalization among all the error terms associated with x_j, so that their weights represents a probability distri-

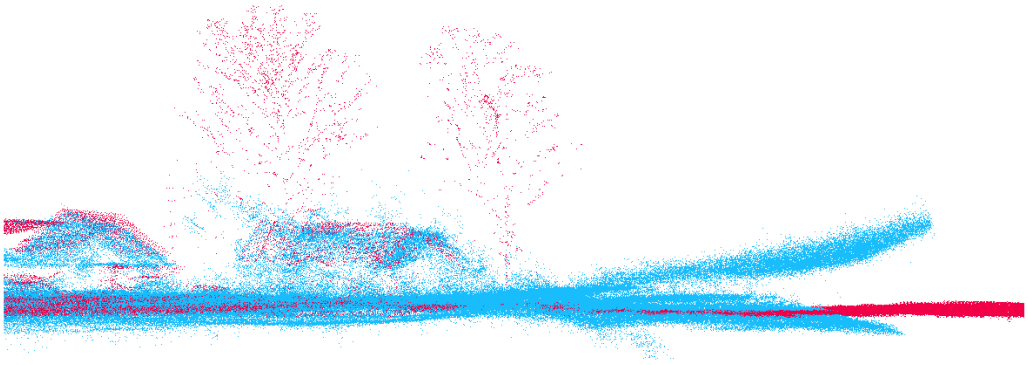


Figure 4.3: Two point clouds from the Linköping dataset. The best achievable alignment is represented: the high amount of noise and distortion (especially at the edges), makes the error in some parts very high.

bution. Equation (4.18) is derived from the EM algorithm, with an additive Gaussian noise model.

The Gaussian in Equation (4.18) works well, provided there are no outliers and all points in the source point cloud have a corresponding point in the target point cloud. However, as already described, a t-distribution is a better choice in presence of outliers, especially when there is lot of distortion in one of point clouds. Distortion, indeed, make finding a very good alignment impossible. Some parts of the clouds will be aligned correctly, while others will not, even if the point correspondences have been estimated correctly. This problem is exemplified by Figure 4.3, where two point clouds from the Linköping dataset are depicted. Even though the best achievable alignment is represented, the high amount of noise and distortion (especially at the edges), makes the error in some parts very high.

Consequently, a more robust equation for the weights, basing on the t-distribution, is given by

$$p_{kj} \propto \left(1 + \frac{\| : -(Rx_j + T)\|^2}{\nu}\right)^{-\frac{\nu+d}{2}} \tag{4.19}$$

$$w_{kj} = p_{kj} \frac{\nu + d}{\nu + \|y_k - (Rx_j + T)\|^2} \tag{4.20}$$

where ν is the degree of freedom of the t-distribution and d the dimension of the error terms (in our case 3, since we are operating with points in the 3D space).

However, in order to calculate the weights, we need an estimate of the rotation and translation, but these are estimated by solving the optimization problem whose error terms are weighted with the weights we want to

calculate. Hence our problem cannot be formulated as a simple least-square error problem, but it has to be reformulated as an Expectation-Maximization problem. During the Expectation phase the latent variables, in our case the weights, are estimated using the previous iteration estimate of the target variables (the rotation and translation), while during the Maximization phase, the problem becomes a least-square error optimization problem, with the latent variables assuming the values estimated during the Expectation phase.

4.5 Experimental Results

Since the proposed approach was designed primarily to deal with the problem of registering a sparse point cloud with a dense one, the first set of experiments we present is relative to this setup. However, since it proved to be effective also for generic point clouds registration, we will present also other kinds of experiments.

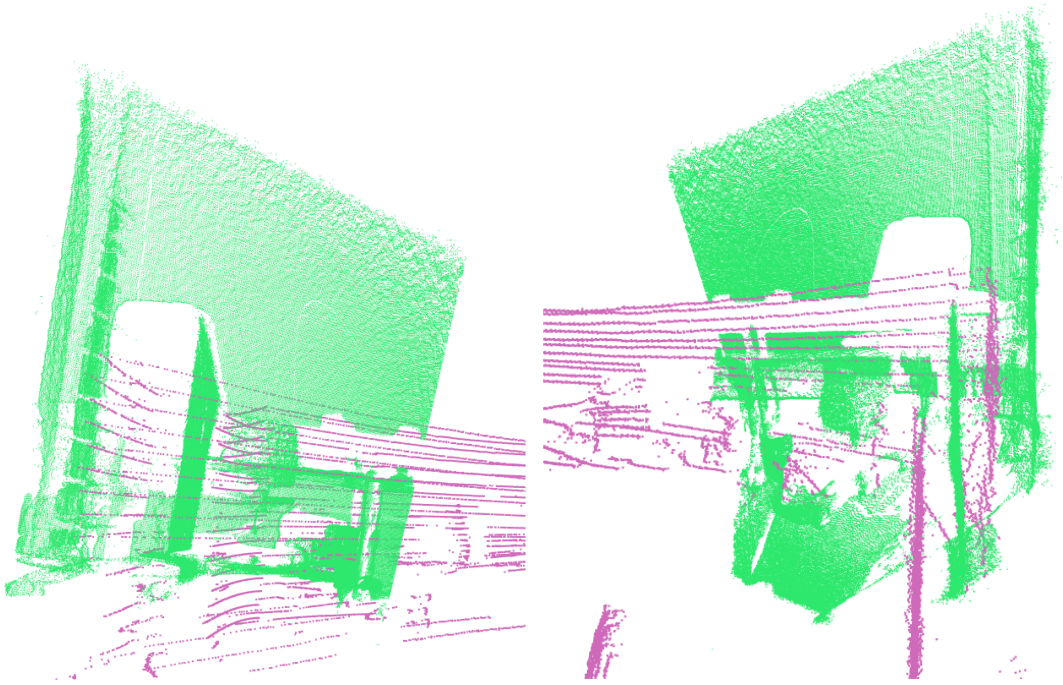
Dense-Sparse Registration Experiments

We tested the proposed approach and compared it against other techniques in the context of dense-sparse registration. Here we will show results on three pairs of point clouds, coming from the Linköping, Office and Corridor datasets. Since the described approach, like the other we compare to, can be used only for fine-registration (that is, it needs a rough initial guess), the transformation between the two point clouds in the same dataset is relatively small, Figure 4.4. Nevertheless, the datasets are very challenging because the two point clouds present very different scanning patterns, different densities and, in some cases, are affected by a large amount of noise and distortion.

Results

Since our objective was to compare our probabilistic approach to other point clouds registration techniques, we used various algorithms on the same data and compared the results. The techniques we compared are:

- Iterative Closest Point (ICP)
- Generalized Iterative Closest Point (G-ICP)
- Normal Distribution Trasform (NDT)
- The proposed “probabilistic” approach



(a) (b)

Figure 4.4: Two different views of the initial misalignment between the two point clouds in the "office" dataset. Flat colours have been used for clarity.

These algorithms, other than our, have been chosen among the many in the literature, because they are the *de-facto* standard for point clouds registration and because they are publicly available for comparisons. Indeed, we used the implementation available in the PCL Library, [29].

The metric used to compare the results is the mean distance between the points in the source point clouds, as aligned by an algorithm, and the ground truth. That is:

$$\frac{\sum_{i=0}^{N} ||x_i - g_i||}{N} \tag{4.21}$$

Where x_i is a point in the registered source point cloud, g_i is the corresponding point in the ground truth and N is the cardinality of the point clouds. Since the source cloud and the ground truth are actually the same cloud displaced, we know exact point correspondences. This is possible because applying a roto-translation, using the PCL library, preserves the order of the points. Given the metric, the smaller the value of Equation (4.21) the better the result.

The results of our experiments are shown in Tables 4.1 to 4.3.

One problem with our proposal is that, since the point associations do not change at every iteration, it is very sensible to the initial data association. Essentially it is the equivalent of running a single iteration of ICP or G-ICP. To choose the points in the target cloud to be associated to a particular point in the source cloud, we simply look at those within a certain distance (the *radius* parameter of the algorithm) or, in a way similar to ICP, to the k-nearest neighbours. If among all the points in the target cloud associated to a particular point in the source cloud the right data association is not present, whenever this happens for a significant number of points, the algorithm will not converge to the right solution. One way to avoid this problem consists in applying our probabilistic algorithm several times, each time using as input the solution of the previous run, and thus re-estimating the correspondences. In this way, our technique becomes similar to ICP, where each iteration is a run of our algorithm. In this case, the difference w.r.t. ICP is that, instead of using a single nearest neighbour, we use a set of points, and each association is weighted as described. In the tables this technique is described with the phrase "Multiple Runs". When used in this way, the execution time of the algorithm increases greatly. As an example, for the corridor dataset, on a machine with an Intel i5-760 processor and 8 GB of RAM, the execution times are:

- $7741ms$ for the "Multiple Runs" mode
- $252ms$ in single run mode
- $70ms$ for ICP
- $2140ms$ for G-ICP
- $1902ms$ for NDT

These times are purely indicative, since they depends strongly on the problem that has to be solved and from the parameters used. The execution time, indeed, increases with the number of neighbours used.

The experiments show that the proposed probabilistic approach is not always better than the other point clouds registration techniques when used with a single iteration. This is expected because of the limitations just explained. However it provides much better results than all the other techniques when used in the "Multiple Runs" mode. Since this approach seemed very promising, we further developed it.

On the Office dataset our approach performed, in *Multiple Runs* mode, one order of magnitude better than ICP and NDT. It performed better than

G-ICP too, but in that case the difference is much smaller and probably negligible.

On the Corridor dataset it still performed better than the other algorithms tested, but with a smaller difference than for the Office dataset. Where it really excelled is with the Linköping dataset, where the difference w.r.t. ICP and NDT is very large. G-ICP, instead, still performs very well, even though worse than our approach. This demonstrates that our intuition on how to deal with noise and distortion was right. Indeed, our approach performed the best especially in that situation.

During these experiments the number of iterations was fixed and the parameters of the registration algorithm remained the same among the iterations. These parameters are:

- the maximum number of neighbours for a point;
- the maximum distance for a point in the target cloud, in order to be considered a neighbour of a point in the source cloud;
- the degree of freedom of the t-distribution and whether to use it instead than a Gaussian;
- how each point clouds (source and target) is sub-sampled. Sub-sampling is a very important step in point clouds registration. As we will show in the following chapters, it can heavily affect the quality of the results if done in the wrong way and, sometimes, even if not performed. Moreover, there is no reason that force us to keep the same level of sub-sampling among the iterations. So it could be iteratively adjusted in order to improve the quality of the result or the speed of convergence (or maybe both).

The effect of these parameters on the quality of the result will be examined in the following chapter. Since the results change considerably according to the parameters used, the results in the tables are the best obtained, after trying with different parameters set.

Standard Point Clouds Registration Experiments

Since our approach is suitable also for the resolution of standard point clouds registration problems, we performed experiments also on two other datasets: the Bremen and Hannover datasets.

The point clouds in the Bremen dataset are very misaligned, therefore local registration techniques often struggle to find proper solutions. For this

Table 4.1: Tests with the Office dataset

Algorithm	**Residual Mean Distance**
Probabilistic	0.386401
Prob. (multiple runs)	0.040809
ICP	0.553467
G-ICP	0.074794
NDT	0.783207

Table 4.2: Tests with the Corridor dataset

Algorithm	**Residual Mean Distance**
Probabilistic	0.315502
Prob. (multiple runs)	0.073592
ICP	0.286967
G-ICP	0.100754
NDT	0.129964

Table 4.3: Tests with the Linköping dataset

Algorithm	**Residual Mean Distance**
Probabilistic	1.842854
Prob. (multiple runs)	0.43761
ICP	1.680124
G-ICP	0.6527864
NDT	3.564592

Table 4.4: Tests with the Bremen dataset

Algorithm	**Residual Mean Distance**
Probabilistic	12.9304
Prob. (multiple runs)	0.23375
ICP	0.316433
G-ICP	0.227441
NDT	0.263119

Table 4.5: Tests with the Hannover dataset

Algorithm	**Residual Mean Distance**
Probabilistic	6.96368
Prob. (multiple runs)	0.115388
ICP	0.116849
G-ICP	0.162272
NDT	6.41092

reason we decided to take the ground truth of one point cloud and apply to it a small translation and rotation, thus replicating the usual working conditions of the tested algorithms, see Figure 4.5. The results are shown in Table 4.4. All the algorithms, with the exception of the probabilistic algorithm used in single run mode, were able to find very good solutions. Differences in the error w.r.t. the ground truth are negligible, therefore the solutions have to be considered equivalent. The result of the registration with the Probabilistic approach is shown in Figure 4.5.

The results of the experiments on the Hannover dataset are shown in Table 4.5. G-ICP, ICP and the Probabilistic approach in multiple runs mode got very good results that are, considered the resolution of the point clouds, equivalent. On the contrary, NDT and the Probabilistic approach used in single run mode, could not perform a proper alignment. The result of the registration with the Probabilistic approach is shown in Figure 4.6.

The tests on these two datasets show that multiple iterations of the Probabilistic approach are able to obtain a proper alignment also with standard point clouds registration problems, not only when there is a difference in densities between the two point clouds. On the other hand, a single iteration is often not enough to obtain a good result. For this reason the following chapter will concentrate on finding some automatic termination criterion that goes beyond the use of a fixed number of iterations.

Figure 4.5: The point clouds from the Bremen dataset, before and after the registration with the Probabilistic approach.

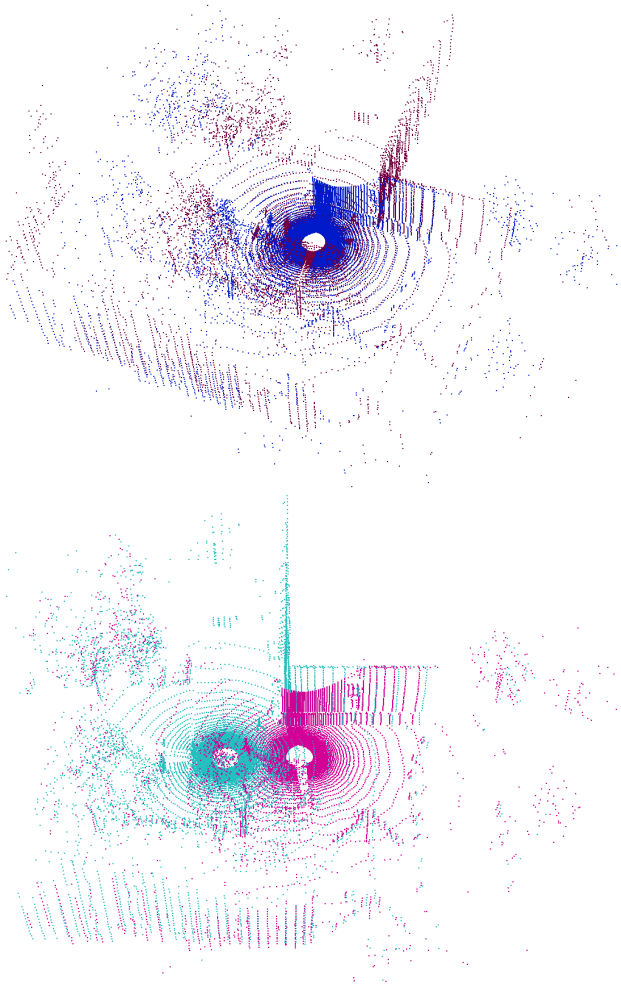


Figure 4.6: The point clouds from the Hannover dataset, before and after the registration with the Probabilistic approach.

Chapter 5

Multi-iteration Probabilistic Point Clouds Registration

5.1 Why Multiple Iterations?

In case that the source and the target point clouds are very close, a single iteration of the proposed probabilistic point clouds registration algorithm may be enough. However, in the typical real scenario, more iterations are necessary. In order for the algorithm to converge, most of the correspondences used to form the optimization problem needs to be right. Since we use a data association policy based on the euclidean distance, this happens only if the two point clouds are close enough. Two parameters control which and how many points in the target point cloud are associated to a particular point in the source point cloud: the maximum distance between neighbors and the maximum number of neighbors. Setting these parameters to very high values could help the algorithm to converge to a good solution even when the starting poses of the two point clouds are not really close. However, this trick will allow more outliers, *i.e.*, wrong data associations, to get into the optimization step. Even tough the probabilistic approach has the capability to soft-filtering out outliers, thanks to the probabilistic weighting technique, using too many points will lead to a huge optimization problem which would be very slow to solve. Usually, a much more practical and fast solution is to use low values for the maximum distance and the maximum number of neighbors and use multiple iterations of the probabilistic approach, that implies re-estimating the data associations, in the same way it is done, for example, in ICP and G-ICP.

With this technique, our approach becomes composed of two nested loops. The inner one solves an optimization problem using the Levenberg-Marquard

algorithm. The outer one moves the point cloud using the solution found in the previous step, estimates the point correspondences and build the corresponding optimization problem. This process is repeated until some convergence criterion is met.

As we show in Section 4.5, the multi iteration version of our algorithm provides good results, even compared to other state of the art algorithms. Of course, in order to be of practical usefulness, such an algorithm needs some kind of automatic termination criterion. It means that the algorithm should decide by itself when it should stop.

5.2 Termination criteria

The most simple termination criterion is to use a fixed predefined number of iterations. This solution is far from being optimal, since the number of required iterations becomes a parameter of the algorithm. There is no automatic way of estimating this parameter a-priori, so this solution is unpractical and has to be discarded. Using a fixed value for this parameter would probably mean using too many iterations in some cases and using too few in others. On the other hand, using a very high value would greatly increase the execution time, in many cases without improving the quality of the result.

To develop an automatic termination criterion, we used the following idea. Suppose we have a ground truth for our source point cloud, *i.e.*, we know the rototranslation between the reference frames of the source and target point clouds. At the end of each iteration, we get an estimate of this rototranslation. Thus, we can calculate the difference between our estimate and the ground truth. Theoretically, this difference should decrease among the steps of the outer loop of the algorithm, therefore the more the iterations, the smaller the difference becomes. Practically at some point this difference will cease to decrease, or, more precisely, it will start decreasing of a negligible amount. This is the iteration to which we should stop, since it means that our algorithm has converged to a solution. Note that it does not mean that it has converged to the right solution, but, nevertheless, that is the best solution we can get with that set of parameters. Obviously, in a real application, the ground truth is not available, thus we have to find an alternative metric that behaves in the same way, but uses the data we have.

Our first choice was to use, as an alternative, the Mean Squared Error (MSE) with respect to the previous iteration. We take two point clouds: one aligned with the current estimate of the transformation and the other aligned with the result of the previous iteration. Since the two point clouds

are actually the same cloud displaced, the point correspondences are known and exact. Simply, the $i-th$ point in the first point cloud corresponds to the $i-th$ point in the other, since applying a transformation preserves the order of the points. Thus, given one point cloud at two consecutive iterations of the algorithm X^t and X^{t-1}, with point x_i^t being the corresponding point of x_i^{t-1} and N the dimension of the point cloud, we used Equation (5.1) to calculate the Mean Squared Error (MSE).

$$MSE(X^t, X^{t-1}) = \frac{\sum_i^N ||x_i^t - x_i^{t-1}||^2}{N} \tag{5.1}$$

Therefore, we stop the algorithm when the MSE drops under a certain relative threshold. With *relative* we mean that we are not using a fixed absolute threshold, but we want to stop when, for example, the Mean Squared Error at an iteration becomes smaller than a certain fraction of that at the previous iteration. This means that we are stopping the algorithm when it is not able to move (or it is moving of a negligible amount) the source point cloud any more, thus it has converged. We use a relative threshold, instead than an absolute, because it is much more flexible and does not have to be tuned for each set of point clouds.

Another option is to use the so-called *Cost Drop*. During each outer iteration of the multi-iteration version of our Probabilistic approach, an optimization problem is solved. Initially the problem we are going to optimize will have a certain cost. The optimizer will, hopefully, reduce this cost to a lower value. The difference between the initial cost of the optimization problem and the final cost is called *Cost Drop*. As an alternative to the mean squared error, we could use the cost drop, for example stopping the outer loop of our probabilistic algorithm, when the cost drop of the inner optimization problem drops under a threshold. We want to avoid absolute thresholds, since they need to be specifically tuned for each application. Instead, we express this threshold with respect to the initial cost of the iteration: for example we could stop when the cost drop is less than 1% of the initial cost of the problem. This is what we used for our experiments and proved to be a good threshold for obtaining accurate registrations.

Are these two metrics good for our purposes? Which one is the best? First of all, a good termination criterion should behave, more or less, like the difference w.r.t. the ground truth, that is the ideal metric taken as reference. This criterion is satisfied by both our choices, as we will show in Section 5.3.

However, there is a big difference between the two alternatives. The Mean Squared Error has to be specifically calculated after each iteration and is relatively heavy to calculate, since the whole source point clouds has to be traversed. This is not a computationally expensive operation *per se*, but, on

the other hand, the relative cost drop is very fast to compute. Indeed, while solving an optimization problem we already calculate the absolute cost drop, since it is used as termination criterion of the inner loop by the optimization library. Thus, calculating the relative cost drop requires only few mathematical operations: it comes practically for free. For this reason we have chosen to use the Cost Drop as termination criterion: it is very fast to compute and is as good as the Mean Squared Error.

5.3 Experimental Results

To discover the best termination criterion we performed a set of experiments on the Standford Bunny and Bremen datasets, [34, 32]. These experiments have been performed only on two datasets to find a suitable termination criterion. This criterion will the be used to perform all the other experiments presented in this work and thus it has been tested also on the other datasets.

Besides the criteria described in the previous section, we tested also another one: the number of successful steps of the optimization problem. Solving an optimization problem with Levenberg-Marquard is an iterative process. Each step of this process can be successful, if the step managed to reduce the cost of the problem, or, otherwise, unsuccessful. Since also this quantity is very easy to calculate, we wanted to test if it could be used as termination criterion somehow.

In Figure 5.2 we plotted various termination criteria while aligning two point clouds from the Standford Bunny dataset. The transformation between the two clouds was a rotation of 45° around the vertical axis, Figure 5.1. On the x-axis we have the number of the iteration, while on the y-axis we find: the number of successful steps of the "inner" optimization problem, the initial and final cost of the "inner" optimization problem, the cost drop (*i.e.*, the difference between the two previous values), the Mean Squared Error w.r.t. the previous iteration, the Mean Squared Error w.r.t. the ground truth and the discrete derivatives of the last three variables. We plotted also the discrete derivatives because they do a good job at showing when a variable is not changing anymore: when the derivative becomes zero, the value of a variable has stabilized.

We can see that both the cost drop and the MSE w.r.t. the previous iteration have a very similar trend to the MSE w.r.t. the ground truth. Most important, the three values stabilizes more or less at the same iteration. This is particularly obvious if we compare the discrete derivatives: they become almost zero more or less at the same time. Although the MSE w.r.t. to the ground truth keeps decreasing for a few iterations after the other two values

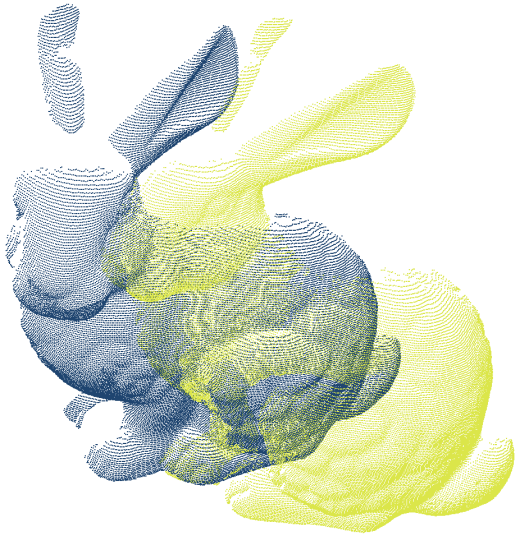


Figure 5.1: Two point clouds from the Standford Bunny Dataset.

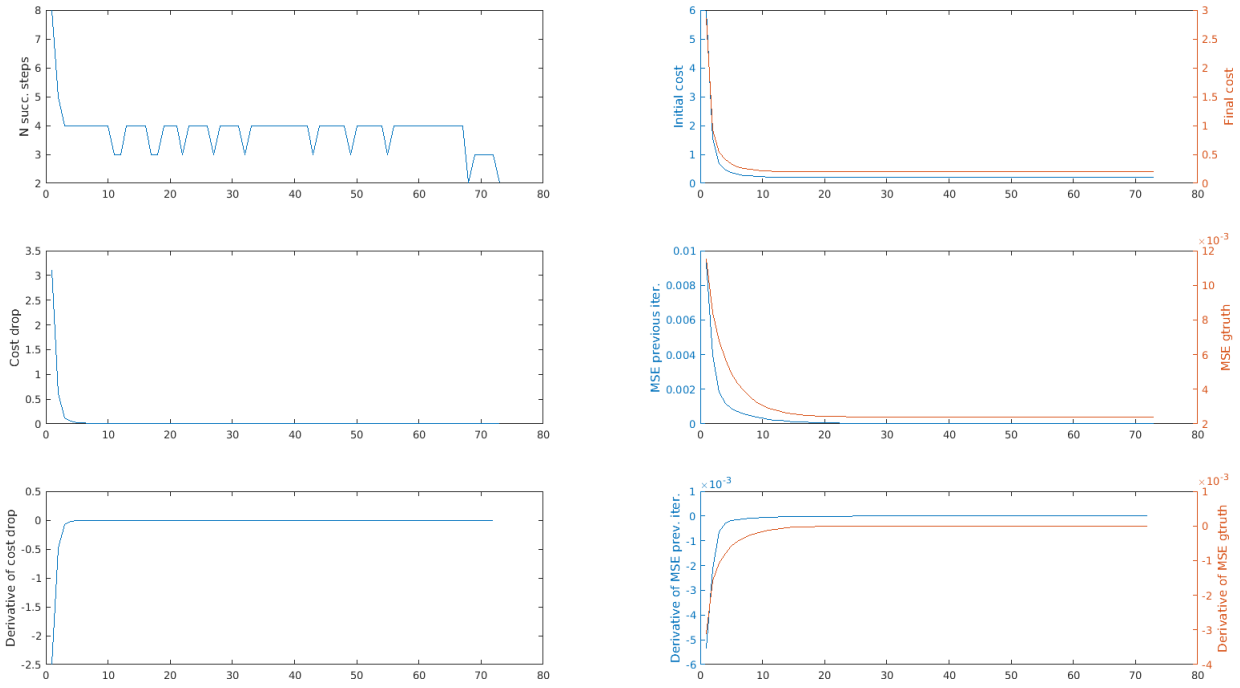


Figure 5.2: Plots of various termination criteria and their derivative for the Standford Bunny dataset.

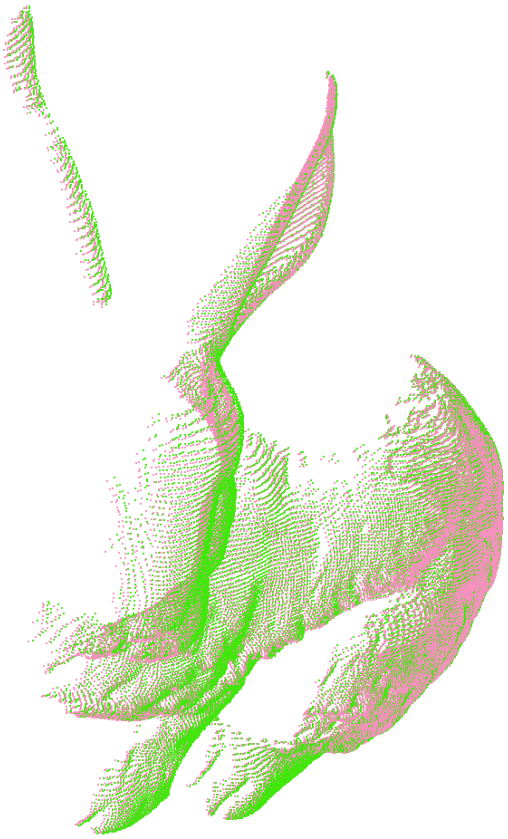


Figure 5.3: The same point cloud aligned with two different termination criteria: a large number of iterations (green point cloud) and our termination criterion based on the cost drop (pink point cloud).

stabilizes, its effect on the quality of the result is negligible. This becomes obvious looking at Figure 5.3, where we have two point clouds, one aligned using a predefined very large number of iteration, the second one using as stopping criterion the cost drop. We can seen that they overlap practically perfectly. The difference between the errors with respect to the ground truth of the two alignments is less than one tenth of the resolution of the point clouds, thus is definitely negligible.

The preliminary idea of using the number of successful steps as termination criterion revealed to be a mistake, since that value oscillates a lot and appears to be not correlated to the MSE w.r.t. the ground truth. For these reasons it was discarded. In Figures 5.6 and 5.7 we show the data we obtained using the Bremen Dataset, to which we applied, respectively, a small rotation, Figure 5.4, and a small translation, Figure 5.5. Also in these cases, we can see that the cost drop stabilizes more or less when also the MSE

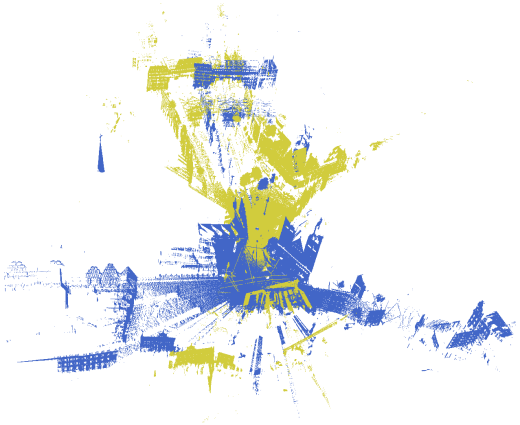


Figure 5.4: Two point clouds from the Bremen Dataset, to which we applied a small rotation.

w.r.t. the previous iteration stabilizes and, anyway, when the cloud has already been moved to the right solution (successive adjustment are negligible compared to the resolution of the point cloud).

The previous experiments collected data on successful alignments, but we did experiments also with clouds that the algorithm was not able to align properly. The reason behind this choice is that we wanted to discover whether the termination criteria were able to stop the algorithm early enough, so that computational time is not wasted.

The first unsuccessful alignment involved two point clouds from the Standford Bunny dataset, whose misalignment is a rotation of 90° around the vertical axis, Figure 5.10. The second one uses the same point clouds, but with a rotation of 180° around the vertical axis. In these case it can be seen that the cost drop converges much earlier than the MSE w.r.t. the ground truth. This behavior, indeed, is good, since it appeared only in unsuccessful alignments, during which, stopping earlier is an advantage (going further would be only a waste of time).

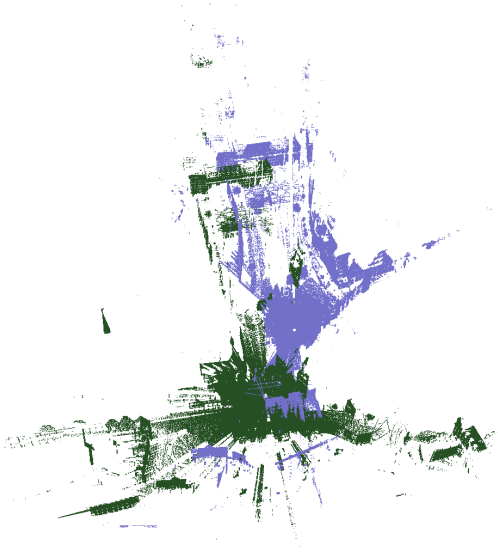


Figure 5.5: Two point clouds from the Bremen Dataset, to which we applied a small translation.

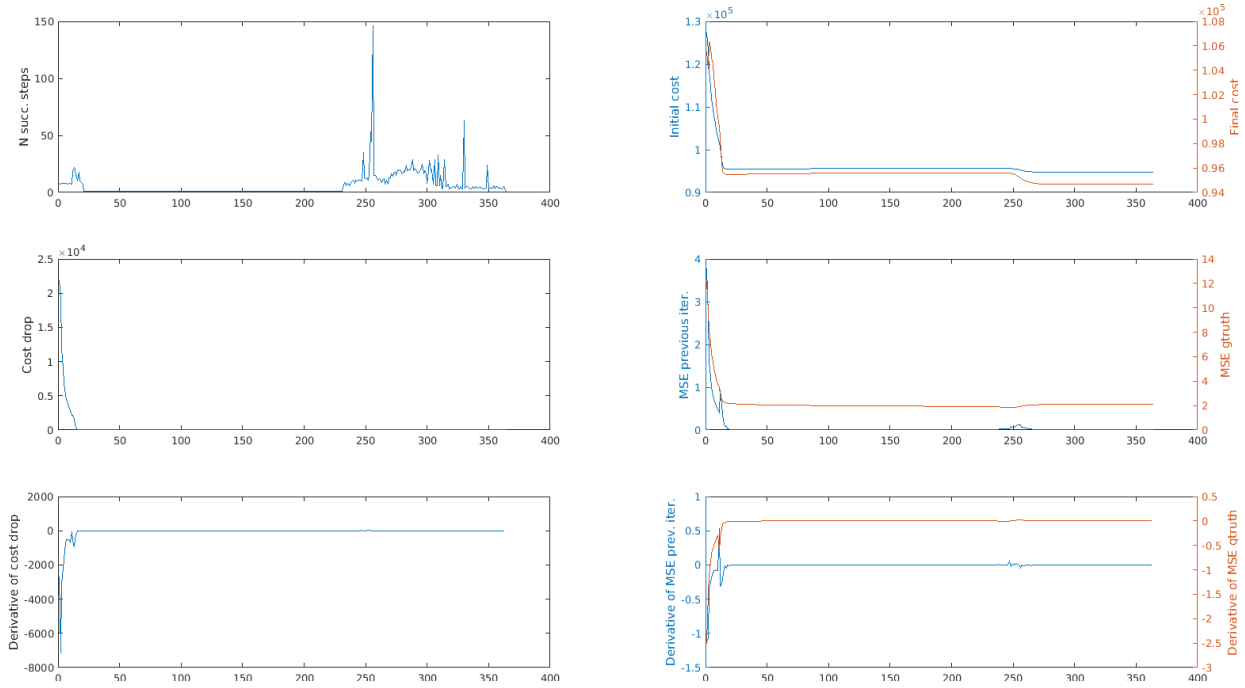


Figure 5.6: Termination criteria for the Bremen Dataset with a small rotation applied.

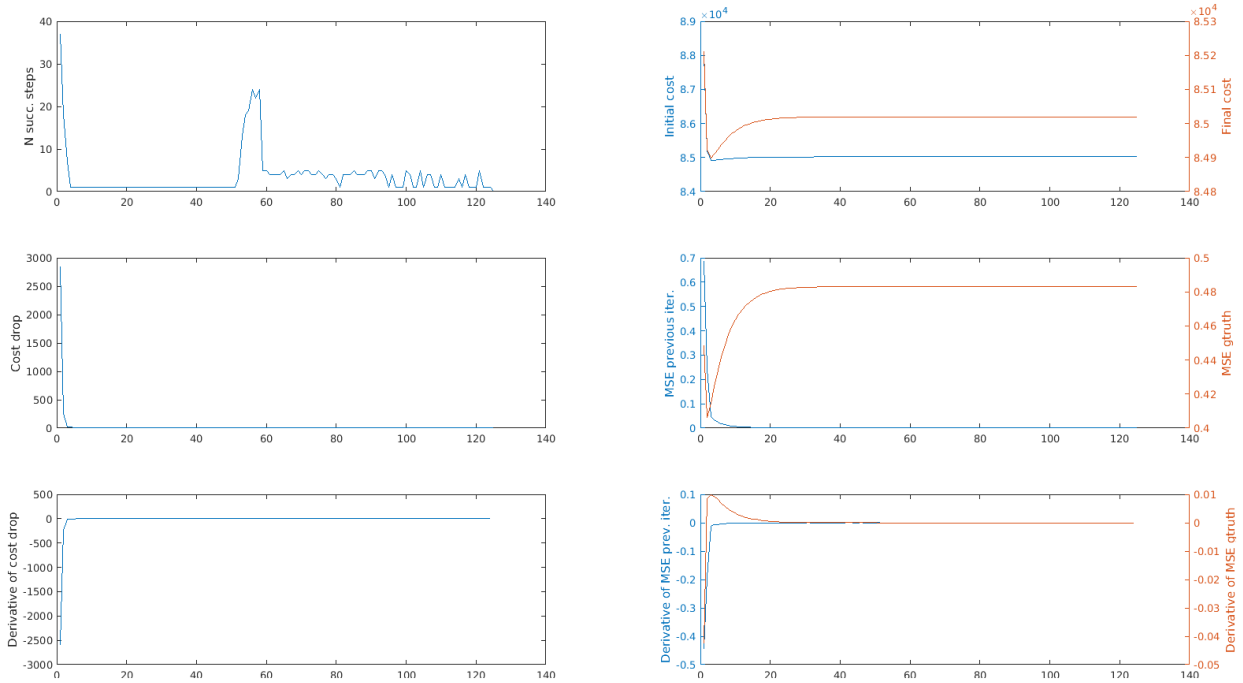


Figure 5.7: Termination criteria for the Bremen Dataset with a small translation applied.

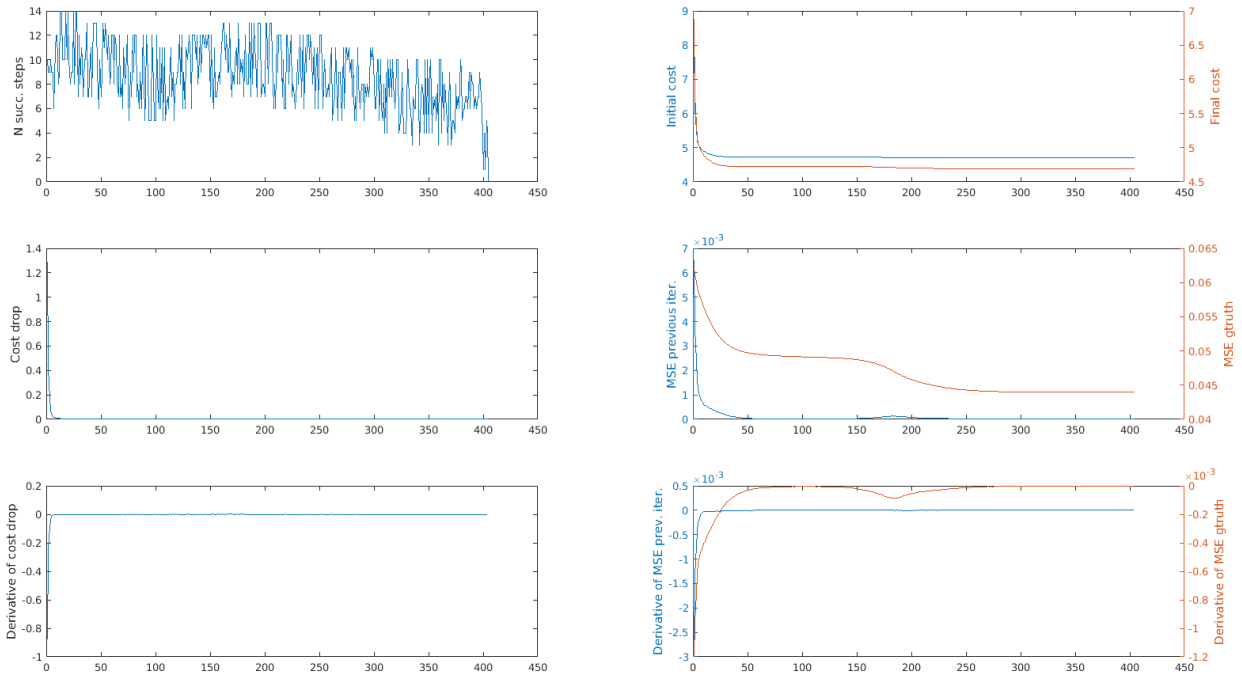


Figure 5.8: Termination criteria for two point clouds from the Bremen Dataset. Initial misalignment of 90°.

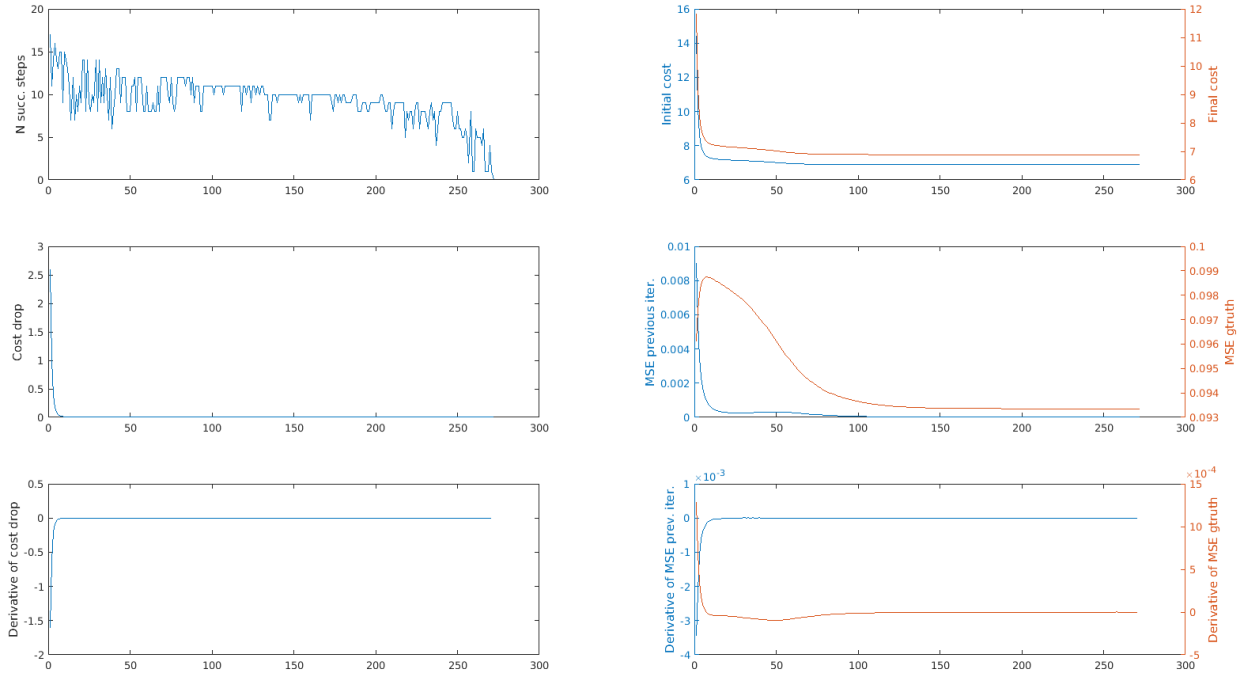


Figure 5.9: Termination criteria for two point clouds from the Bremen Dataset. Initial misalignment of 180°.

Figure 5.10: Two point clouds from the Standford Bunny dataset. Initial misalignment of 90°.

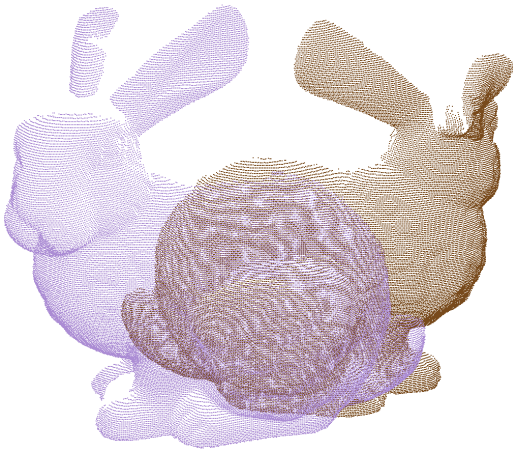


Figure 5.11: Two point clouds from the Standford Bunny dataset. Initial misalignment of 180°.

Chapter 6

Parameters' sensitivity

Even though the current state-of-the-art approaches for point clouds registration work relatively well, we think that they have an important drawback: they depend heavily on fine tuned parameters. We show that even small changes in these parameters can affect negatively the quality of the result. Although this problem is very understudied in the literature, we think that its resolution is a critical step towards robots that are really autonomous: a robot is not really autonomous if its behaviour depends on parameters that have to be carefully tuned for each scenario, especially if no automatic calibration procedure exists.

Since, to our knowledge, there exists no work that studies this problem thoroughly, we decided to conduct a series of experiments to test the sensitivity of the major state-of-the-art algorithms in the field of point clouds registration. These experiments have been conducted using the same datasets described earlier in this work. In this chapter NDT has not been used, since it has a completely different set of parameters. On the other hand, the other algorithms, including our, are all variants of ICP and thus possess parameters that can be compared.

6.1 Maximum Distance

One of the most critical parameters, common to all ICP variants (including the probabilistic approach proposed in this work) is the maximum distance between neighbours. That is, associations whose points lie at a distance greater than a threshold (the maximum distance), are discarded and not used during the optimization process.

In Figure 6.1 we have the results of the experiments on the Hannover datasets for ICP, G-ICP and the Probabilistic approach. The different val-

ues of the maximum distance are on the x-axis, while on the y-axis we have the corresponding mean error w.r.t. the ground truth, calculated as in Section 4.5. As can be seen, there is a small range of values for which the three algorithms perform equally well. However, they differ substantially in how the performances decrease when the parameter is out of this range. Both ICP and G-ICP have a big sudden drop in performances, like there was a threshold value above which they do not provide a meaningful result. Moreover, there are some "random" values for which they provide an unexpected bad result, such as for a maximum distance of 0.4 for G-ICP. The probabilistic approach, on the other hand, performs often better than the other two algorithms and its performances degrade in a much smoother way and without unexpected bad results. This is a very desirable characteristic for a point clouds registration algorithm, because often the best value for the Maximum Distance is unknown. Therefore, it has to be guessed in some way and is often overestimated. With ICP and G-ICP there is the risk that, even choosing a value only a little bit wrong, the result changes substantially. On the other hand, with the probabilistic approach a slightly mis-estimated value for the maximum distance is not going to lead to a catastrophic result.

The situation is different with the Office Dataset, Figure 6.2. The three algorithms perform well with proper values for the maximum distance parameter. While leaving the "optimal area", ICP performs very badly with sudden drops in performance. On the contrary the performances of the Probabilistic approach degrade much more smoothly. G-ICP, instead, is the clear winner of this experiment, since its performances are more or less constant among the range of possible values for the maximum distance parameter. Anyway, with G-ICP there are still some values that lead to unexpected bad results, such as for a maximum distance of 3.5 or 3.6.

With the Linköping Dataset both G-ICP and the Probabilistic approach perform equally well, revealing a very low sensibility to the maximum distance between associated points, Figure 6.3. ICP, instead, performs much worse, but without the big threshold effect it shows with other datasets.

Lastly, with the Bremen Dataset, ICP and the Probabilistic approach perform very similarly in terms of sensibility to the maximum distance parameter, Figure 6.4. G-ICP performs, as usual, very well, but with some bad results randomly distributed.

6.2 Max Neighbors

Using the proposed Probabilistic approach, in addition to the maximum distance between correspondent points, also the maximum number of points

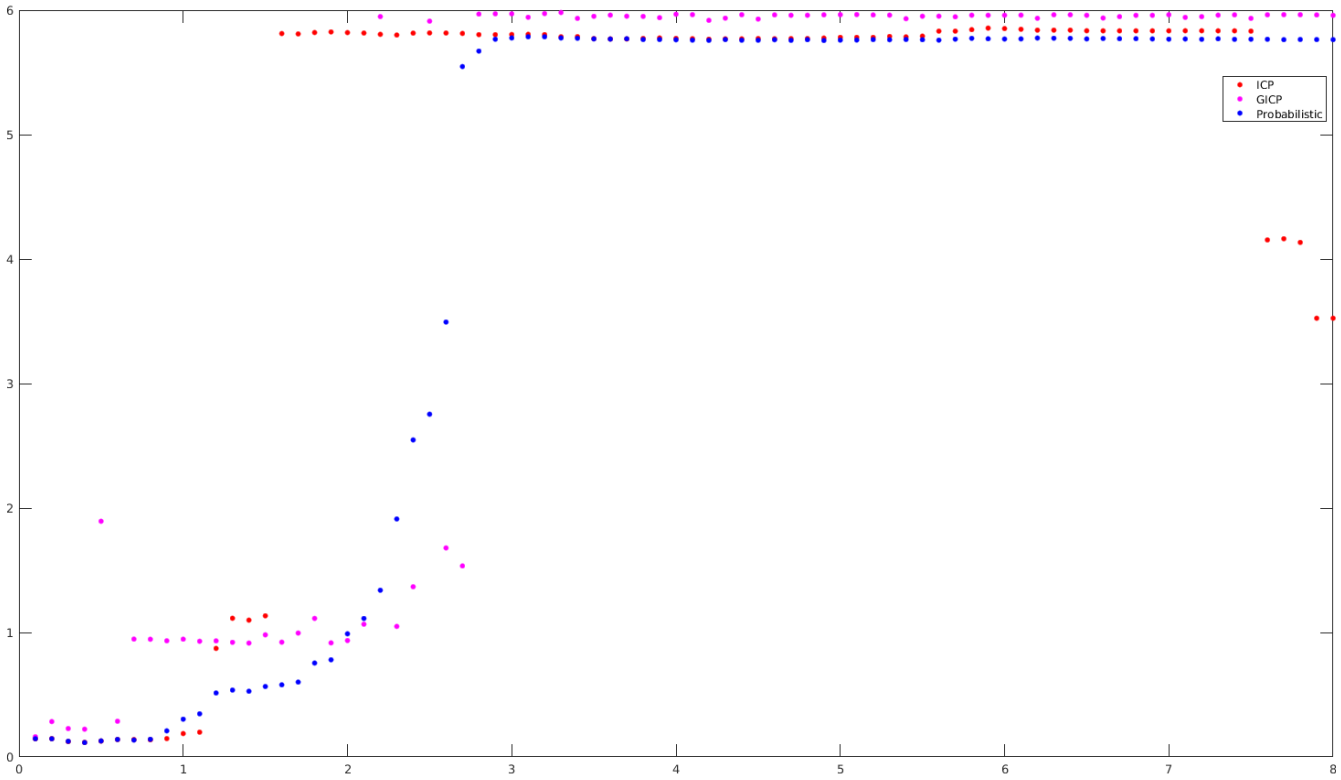


Figure 6.1: How different algorithms performs with different values of the maximum distance parameter on the Hannover Dataset. On the y-axis we have the mean error w.r.t. the ground truth.

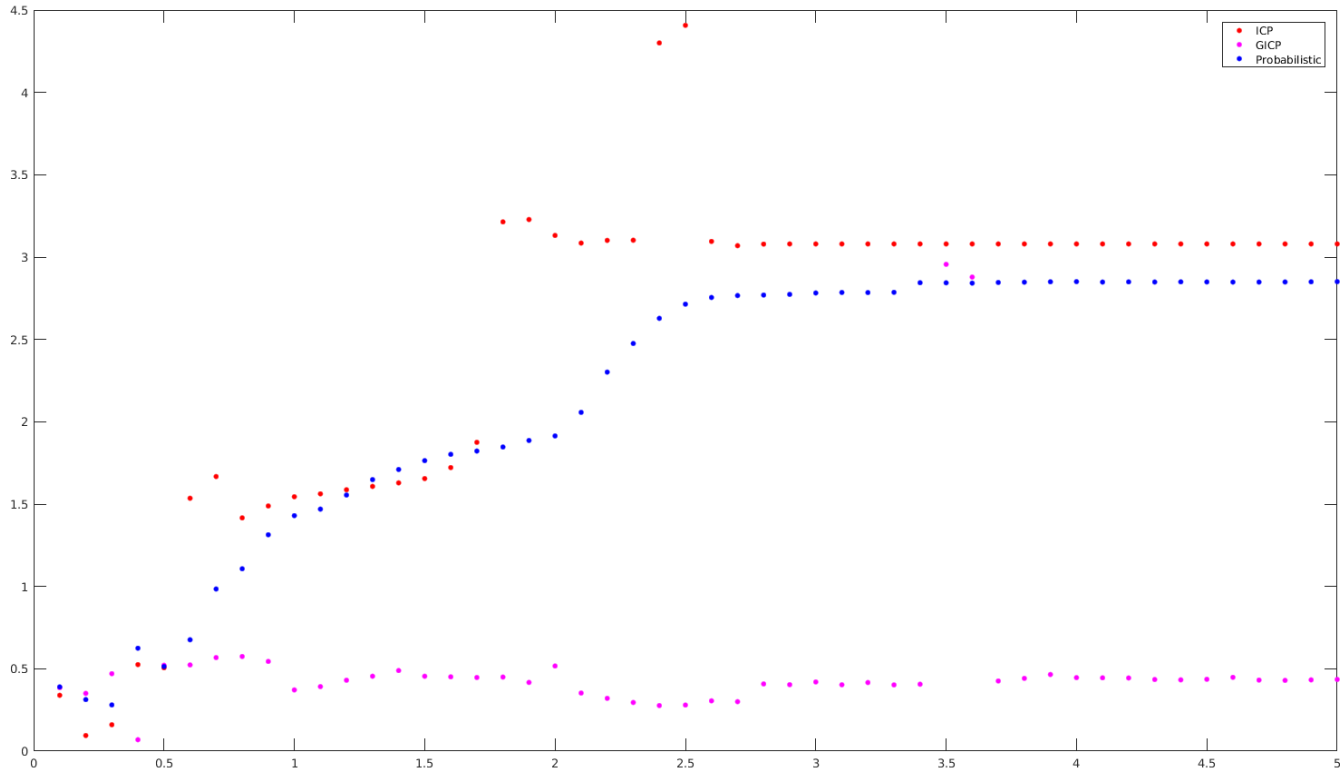


Figure 6.2: How different algorithms performs with different values of the maximum distance parameter on the Office Dataset. On the y-axis we have the mean error w.r.t. the ground truth.

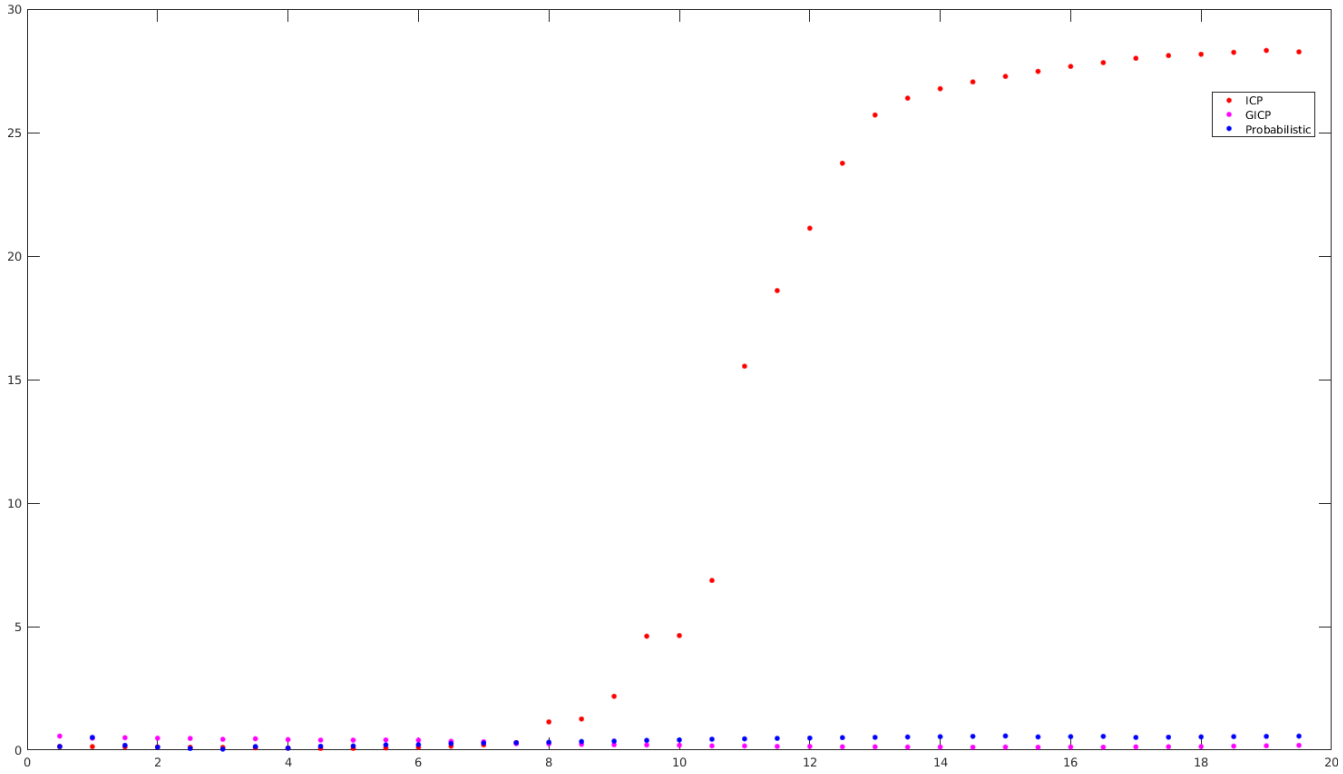


Figure 6.3: How different algorithms performs with different values of the maximum distance parameter on the Linköping Dataset. On the y-axis we have the mean error w.r.t. the ground truth.

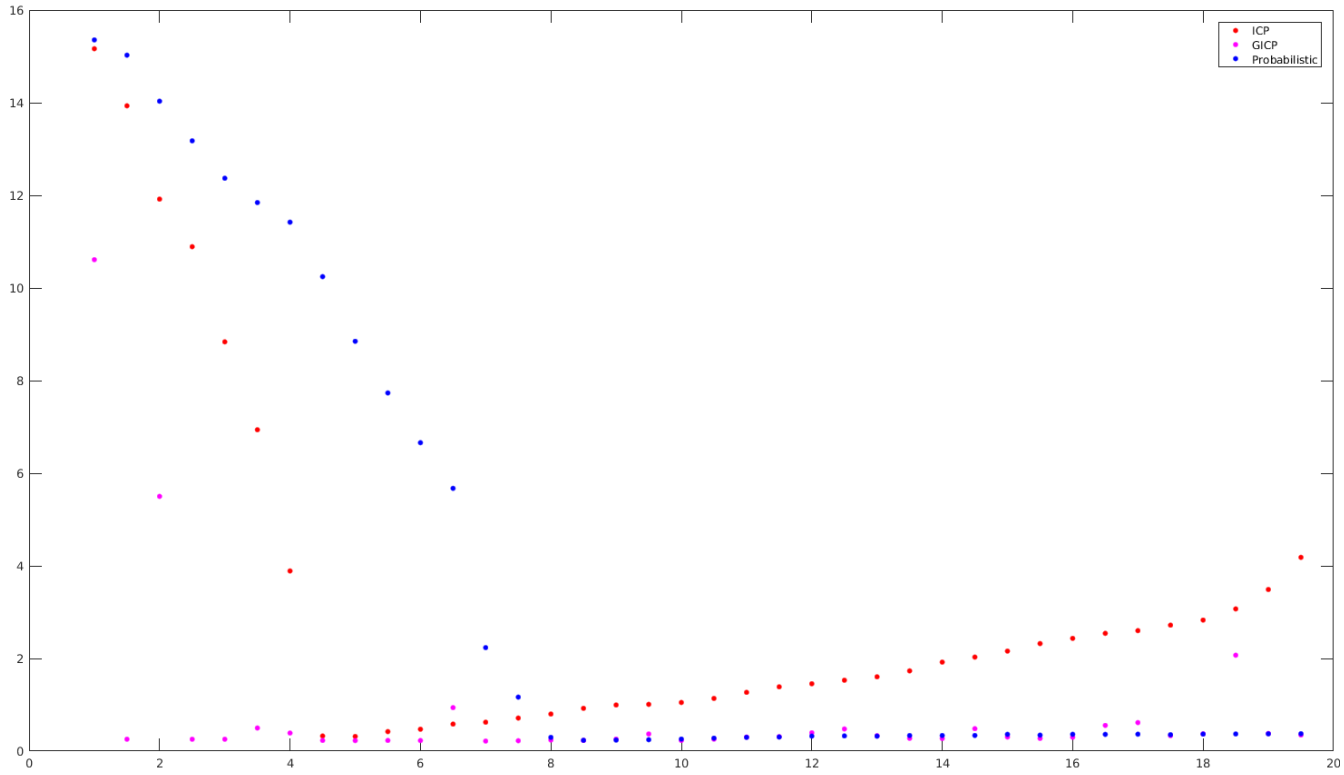


Figure 6.4: How different algorithms performs with different values of the maximum distance parameter on the Bremen Dataset. On the y-axis we have the mean error w.r.t. the ground truth.

associated to a single point in the source point cloud needs to be specified. It is the parameter we called *Max Neighbours* and, of course, it makes sense only for our algorithm, since in ICP and G-ICP the associations are always one-to-one. We tested our approach on the various datasets, with different limits for the maximum number of allowed neighbours. The maximum distance between correspondences has been set to infinite, so that this parameter does not interfere with the tests excluding some correspondences. In a real scenario, the correspondences used by the optimization step will be limited by the max distance parameter, the maximum neighbours parameter or a combination of both.

On the Office and Linköping datasets, Figures 6.5 and 6.6 the maximum number of neighbours does not affect substantially the quality of the result. This is a very valuable characteristic, since it means that the probabilistic weights do a good job at soft-filtering the wrong correspondences. The same considerations could apply also to the Hannover and Bremen datasets, Figures 6.7 and 6.8. On these datasets worse results are obtained only for very low value of the parameter. In both cases the result stabilizes when the maximum number of neighbours is over fifteen.

In conclusion, we could say that the maximum number of neighbours is not critical parameter and thus does not need to be fine tuned.

6.3 Sub-sampling

In order to perform the experiments in this work, the point clouds were always sub-sampled with a voxel grid. This means that a 3D grid is fitted to the point cloud and that each point in a particular box is approximated with the centroid of the box, Figures 6.9 and 6.10.

The 3D grid is composed of several cubes of the same size. The size of the side of a cube is called leaf-size and is a measure of the level of sub-sampling applied. The greater the leaf-size, the bigger will be the cubes of the 3D grid and thus the more the cloud will be down-sampled. Sub-sampling is useful for several reason. First of all it greatly reduces the execution time of the algorithm. Usually a point cloud contains much more points than needed to accurately perform a registration, *i.e.*, it is too dense. Since the execution time of all the tested algorithms is a function of the number of associations used, reducing the number of points used is a way of reducing the computation time. Moreover sub-sampling could also be a way to reduce noise in a point cloud. Since each voxel is approximated with its centroid, if a voxel contains a few outliers caused, perhaps, by sensor noise, they would be easily filtered out. Of course sub-sampling too much could eliminate important in-

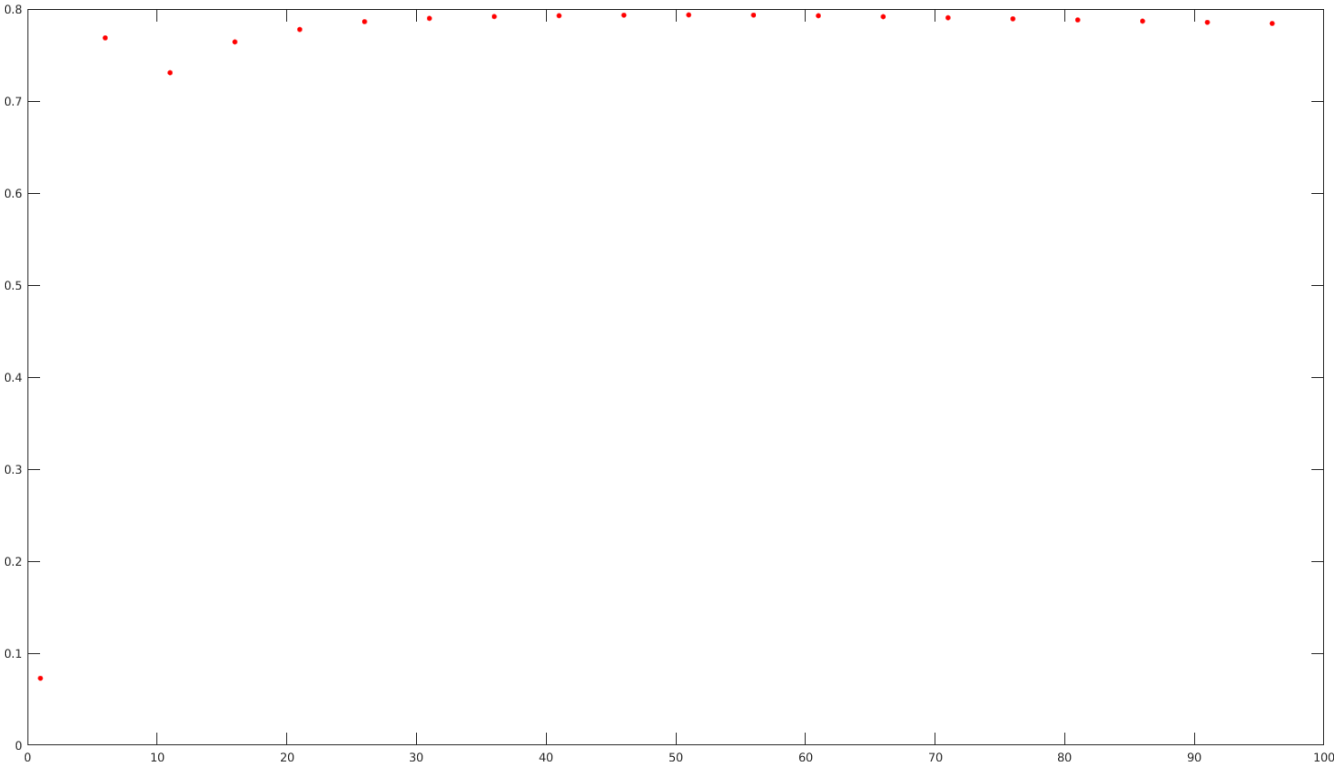


Figure 6.5: How the Probabilistic approach performs with different values of the Max Neighbours parameter on the Linköping Dataset. On the y-axis we have the mean error w.r.t. the ground truth, on the x-axis the maximum number of neighbours

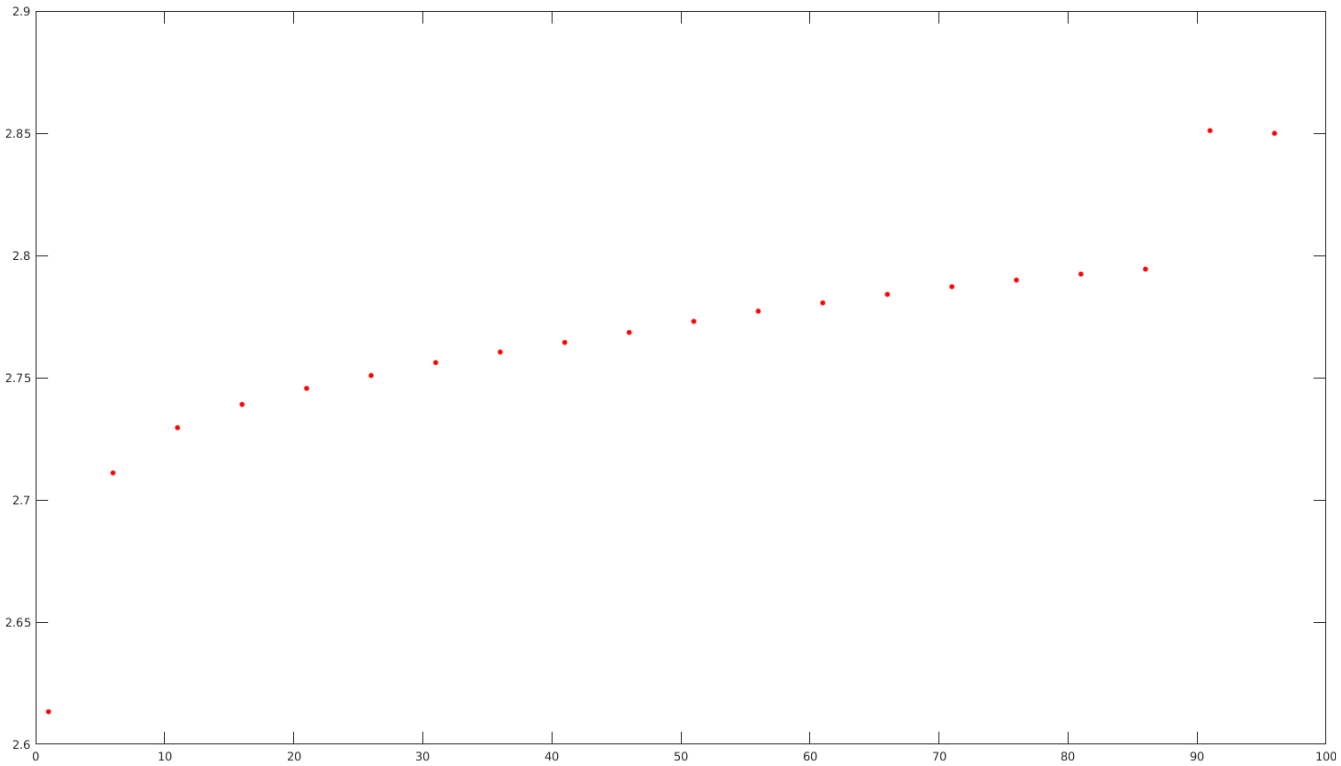


Figure 6.6: How the Probabilistic approach performs with different values of the Max Neighbours parameter on the Office Dataset. On the y-axis we have the mean error w.r.t. the ground truth, on the x-axis the maximum number of neighbours

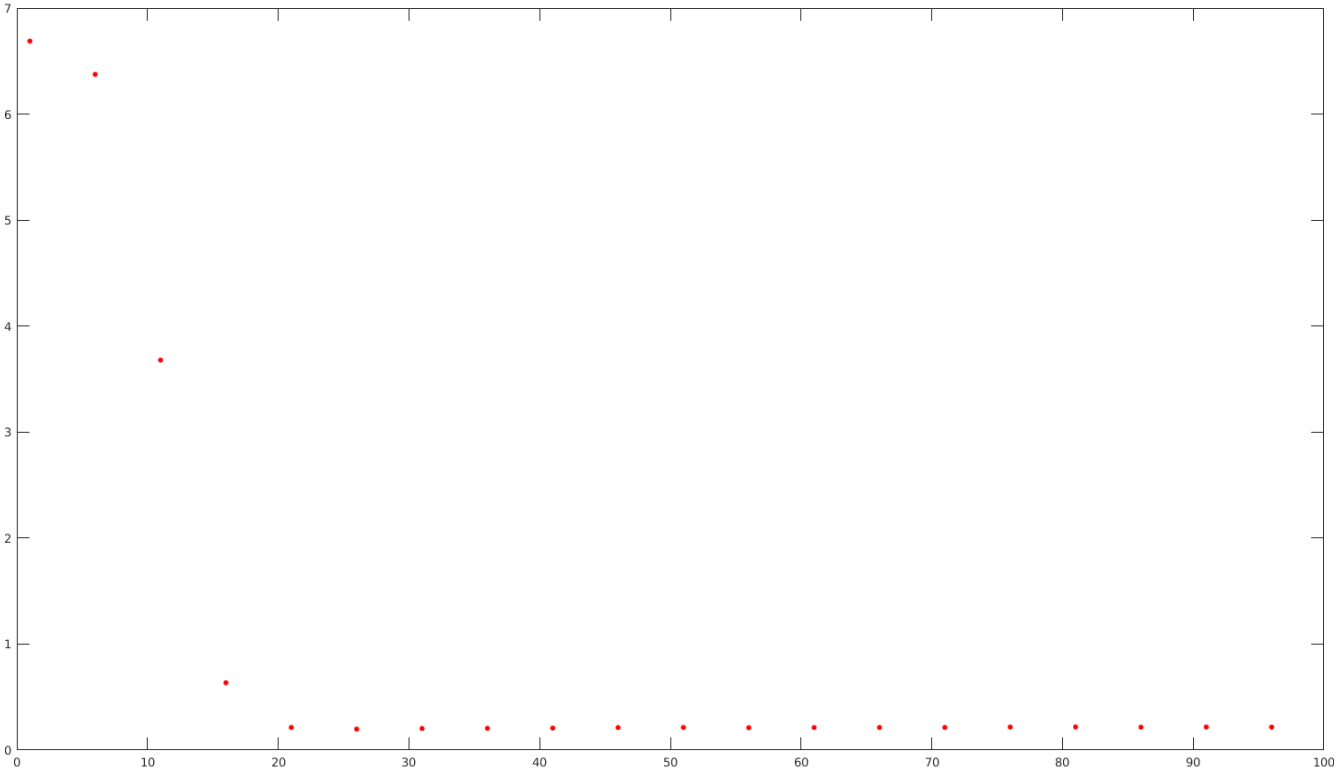


Figure 6.7: How the Probabilistic approach performs with different values of the Max Neighbours parameter on the Hannover Dataset. On the y-axis we have the mean error w.r.t. the ground truth, on the x-axis the maximum number of neighbours

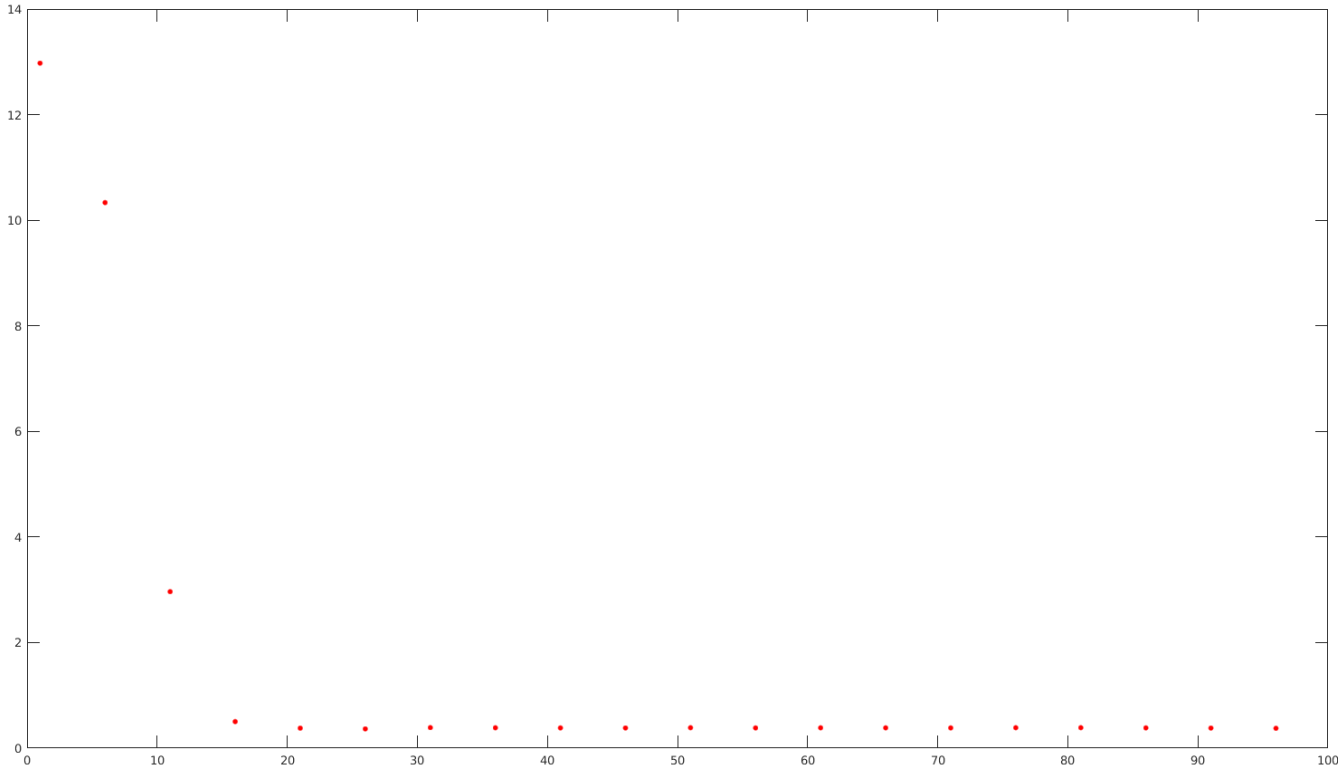
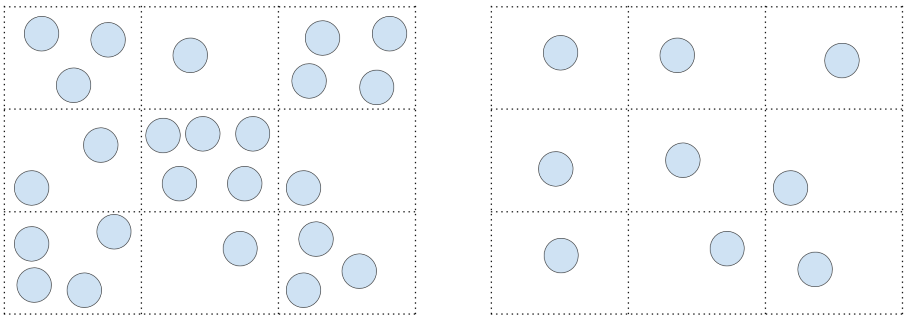
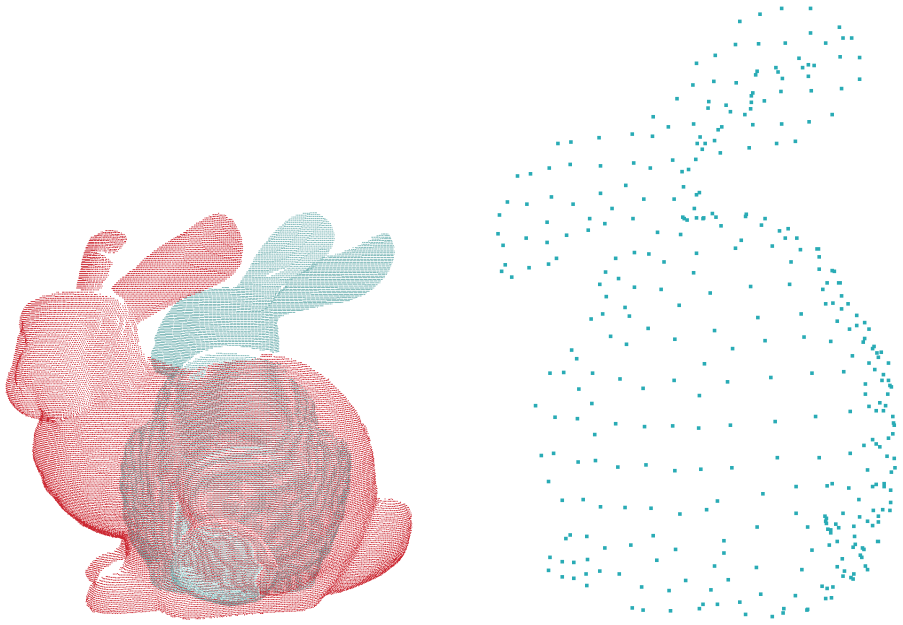


Figure 6.8: How the Probabilistic approach performs with different values of the Max Neighbours parameter on the Bremen Dataset. On the y-axis we have the mean error w.r.t. the ground truth, on the x-axis the maximum number of neighbours.



(a) (b)

Figure 6.9: A point cloud and its sub-sampled version. In (b) the points in a squared have been replaced with their centroid.



(a) (b)

Figure 6.10: (a) The Standford Bunny and (b) a subsampled version.

formation and decrease the performances of the registration algorithm. For this reason finding an appropriate value for the leaf-size is a crucial step.

We tested the three algorithms on the datasets, recording the mean error w.r.t. the ground truth with increasing values of leaf-size, keeping all the other parameters unaltered.

With the Hannover dataset, Figure 6.14, both ICP and G-ICP got very unpredictable results, with a small difference in the leaf-size making a huge difference on the quality of the result. The Probabilistic approach, instead, behaves much better, except for very high values of leaf-size.

On the Linköping dataset, Figure 6.12, both G-ICP and the Probabilistic approach got very good results, showing an high degree of independence from the leaf-size. This independence was not shown, instead, by ICP, that, nevertheless, got bad results only for high values of leaf-size. We can conclude that, on the Linköping dataset, all the three algorithms got good results.

On the Office dataset, Figure 6.13, the Probabilistic approach got uniform results among the space of possible values of leaf-size. On the other hand, the other approaches got very scattered plots, showing a high sensibility to the value of the leaf-size parameter.

Lastly, on the Bremen dataset, Figure 6.14, as opposed to the other datasets, ICP and G-ICP provided better results and less sensitivity to the

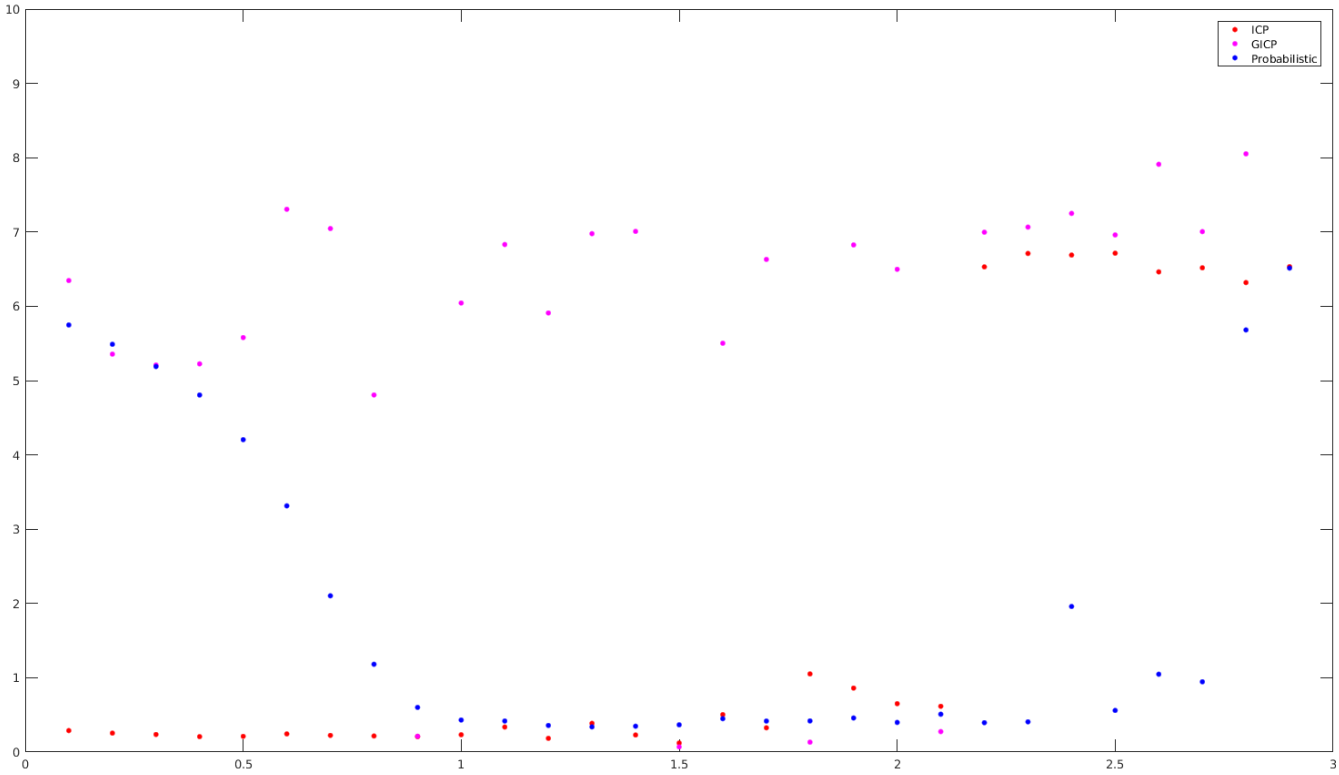


Figure 6.11: How different algorithms performs with different values of leaf-size on the Hannover Dataset. On the y-axis we have the mean error w.r.t. the ground truth, on the x-axis the value of the leaf-size.

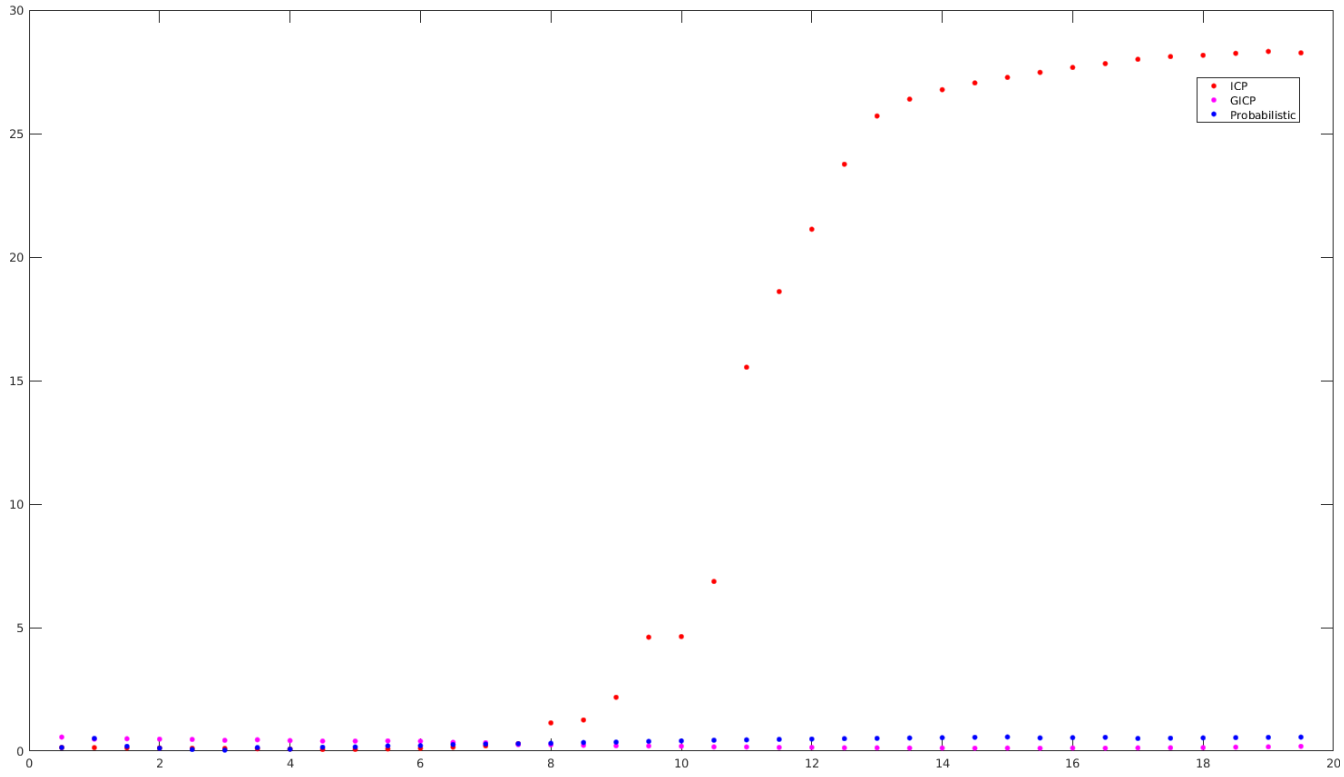


Figure 6.12: How different algorithms performs with different values of leaf-size on the Linköping Dataset. On the y-axis we have the mean error w.r.t. the ground truth, on the x-axis the value of the leaf-size.

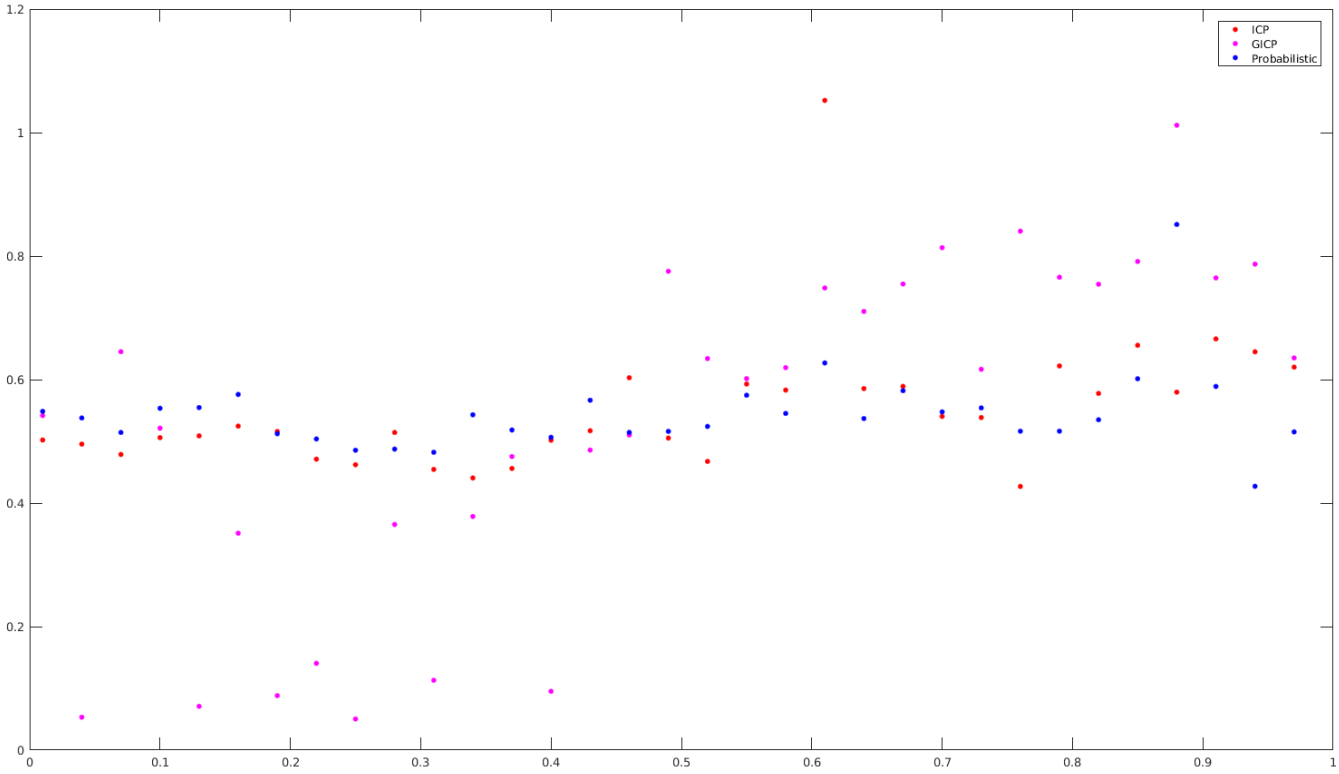


Figure 6.13: How different algorithms performs with different values of leaf-size on the Office Dataset. On the y-axis we have the mean error w.r.t. the ground truth, on the x-axis the value of the leaf-size.

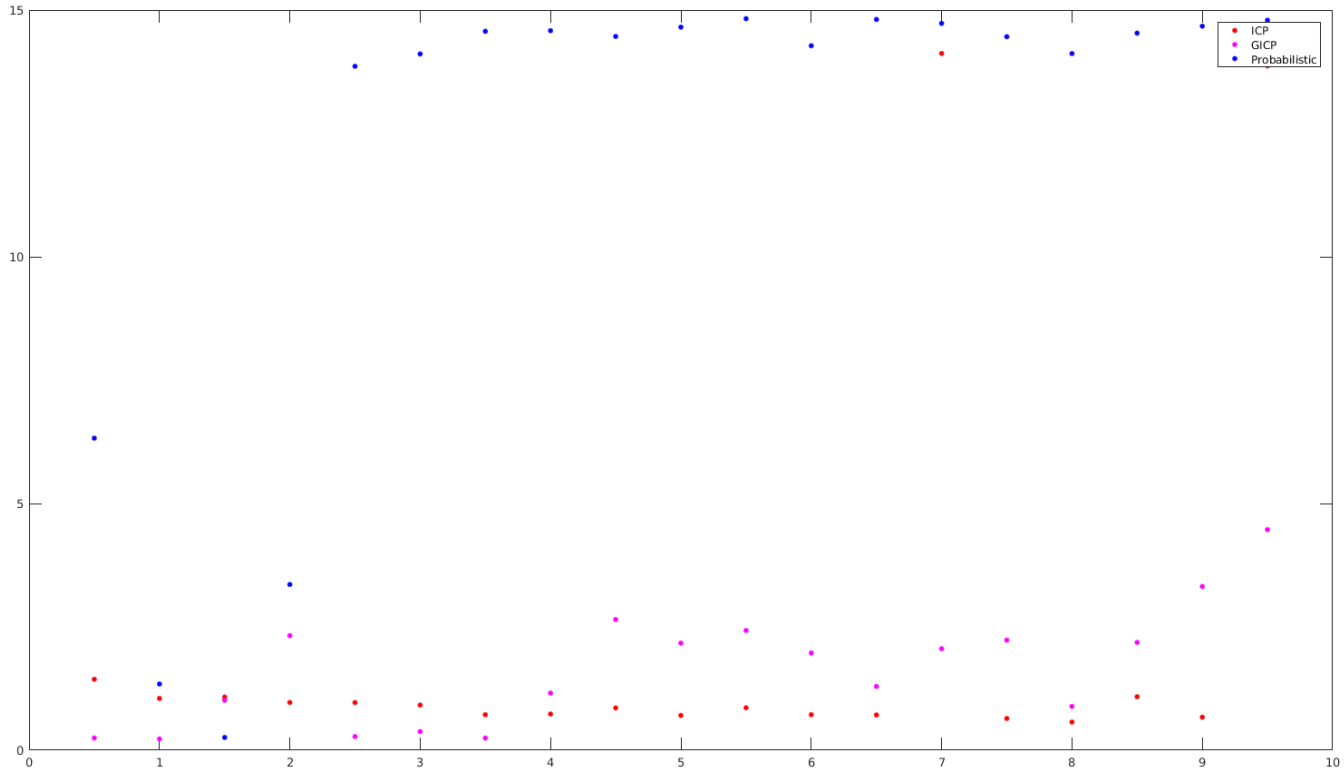
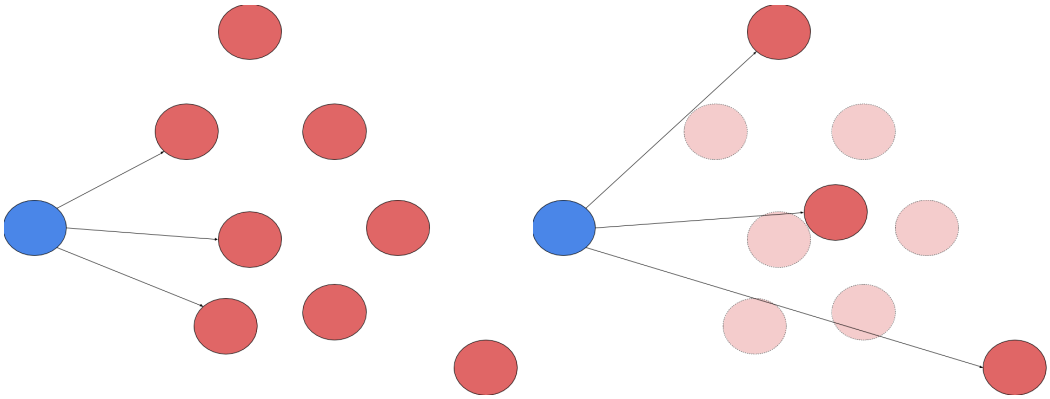


Figure 6.14: How different algorithms performs with different values of leaf-size on the Bremen Dataset. On the y-axis we have the mean error w.r.t. the ground truth, on the x-axis the value of the leaf-size.



(a) (b)

Figure 6.15: How the data association changes with different levels of sub-sampling. Semi-transparent points are transparent and have been replaced with their centroid.

leaf-size parameter.

Regarding the Probabilistic approach and the size of the leaves of the voxel filter, an important consideration must be done. Indeed, the effect of this parameter cannot be completely separated from the effect of the other two parameters, the maximum distance between associated points and the maximum number of neighbours. Given a fixed number of neighbors, reducing the resolution of the target point cloud, has the effect of associating farther points to a particular point in the source point cloud. This happens as long as the farther points are within the maximum distance threshold. In Figure 6.15a we have a point in the source point cloud (coloured in blue), associated to the three closest points in the target cloud (colored in red). In Figure 6.15b the target point cloud has been sub-sampled: some points have been removed (transparent points) and have been replaced with their centroid. Since the max neighbours parameter remains constant, farther points are associated to the point in the source point cloud.

6.4 Conclusions

We showed that the Probabilistic approach, in most cases, is less influenced by a fine-tuning of the parameters than other state of the art approaches (ICP and G-ICP). This is a very valuable characteristic in robotics applications, where a fine tuning of the parameters could be impractical or even impossible. These parameters, indeed, should be fine tuned in every new scenario limiting

greatly the degree of autonomy of the robots on which they are used.

Chapter 7

Global Point Clouds Registration

7.1 Introduction

State of the art point clouds registration algorithms, such as ICP (and its variants) or NDT, suffer from a very important drawback: they use local optimization algorithms. This implies that, in order for the algorithms to converge to the right solution, the two point clouds should already be roughly aligned. Otherwise, the algorithm would probably converge to a local minimum instead than to the global one. Indeed, point clouds registration problems often have many sub-optimal local solutions, see Figure 7.1. This is a well know problem that the research community has tried, and still is trying, to tackle. One way to deal with this issue is to use some kind of features (see Section 2.2), but as was explained, we want to avoid using geometric features, in order to make our technique as general as possible. Indeed, calculating 3D descriptors of salient points requires the normal to the surface on which the point lies. However, the normal to a surface cannot be correctly estimated when dealing with very sparse point clouds, because they could be locally too sparse to represent the local surface in an informative way.

Instead, we used a completely different approach: rather than using mathematical optimization techniques (that could find only local minima), such as Levenberg-Marquardt or the SVD decomposition (which are used, respectively, in G-ICP and ICP), we opted for a soft computing optimization technique: Particle Swarm Optimization (PSO), [40]. While it is not guaranteed to find the global optimum, it proved to be very good at escaping from local minima, therefore it is particularly suitable to our application. Moreover, it has been designed to work on continuous search spaces, such as our and is

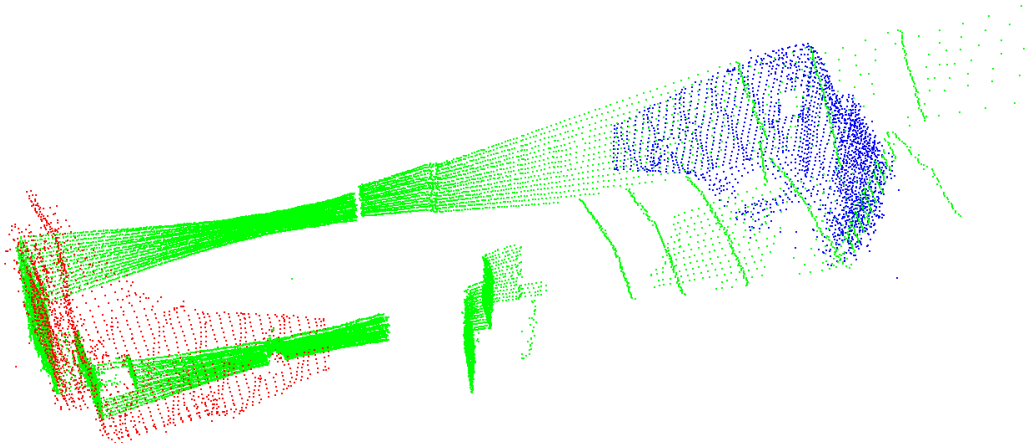


Figure 7.1: A failure due to an algorithm converging to a local minimum. The blue point clouds is the solution found, the red one is the right solution. The minimum found by the algorithm is actually very good, nonetheless is wrong. However, this pair of point clouds have many different possibly good alignments and there is no way for an algorithm to discover the best one. The problem is inherently ambiguous. For this reason the corridor dataset was not used to perform the experiments in this chapter.

very easy to implement and adapt to specific problems.

7.2 Metrics

Before finding a solution to the global point clouds registration problem, we must answer a fundamental question: how do we measure how well two point clouds are aligned? Unfortunately, in many practical situations, there is no exact answer to this question. If we would know the exact point associations, i.e. which point in the source point cloud corresponds to a particular point in the target point cloud, the correctness of the alignment could be easily calculated, for example using the sum of the squared distances between associated points. This is actually the method we used to calculate the distance between a solution and the ground truth in the preceding chapters. In that case it was possible because the cloud and the ground truth are actually the same point cloud, only displaced in the space. Thus, simply the $i-th$ point in the point cloud corresponds to the $i-th$ point in the ground truth.

Unfortunately, in a realistic situation, the associations are not known and, without using some kind of feature, there is no way of estimating them. Moreover, it could happen, and in some applications it hopefully will, that

the target and source point clouds do not overlap perfectly, so that some points could not have any associated point in the other cloud. Therefore, it would be also necessary to know which parts of the clouds overlap and which not. This information is not usually available either. Finally, exact point associations often do not exists at all! Because of the functioning of some sensors, it might happen that the correspondent of a particular point in a cloud could lie between two or more points in the other cloud. Which point should we use as correspondence?

These problems, of course, are common also to state-of-the-art (local) point clouds registration algorithm. The most common solution, used for example in ICP, is to use a greedy approximation of the "true", unknown, data association. For each point in the source point cloud, we take the closest point in the target point cloud. With closest, as usual, we mean the point with the smallest Euclidean distance. Thus, the sum of the squared distances becomes our metric for evaluating the goodness of the alignment. Suppose that we have two point clouds. X and Y, with point y_i in Y being the closest point to x_i in X, the metric M used to evaluate the goodness of the alignment given by the application of a rotation R and a translation T is given by Equation (7.1).

$$M(R,T) = \sum_{i=0}^{N} (R \cdot x_i + T - y_i)^2 \tag{7.1}$$

Where N is the cardinality of X and Y. This formulation implies that each point in X has a correspondent in Y. In case this assumption was not true, we would simply use only the points in X with a correspondent and discard the others.

This metric has been extensively used by many other approaches and behaves well when the two point clouds are already roughly aligned. We want to discover whether it can be used also for global registration.

The first, and most important, issue is to check whether the minimum of this metric corresponds also to the best alignment. In case this assumption does not hold, no matter how good the optimization algorithm is, it will not be able to consistently find the best solution.

The second issue to deal with is the "shape" of the function. Even though the global minimum corresponds to the best alignment, there will probably be many other local minima, perhaps corresponding to completely wrong solutions. In Figure 7.1 we can see an example of this kind of failure due to an algorithm converging to a local minimum. The blue point clouds is the solution found, the red one is the right solution. The alignment found is actually a good alignment, but is globally wrong. However, this pair of point

clouds have many different possibly good alignments and there is no way for an algorithm to discover the best one. The problem is inherently ambiguous. For this reason the corridor dataset was not used to perform the experiments in this chapter.

We have to check how many global minima and, most important, how good those minima are. Indeed, if there are many, very good, local minima, the optimization algorithm is more likely to get stuck in a sub-optimal solution. Most global optimization algorithm are, in fact, heuristic algorithms that are not guaranteed to converge to the global minimum, [41]. Even though they are usually able to escape from local optima, the greater the number and, most important, the quality of the minima is, the higher the probability to converge to a sub-optimal solution, that could be even very wrong.

Since we want to estimate a rigid transformation in space, that is a roto-translation, the space of possible solutions to our problem has six dimensions, three for the translation and three for the rotation. For this reason, the metric function cannot be plotted, making answering to the previous two questions much harder. Nevertheless, we can reduce the dimensionality of the problem to just two dimensions (for example two angles of rotations) and plot the obtained "sub-function". This plot can be used as necessary condition for the two requisites: if the reduced plot does not satisfy the requirements, for sure neither the complete function is going to satisfy them, since the studied problem is a simplified form of the complete one.

Besides the sum of squared distances, we wanted to test also another metric, so to find which one is the best. This metric is the median of the set of squared distances, as in Equation (7.2), where N is the size of the source point cloud and x_n is the n^{th} point in a cloud, given that the points are ordered according to the distance from their correspondent in the other cloud. The distances are calculated in the same way as in ICP: for each point in a cloud, the closest point in the other cloud is taken as correspondent point and their distance is calculated.

$$M(R,T) = \begin{cases} (R \cdot x_{\frac{N+1}{2}} + T - y_{\frac{N+1}{2}})^2 & \text{if } N \bmod 2 = 1 \\ \frac{(R \cdot x_{\frac{N}{2}} + T - y_{\frac{N}{2}})^2 + (R \cdot x_{\frac{N}{2}+1} + T - y_{\frac{N}{2}+1})^2}{2} & \text{otherwise} \end{cases} \tag{7.2}$$

Moreover, for both metrics, we tested also a variant where we discard from the computation all the correspondences whose distance is greater than three times the median distance and less then one third of it. We called these variants "robust" because they aim at removing outliers from the set.

The reason behind this idea is that, if two point clouds are properly aligned and thus the correspondences are correct, the distances between correspondent points should all be more or less the same. Distances very different from the others are more likely to be related to outliers, *i.e.*, wrong correspondences that should be filtered out. The use of the median distance in this way, rather than, for example, the average, is well studied in the literature, [42]. Moreover, we show that changing the factor to which we multiply the median (in our case three), does not modify the result in a substantial way.

For the tests we used some publicly available datasets, in conjunction with data recorded by us. These are:

1. The *Bremen Dataset*, [32].
2. The *Hannover Dataset*, [33].
3. The *Linköping Dataset*, [13].
4. The *Office Dataset*, recorded by us, [13].

For each of these datasets we had a ground truth. Thus, we started with a pair of perfectly aligned point clouds and progressively moved one with respect to the other, while calculating the value of the various proposed metrics.

In Figure 7.2 we can see the value of the median squared distance between two overlapping point clouds taken from the Bremen Dataset. On the x-axis we have the yaw between the two clouds, on the y-axis the pitch, both between 0° and 360°. We can see that this metric respects the first constraint, *i.e.*, the minimum value coincides with the best solution, that is when the angles are either 0° or 360°. Anyway, there is still an important issue: it has many local minima that are very close, in value, to the global one, but correspond to very wrong alignments. This is easily visible from the side view in Figure 7.2b. This is a big problem for the optimization algorithm that, although it has some abilities to escape from local minima, it will hardly converge to the right solution with this great number of sub-optimal, yet very close in value to the minimum, solutions. Moreover, it has to be considered that we are plotting only a simplification of the real metric function; in reality the situation can get only worse than what depicted.

The plot of the *robust* median squared distance, Figure 7.3, is very similar to that of the *bare* median. The plot of the sum of squared errors has the same problems of the latter, but with an extra drawback: the global minimum of the function does not correspond to the best alignment, but is shifted of a

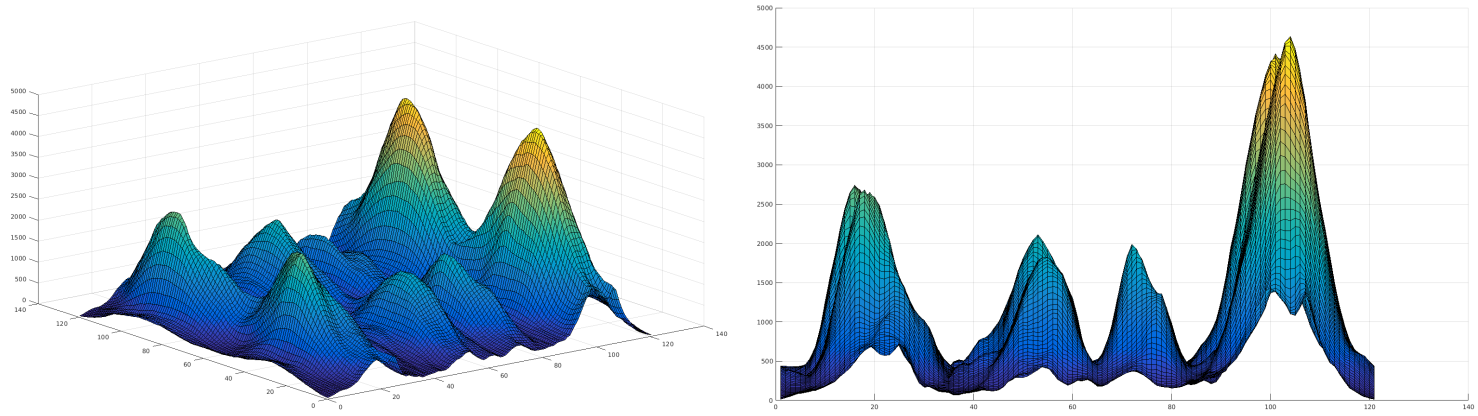
few degrees. This is particularly visible in the side view in Figure 7.3b. In Figures 7.5 and 7.6 we can see the plots of the robust sum of squared errors and its *normalized* version, where the sum is divided by the number of points that have not been filtered out. The concept behind this metric is that we want to reward the alignments that best aligns the greatest number of points and give a lower score to those aligning very well only a few number of points. This is the main danger of the *robustified* version, since because it does not use the supposed outliers for calculating the score, it could give a very good score to very wrong alignments that, nevertheless, align well a small number of points (those not filtered out).

From the plot of the simplified experiments, there seem to be no real reason to prefer one metric to the other. But the situation could get completely different with the full 6-DOF error function. Indeed, we will see that, in practice, the results can get really different depending on the metric used.

Of course the results shown cannot be generalized. There could exist a pair of point clouds for which the proposed metrics behave very differently from what is depicted in this work. For this reason we tried to test many different cases, so to cover many different situations and setups.

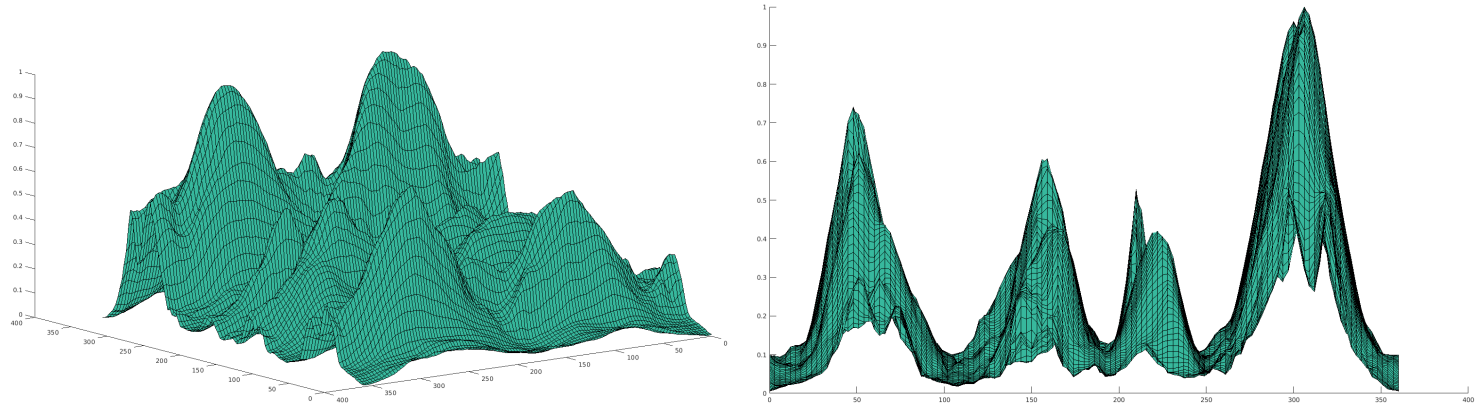
For the Office Dataset, the differences between the metrics are very subtle, at least for the 2-DOF simplification we are using. For this reason we plotted together many metrics, to better show the subtle differences. In Figures 7.7 and 7.8 we plotted the three variants of the sum of squared errors and the median based metrics together. Of course we had to normalize the functions, since they have very different scales and, otherwise, could not be plotted together in a meaningful way (for example the sum of squared errors is for sure several order of magnitude greater than the median). As can be seen, the metrics are very similar, once normalized. They all behave relatively well, without a great number of local minima and with the global minima in the right position. Again, from the plots there is no hint on which metric to choose. In Figures 7.9 and 7.10 we plotted the metrics for the Linköping dataset, to whom the same considerations of the Office dataset can be applied.

Of course restricting our search space to only two angles is not the only way to reduce the dimensionality of the problem. For example we could test how the metrics perform when the two point clouds have been displaced by a translation along two axis. This is what we did in Figures 7.11 and 7.12 for the Bremen Dataset, where we applied a translation on the x and y axis, ranging from 0 to a dataset-based maximum (in the case of the Bremen Dataset, 30 meters), with a step size also dependent from the dataset (1 meter in this case). Since the datasets have very different scales, testing on the same range for each one, like we did for the angles, would not make any



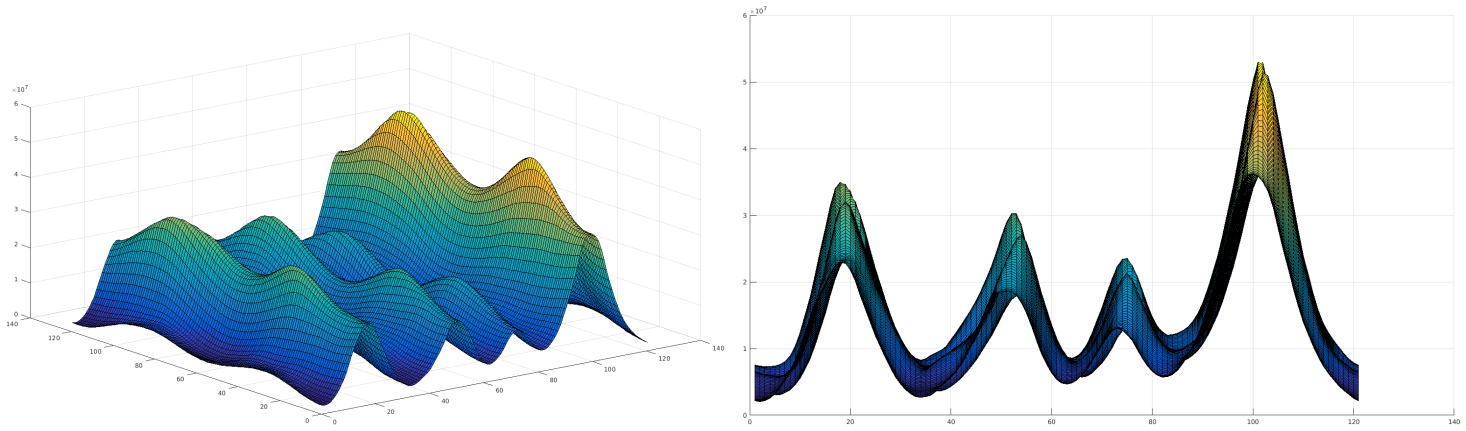
(a) (b)

Figure 7.2: Median Squared Distance, Bremen Dataset. The 3D graph of how the median squared distance behaves varying the pitch and the yaw between the two point clouds.



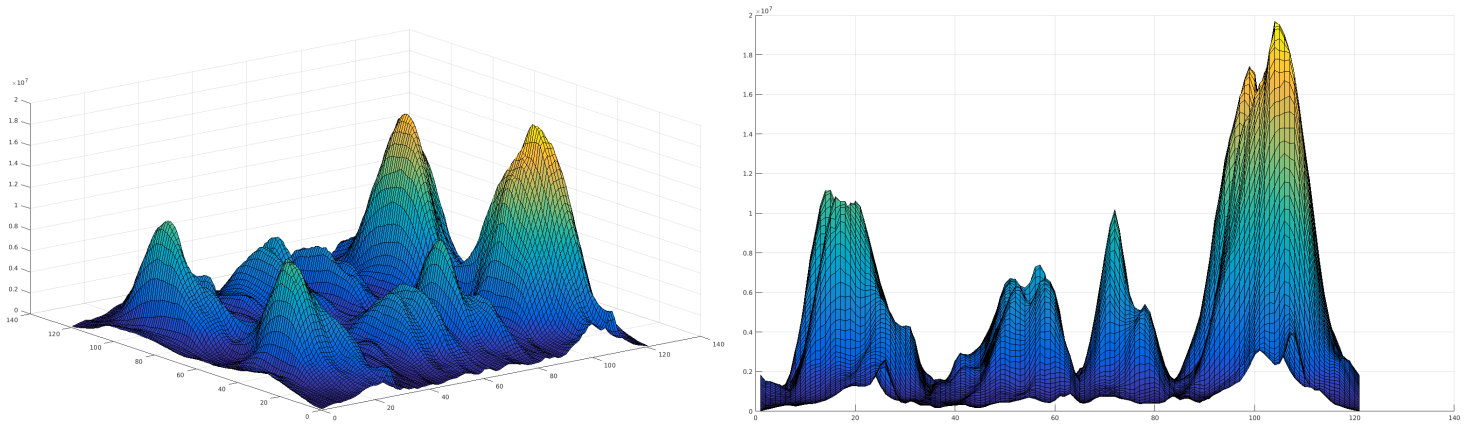
(a) (b)

Figure 7.3: Robust Median Squared Distance, Bremem Dataset. The 3D graph of how the robust median distance behaves, varying the pitch and the yaw between the two point clouds.



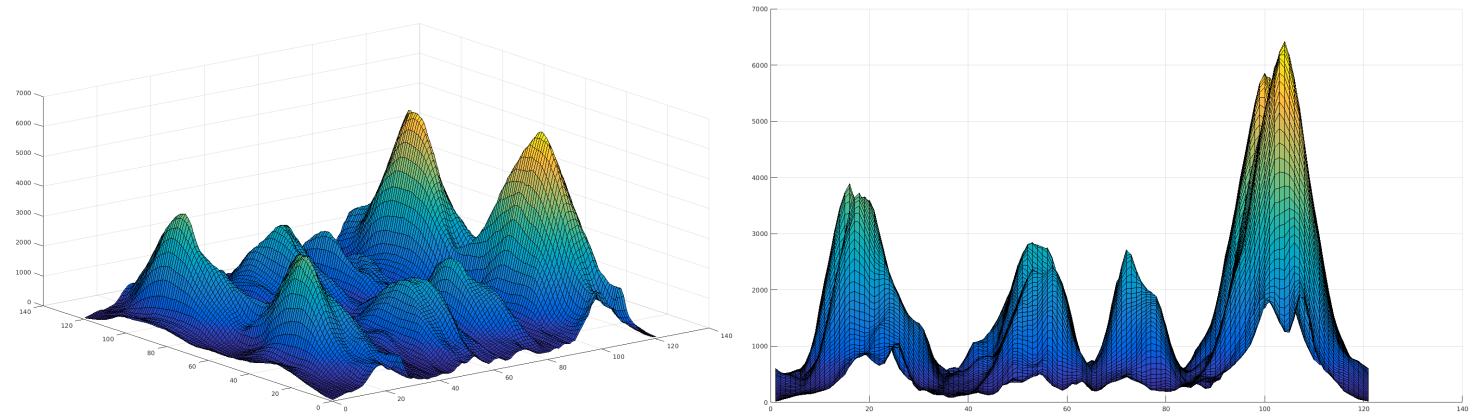
(a) (b)

Figure 7.4: Sum of Squared Distances, Bremen Dataset. The 3D graph of how the sum of squared distances behaves, varying the pitch and the yaw between the two point clouds



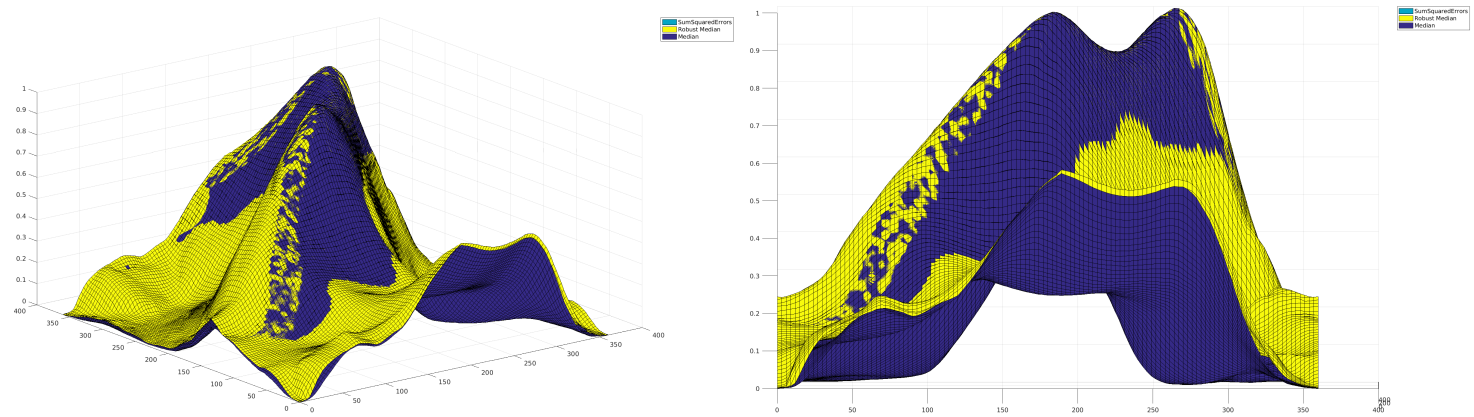
(a) (b)

Figure 7.5: Robust Sum of Squared Distances, Bremen Dataset. The 3D graph of how the "robust" sum of squared distances behaves, varying the pitch and the yaw between the two point clouds.



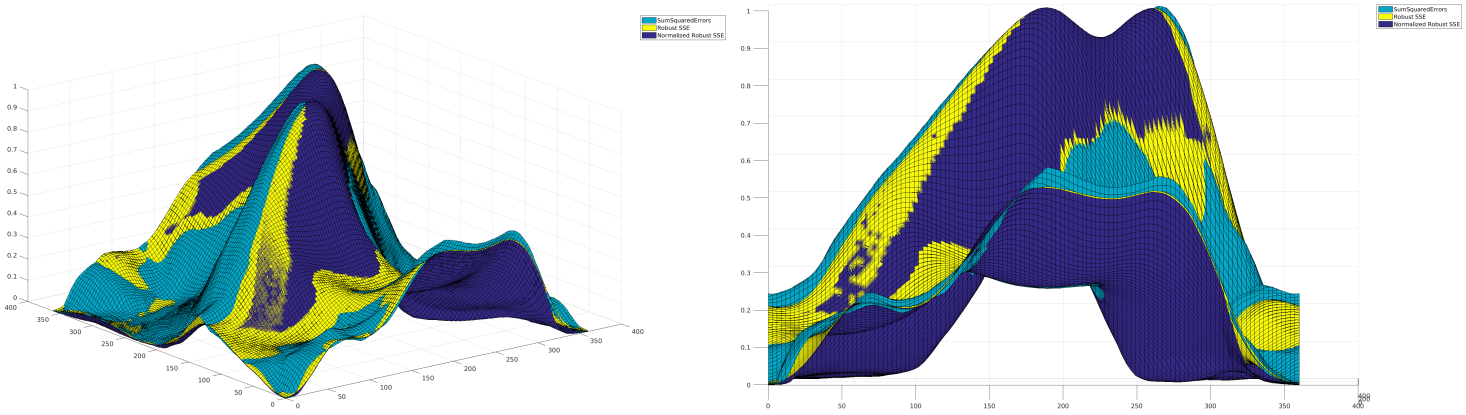
(a) (b)

Figure 7.6: Normalized Robust Sum of Squared Distances, Bremen Dataset. The 3D graph of how the normalized "robust" sum of squared distances behaves, varying the pitch and the yaw between the two point clouds, for the Bremen dataset.



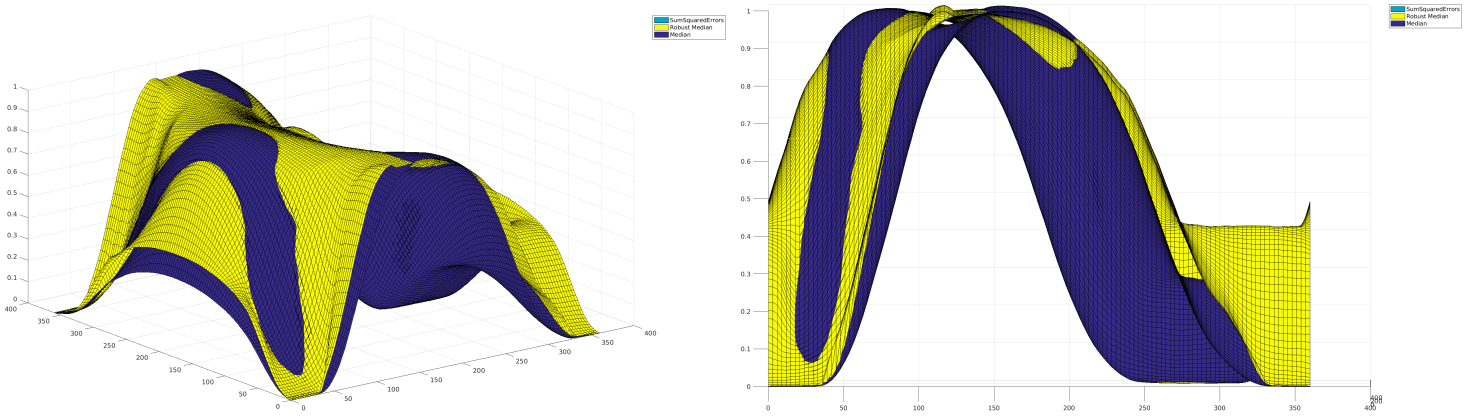
(a) (b)

Figure 7.7: Metrics for the Office Dataset.



(a) (b)

Figure 7.8: Sum of Squared Errors variants for the Office Dataset.



(a) Metrics, Dataset (b)

Figure 7.9: Metrics for the Linköping Dataset.

sense. We can see, from Figure 7.11 that there are not many differences between the median and the sum of squared errors. The only relevant one is that the median has a higher range, that is, the distance between the minimum and the maximum is higher. This is a valuable difference because it makes finding a globally optimal solution easier. The robust median, on the other hand, has a "oscillating" behavior while approaching to the global minimum. Thus it has to be avoided. Looking at Figure 7.12, we can see that the robust sum of squared errors behaves exactly like the sum of squared errors, while, on the other hand, the Normalized version behaves much more like the median.

From these plots we could say that the median and the normalized sum of squared errors seems to be more suitable to our task, but there is still no clear winner.

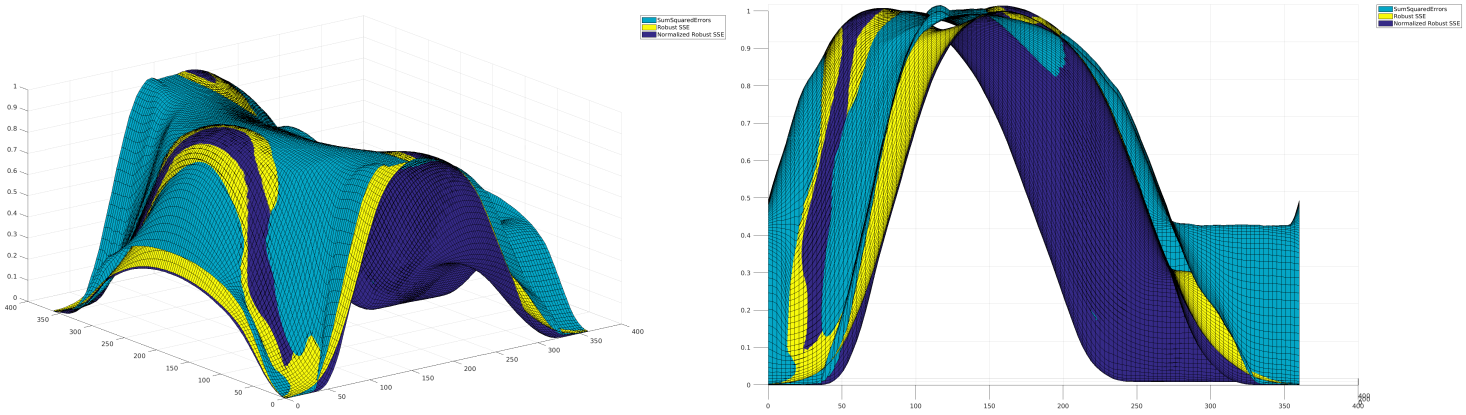
The same considerations apply to Figures 7.13 to 7.16 showing the same plots for the Linköping and Office datasets.

These plots make a good job at showing the *shape* of a function. But the position of the global minimum is not always easily discernible. This is particularly true when there are some very good local minima. Of course, besides the plots, we also have the numeric vales and, thus, with a simple search we were able to verify that the global minimum was almost always in the right position. That is, in any combination of 0° and 360° for the rotation and in 0, 0 for the translation. There were cases when the minimum was not in the best position, but it was always very close to. This, indeed, is not necessarily a problem, since the technique we want to develop here is not a fine registration algorithm, but has its purpose in obtaining a rough initial guess. Thus getting very close to the best position is a perfectly acceptable result, since it will be improved later using a fine registration technique. Moreover, it has to be noted that the ground truth of the datasets is not always 100% correct. This is especially true for the Office and Linköping Datasets, whose ground truth has been produced manually. A few degrees of error, for example, is a perfectly plausible scenario in those cases.

For the *robust* version of the functions, we have chosen to discard all the associations whose distance is above three times the median distance and below one third of it. Thus, we will say that the *robustness* factor is 3.

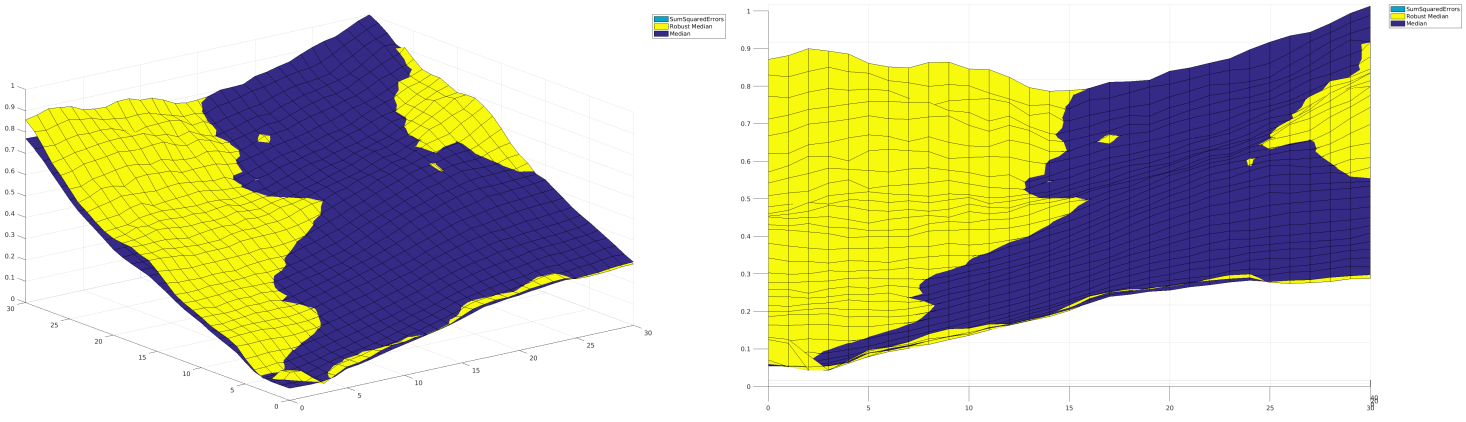
Is it the only possible value? Is it the best value?

Our experiments show that the actual value is not of uttermost importance, since they all tend to behave similarly when approaching to a minimum, that is in the area we are most interested in. As an example, in Figure 7.17 we have calculated the sum of squared errors with different robustness factors (ranging from two to six) for two point clouds from the Bremen Dataset. To perform the tests, the source point clouds has been



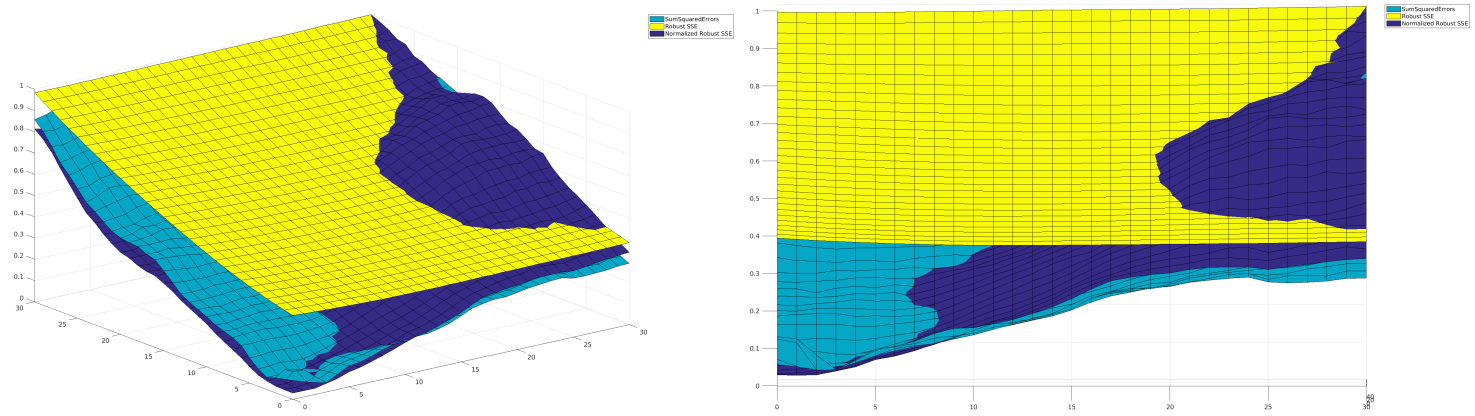
(a) Metrics, Dataset (b)

Figure 7.10: Sum of Squared Errors variants for the Linköping Dataset.



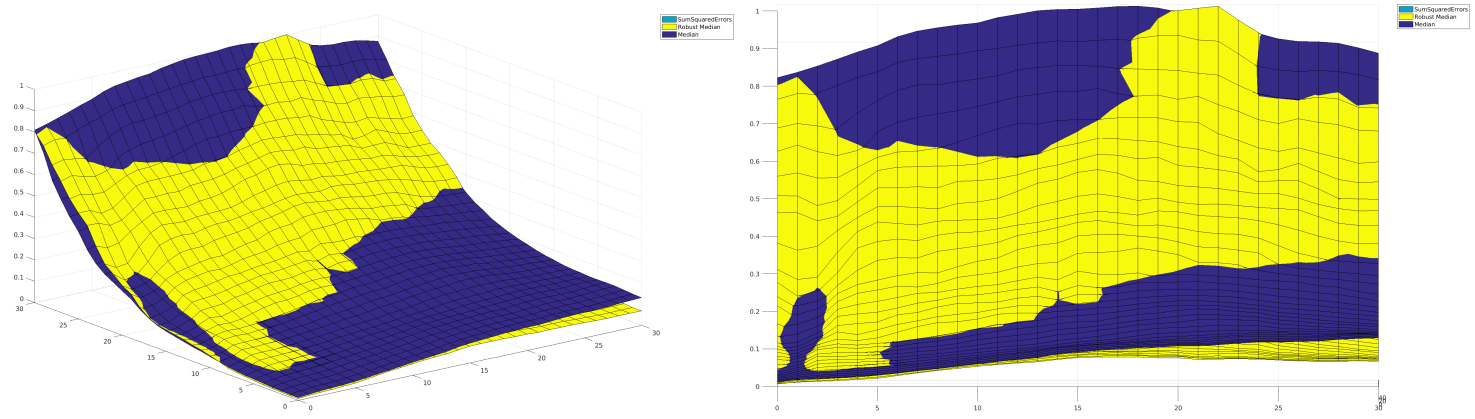
(a) (b)

Figure 7.11: Metrics for the Bremen Dataset, translation.



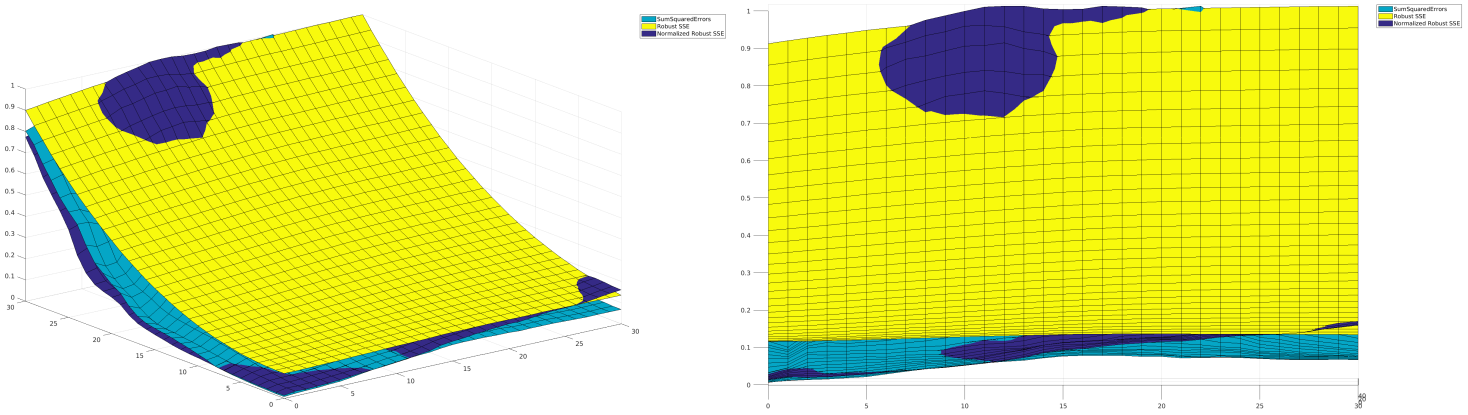
(a) (b)

Figure 7.12: Sum of Squared Errors variants for the Bremen Dataset, translation.



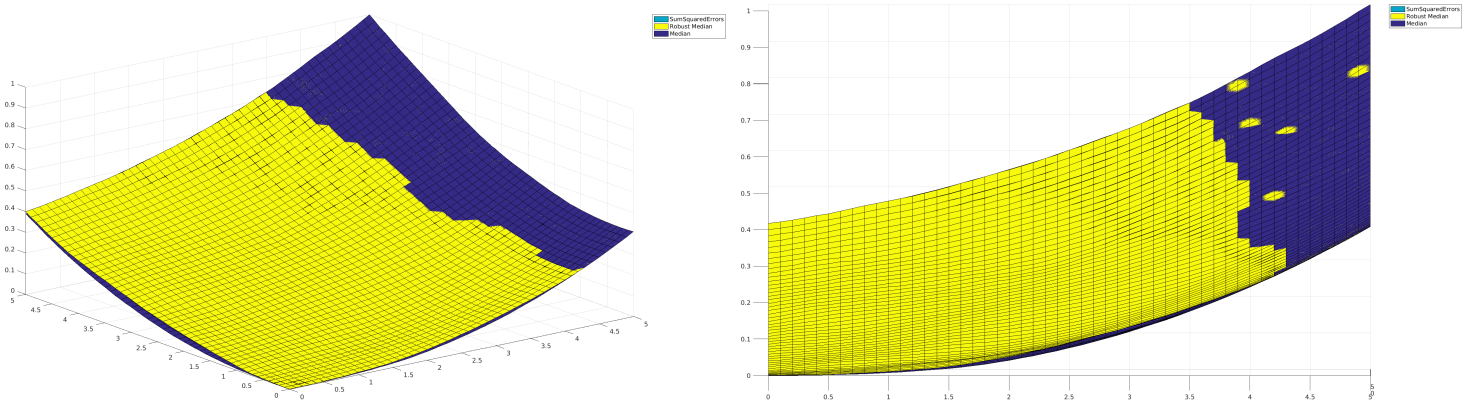
(a) (b)

Figure 7.13: Metrics for the Linköping Dataset, translation.



(a) (b)

Figure 7.14: Sum of Squared Errors variants for the Linköping Dataset, translation.



(a) (b)

Figure 7.15: Metrics for the Office Dataset, translation.

rotated around a fixed axis of an angle of increasing value (from 0° to 360°). We can see that the different robustness factors differentiate mainly at the center of the plot, corresponding to the maximum rotation. On the other hand, near the minima, they overlap.

7.3 Particle Swarm Optimization

General Description

Particle Swarm Optimization (PSO) is a bio-inspired soft computing technique, [40], used to solve optimization problems of arbitrary dimensions, without the need to compute any derivative. Even though there is no guarantee of its convergence to the global optimum (actually it could converge even to a solution that is not even a local optimum), it has been shown of having good properties of escaping from locally optimal solutions, given that a good amount of particles and the right parameters are being used.

There exist many different variants of Particle Swarm Optimization. Here we will describe only the algorithm we actually used for our experiments.

The main idea is that we have a set of particles exploring the space of the possible solutions. Thus, the "position" of the particle represents a possible solution of the optimization problem and it has a score that represents how good the solution is. Each of these particles has a memory used to memorize two kinds of best solution (with *best* meaning the one with the lowest/highest score):

- The local best, that is the best solution found by the particle up to that moment.

- The global best, that is the best solution found so far by the entire swarm. This implies that each particle communicates with each other particle. The way the particles communicate is called topology. The fully connected topology is only one of the many possible, see Figure 7.18. For example, when dealing with problems with many local minima, it is usually better to use a ring topology, [43]. With the ring topology, each particle communicates only with two neighbouring particles, Figure 7.19. In this way, when a new best is found by a particle, it is not propagated to the entire swarm in just one step, but its propagation is much delayed so that the other particles are still free to explore the space. Other kinds of topology are the *wheel* and *Von Neumann* topologies, [44], but, since we did not use them, they will not be described here.

PSO is an iterative algorithm. At each iteration each particle moves toward a combination of its local best and the global best. In addition it has also some inertia, so also the velocity at the previous iteration plays a role.

Formally, the velocity v_t at time t is given by Equation (7.3), where $best(p)$ is the local best of particle p, $gbest(p)$ is its global best, $rand([0,1])$ is a random number uniformly generated in the interval $[0,1]$ and p_{t-1} is the position of particle p at time $t-1$. α and β are two parameters of the algorithm that represents how much a particle should follow its previous velocity (the inertia, in practice) or it should deflect toward the bests.

$$v_t = \alpha \cdot v_{t-1} + \beta \cdot rand([0,1]) \cdot (best(p) - p_{t-1}) + \\ + \beta \cdot rand([0,1]) \cdot (gbest(p) - p_{t-1}) \quad (7.3)$$

The position at time t is simply given by Equation (7.4).

$$p_t = p_{t-1} + v_t \quad (7.4)$$

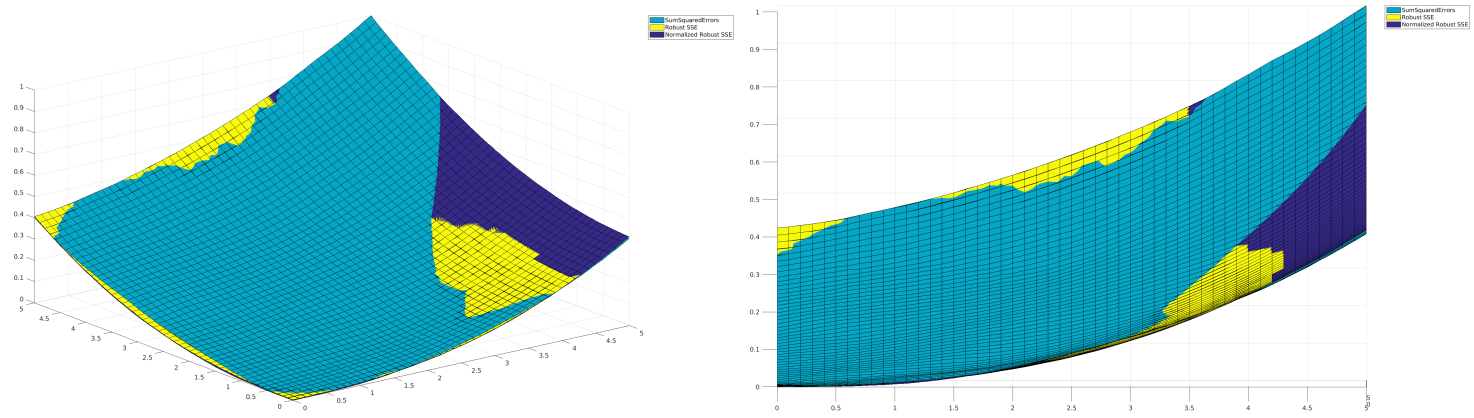
It could happen that the result of Equation (7.4) brings a particle outside of the feasible search space. To avoid this, many alternatives have been studied by the research community, [43], with results that often depend heavily on the studied problem. In this work we decided that, whenever a particle ends outside the search space, it is repositioned at the boundaries and its velocity is reversed after having being damped by a random factor. In practice, the particle bounces on the boundaries of the search space in a not completely elastic way. Also the velocity of the particle has been limited, to avoid the particles bouncing around too quickly, without exploring deeply the space. These solutions, among the many tested, gave us the best results.

The initialization of the algorithm is another important step, since the particles must have an initial velocity and position in order for the algorithm to work. The initial position has been set to a random value uniformly distributed in the search space, while the initial velocity has been simply set to zero. This simple strategy has been proven to be efficient and effective, [45].

The working of the algorithm is described by Algorithm 1.

Particle Swarm Optimization for Initial Guess Estimation

In the previous section we have described the working of a general Particle Swarm Optimization algorithm. In this section we will see how this general



(a) (b)

Figure 7.16: Sum of Squared Errors variants for the Office Dataset, translation.

Algorithm 1 Particle Swarm Optimization

```
for all p do
    random_init(p)
end for
for t = 1 to max_iteration do
    for all p do
        update v_t^p
        p_t = p_{t-1} + v_t^p
        if score(p_t) < score(best(p)) then
            best(p) = p_t
        end if
        if score(p_t) < gbest then
            gbest = p_t
        end if
    end for
end for
return gbest
```

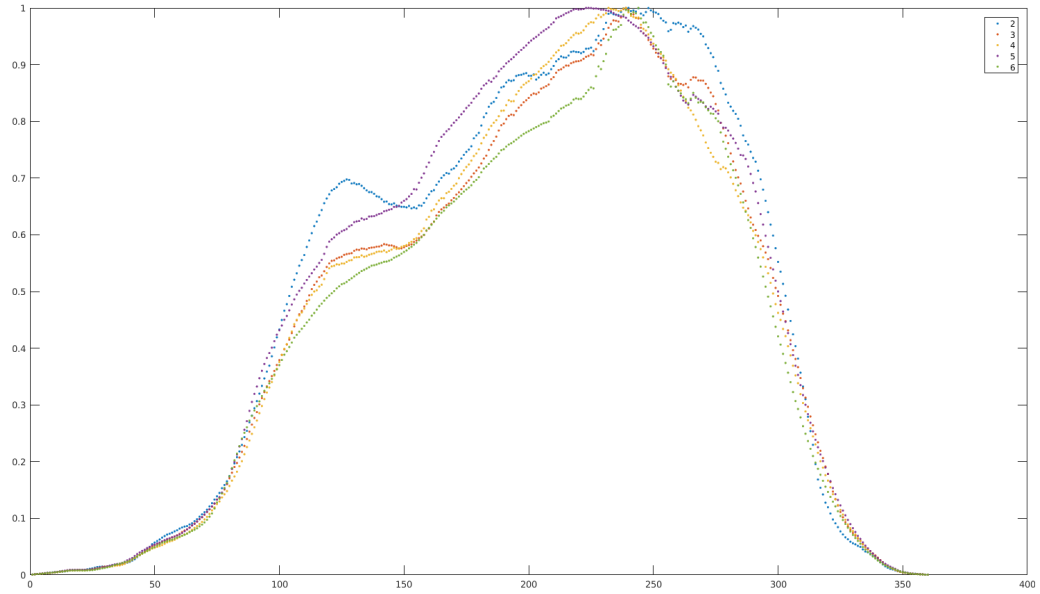


Figure 7.17: The final value of the sum of squared errors for increasing initial rotations, with different robustness factors. On the x-axis we have the angle of the rotation (around a fixed axis), on the y-axis we have the sum of squared errors. The values have been normalized in order to compare different robustness factors on the same plot.

algorithm can be applied to the estimation of an initial guess for point clouds registration.

First of all, we must define what a particle is or, more specifically, which is the particle state for our specific problem. The unknowns we want to estimate are two: the rotation and translation, each with three degrees of freedom. Thus, the state will have six dimensions, three for the rotation and three for the translation. Representing a translation is very straightforward: each value represents the translation on a particular axis (x,y,z). As explained before, the search space has to be limited. For the translation these boundaries have to be specifically set accordingly to the datasets, since the scales could be very different. However, they do not need to be set to a precise value, the range could be very big, they just need to exist. A general solution is to set the boundaries on the three axis to the maximum feasible translation. For our experiment, we set the upper bound of the translation search space to maximum coordinates, on each axis, of the target cloud. We applied the same procedure also to the lower bound, using the minimum. In this way the algorithm does not need any manually specified parameter.

For the rotation the problem is a little trickier, since many different rep-

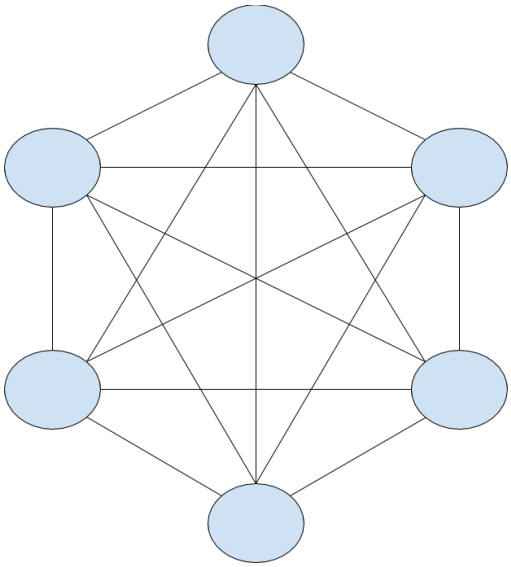


Figure 7.18: Swarm with fully connected topology.

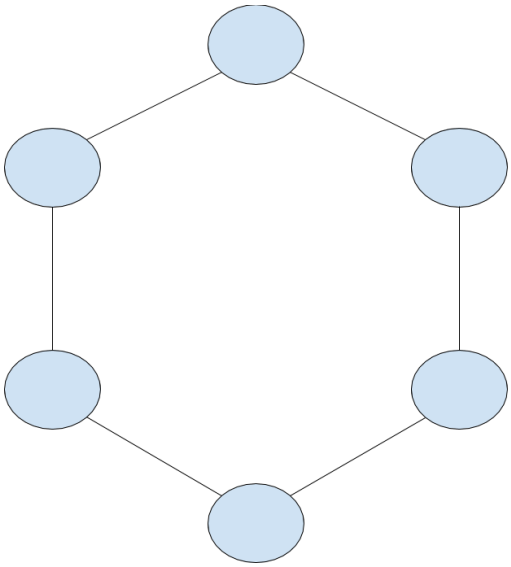


Figure 7.19: Swarm with ring topology.

resentations exist. The most intuitive representation for a rotation in a 3D space is the so-called Euler Angles representation. It is very intuitive because each angle represents a rotation around a particular axis, but, for our application, they suffer of a very important drawback: the Gimbal lock.

For this reason, instead of Euler angles, we use the axis-angle representation, which represents an arbitrary 3D rotation with a rotation of an angle θ around a given unit vector $\boldsymbol{e}$. A vector in the space has three components, but for our representation we consider only unit vectors, i.e. normalized vectors. Since the norm of the vector has to be 1, we have only two degrees of freedom, plus one from the angle, that is a total of three, as expected since we want to represent a 3D rotation. The problem is: how we define the range of every possible rotation? Using Euler angles the solution was straightforward, since they are just rotation angles around an axis. For the axis-angle representation instead, this translates to defining the range representing every possible unit vector and every possible rotation angle. The former is non-trivial using Cartesian coordinates. For this reason we decided to switch to spherical coordinates for the definition of the unit vector. On a sphere of unit radius (since we are considering only unit vectors), a vector is identified by two angles: inclination θ and azimuth φ. Going from spherical coordinates to Cartesian coordinates is very easy, see Equation (7.5).

$$\left\{ \begin{array}{c} x = r \cdot sin\theta \cdot cos\varphi \\ y = r \cdot sin\theta \cdot sin\varphi \\ z = r \cdot cos\theta \end{array} \right\} \tag{7.5}$$

So, with a unit vector in spherical coordinates and an angle, to represent our arbitrary 3D rotation we need only three angles. With this system we have the simplicity of the Euler angles but without the risk of Gimbal locks. Like for the translation, we have to define the boundaries of our search space. In the most general case, we do not have any prior information on it, so we could search all the possible rotations. This is easily expressed allowing the three angles to move between 0° and 360°. This would not be so easy using Cartesian coordinates and is the reason because we switched to spherical coordinates.

As an alternative, we could have used quaternions to represent a rotation. The quaternion representation is basically equivalent to the axis-angle representation and, thus, it does not suffer of the Gimbal lock problem. However, quaternions pose a very important problem for our application. We want a range of rotations that contains every possible rotation in the space. How do we express this range in terms of values of the components of the quaternion? Even in the case we do not want to consider every possible rotation, but a

subset (say, for example that we have a guess coming from another sensor), the problem is still there, only with a smaller range. With the axis-angle representation we solved this issue switching to spherical coordinates.

To summarize, the state of our particles has six dimensions, three for the rotation and three for the translation. To evaluate a particle, *i.e.*, to calculate its score, we use one of the function we described in Section 7.2. We will call such a function, generically, $M(X, Y)$. Given two point clouds X and Y and being R_p and T_p the rotation and translation represented by the particle p, the generic scoring step will use Equation (7.6).

$$score(p) = M(R_p \cdot X + T_p, Y) \tag{7.6}$$

In practice the score is a measure of how well aligned are the two point clouds, once the source has been transformed with the particle's estimate of the transformation.

These are all the customizations needed by the Particle Swarm Optimization algorithm in order to be used for the estimation of an initial guess for point clouds registration.

7.4 Experimental Results

We tested our approach with various datasets and using the different metrics described earlier. In Table 7.1 we present the test of the various metrics using two point clouds coming from the Bremen Dataset. The algorithm has been run each time using exactly the same parameters, that is, 50 particles and 1000 iterations of Particle Swarm Optimization. We have chosen a large number of iterations because we wanted to test the limits of the algorithm without caring, for the moment, for the execution times. Nevertheless, we discovered that, when the algorithm is eventually going to converge to the right solution, it does in a few iterations. Moreover, we used a high number of particles and iterations to minimize the impact of the intrinsic randomness of the algorithm on the results. In this way a good (or poor) result is less likely to having been influenced by an unfortunate initialization (which is random). In Table 7.1 MSE stands for Mean Squared Error and represents the distance w.r.t. the ground truth. It has been calculated as described in Section 4.2. Of course, the lower this value is, the better the result.

As can be seen, the metric we called Robust Normalized SSE provided the best results. In fact, as can be seen from Figure 7.20, the source point cloud (in blue) and its ground truth (in red), overlap almost perfectly. This is way more than what is usually required as an initial guess. Although the

Table 7.1: The results of the experiments on the Bremen dataset, using different metrics

Metric	Metric Value	MSE
SSE	$1.444E+11$	115.918
Robust SSE	77284	124.293
Robust Normalized SSE	12.5224	2.50207
Median	2.58451	125,479
Robust Median	0.00623869	5,08459

MSE for the Robust Median is more or less twice of that of the Robust Normalized SSE, in practice the two results are equivalent, since this algorithm is not aimed at fine registration. Instead, it is enough to bring the source cloud more or less around the proper alignment, since it will be refined later. Since the picture of the alignment using the Robust Median metric is practically identical to that of the Robust Normalized SSE, it is not shown. Very important, for the Robust Normalized SSE, the algorithm converged around the right solution very early, more or less after only 20 iterations.

The other metrics, instead, provided very bad results. In fact, no one was able to provide a meaningful solution, the alignments were all very wrong. This happened because, for each of these metrics, the value of the chosen solution was very close to the value of the right solution, albeit being very distant in the search space. This characteristic makes the job of the optimization algorithm very hard, as already discussed and makes the metric not satisfy the requirements.

Since the Robust Normalized SSE proved to be the best metric, we tested it also on all the other point clouds from the Bremen Dataset. The results are shown in Figure 7.21. Most of the mean squared errors are very low, with the exception of those corresponding to registration number 6 and 7. These experiments have been performed trying to align a point clouds with the previous one, *i.e.*, scan_001 with scan_000, scan_002 with scan_001 and so on. The poor results obtained in these cases is probably due to the small overlap between some of the point clouds. A better strategy is to align a point cloud with another, first roughly and then refining with some point clouds registration algorithm, fuse the aligned point clouds and then align the next one with the complete cloud obtained in the preceding step.

The results of the experiments on the Office Dataset are shown in Table 7.2. In this case all the proposed metrics got very good results and were able to properly align the two point clouds. The two point clouds, aligned using the Robust Normalized SSE are shown in Figure 7.22. Only the SSE

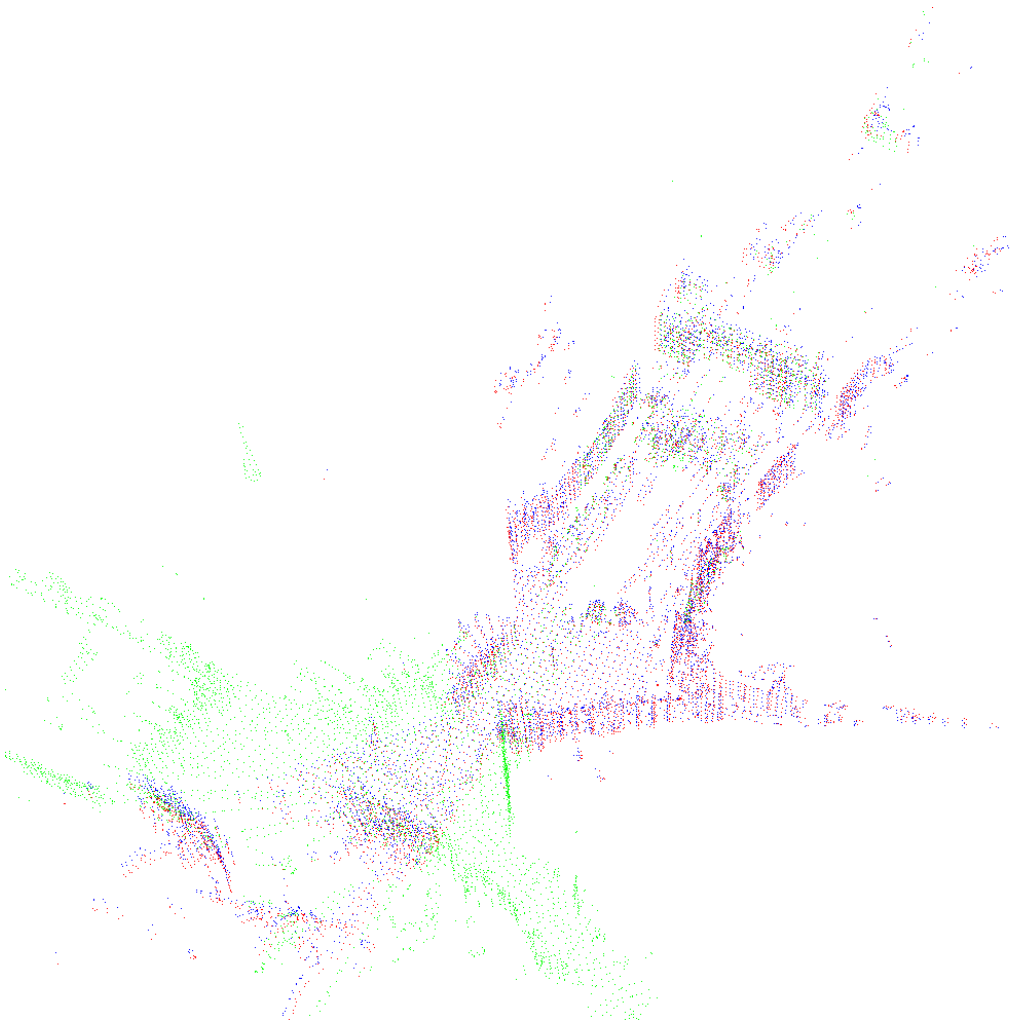


Figure 7.20: Two clouds from the Bremen Dataset, aligned using the Robust Normalized Sum of Squared Error metric. In green the target point cloud, in blue the source cloud, in red the ground truth of the source cloud. As can be seen, the source cloud and its ground truth overlap almost perfectly.

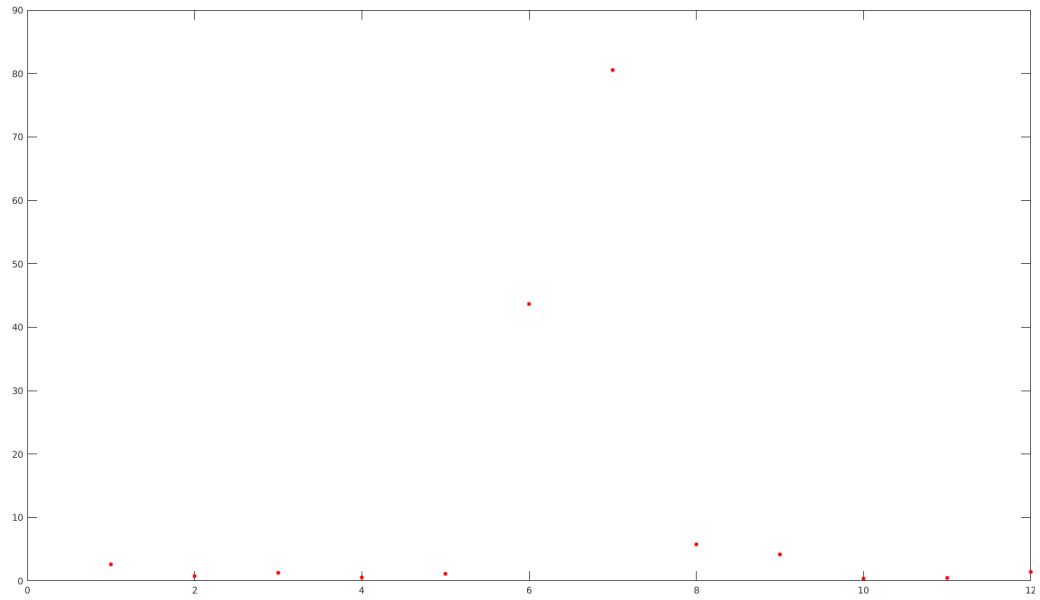


Figure 7.21: The Mean Squared Errors of the registrations of point clouds from the Bremen Dataset. On the x-axis there are the different registrations, on the y-axis the value of the metric.

metric got worse results, but still very good as initial guess, Figure 7.23.

The results for the Linköping Dataset are shown in Table 7.3. During these experiments the point clouds have been heavily subsampled for speed reasons. This should not affect the quality of the result too much, since we are only seeking for a rough alignment during this step. The two clouds aligned using the Normalized Robust SSE metric are shown in Figure 7.24. With this dataset, the only metric not obtaining good results is, again, the bare Sum of Squared Errors. In Figure 7.25 the two point clouds, aligned with this metric, are shown. The result is completely wrong and not useful as rough initial guess. On the other hand, the other metrics obtained very good results.

Table 7.2: The results of the experiments on the Office dataset, using different metrics

Metric	Metric Value	MSE
SSE	199.028	0.222662
Robust SSE	9.4217	0.0408666
Robust Normalized SSE	0.012149	0.0661749
Median	0.0108989	0.0408433
Robust Median	8.00E-06	0.0669206

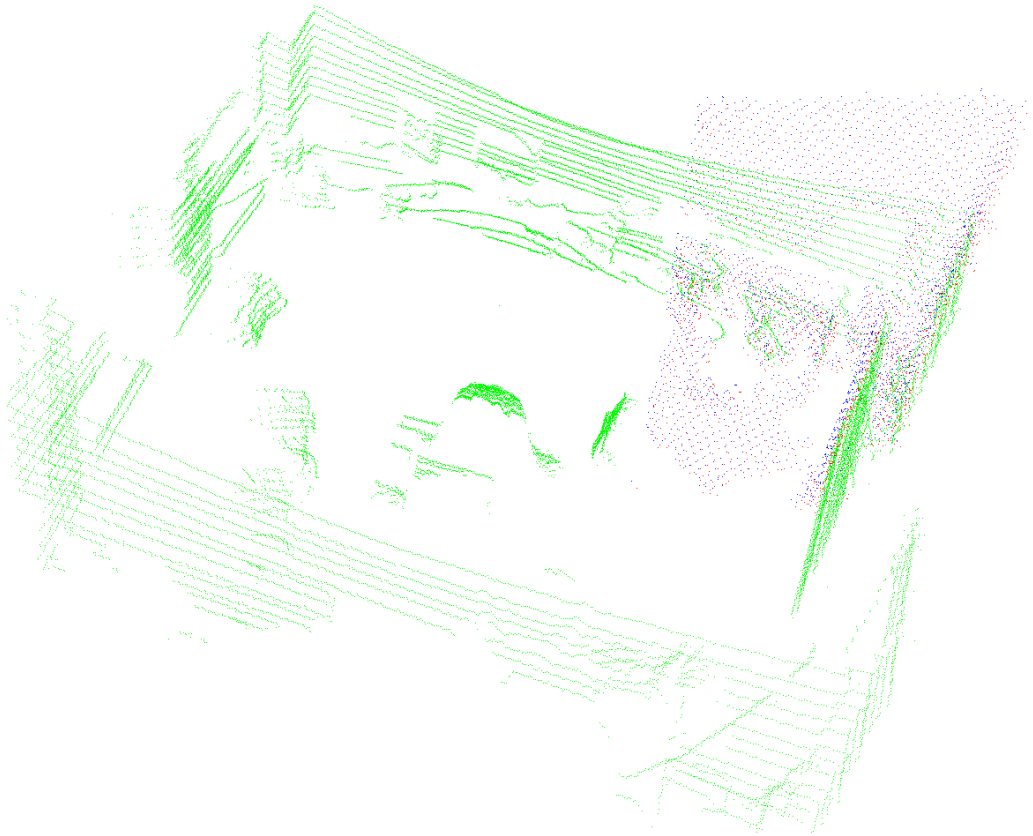


Figure 7.22: Two clouds from the Office Dataset, aligned using the Robust Normalized Sum of Squared Error metric. The colors have the same meaning as in Figure 7.20. The source cloud and its ground truth overlap almost perfectly.

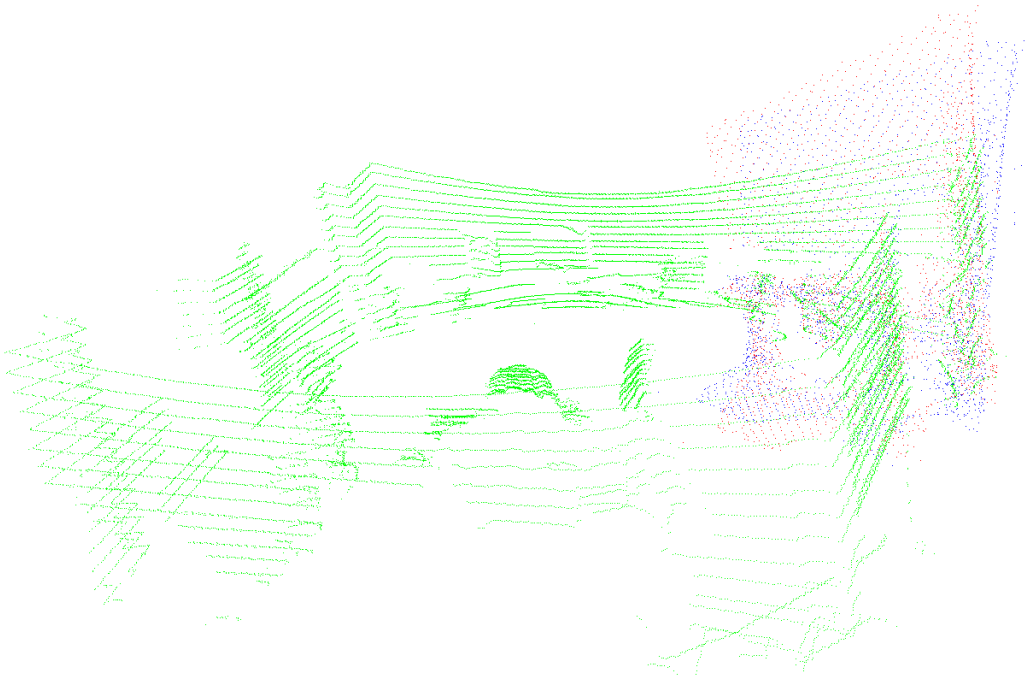


Figure 7.23: Two clouds from the Office Dataset, aligned using the Sum of Squared Error metric. The colors have the same meaning as in Figure 7.20. The result is worse than with other metrics, but still quite good as initial guess.

Table 7.3: The results of the experiments on the Linköping Dataset, using different metrics

Metric	Metric Value	MSE
SSE	173398	38.9075
Robust SSE	3.1554	2.84418
Robust Normalized SSE	2.00363	2.84418
Median	3.00307	2.18768
Robust Median	3.09373	0.00155527

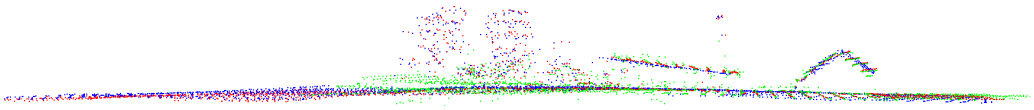


Figure 7.24: Two clouds from the Linköping Dataset, aligned using the Robust Normalized Sum of Squared Error metric. The colors have the same meaning as in Figure 7.20.

Although with the Robust Median we got an error several order of magnitude lower than with the other metrics, this difference is almost negligible, given the resolution and the precision of the point clouds, and considering that we are testing a technique aimed at preliminary rough alignments.

The results for the Hannover datasets are shown in Table 7.4. The experiments have been performed aligning two non consecutive point clouds, to make the problem a little bit harder, reducing the overlap between the clouds. Otherwise we could not show any substantial difference between the various metrics. Nevertheless all the proposed metrics provided very good results. Consider that the point clouds, before the registration, had been sub-sampled with a voxel grid with leaves of size 1. Thus, an error w.r.t. the ground truth of less than one is very good. In conclusion, the differences between the metrics are, in this case, negligible.

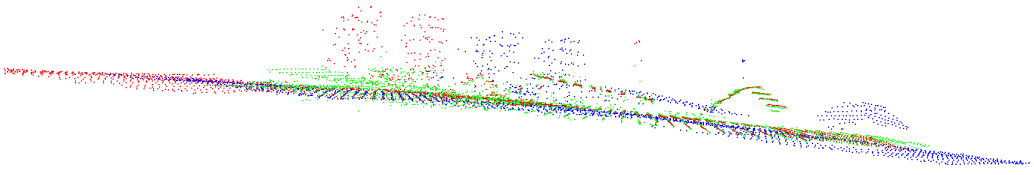


Figure 7.25: Two clouds from the Linköping Dataset, aligned using the Sum of Squared Error metric. The colors have the same meaning as in Figure 7.20. Notice how the blu point cloud is not aligned with the red one, which is the ground truth.

Table 7.4: The results of the experiments on the Hannover Dataset, using different metrics

Metric	Metric Value	MSE
SSE	5088.82	1.08132
Robust SSE	521.855	0.443654
Robust Normalized SSE	0.23484	0.455157
Median	0.382769	0.186318
Robust Median	0.000354692	0.374115

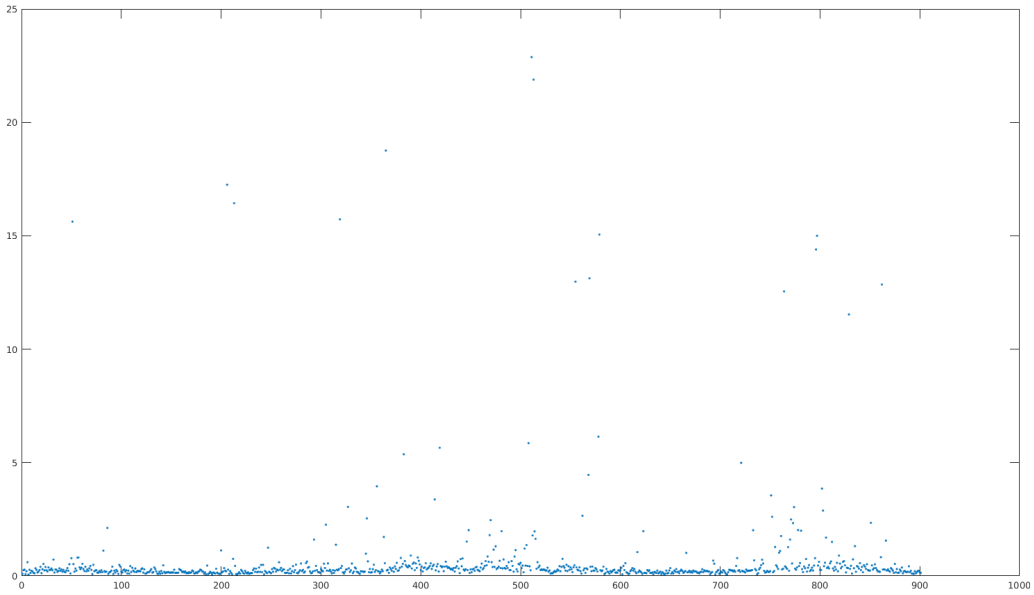


Figure 7.26: The MSE w.r.t. the ground truth of the experiments on the Hannover Datasets.

The Hannover Dataset is composed of 902 point clouds. We tried to align each of these clouds with an adjacent one and calculated the Mean Squared Error (MSE) w.r.t. the ground truth. The results are plotted in Figure 7.26. The average MSE is 0.6653 and, as can be noted, most errors are very small, with just a few outliers. Notice that there is a random component in the PSO algorithm, so some randomly wrong results among so many registrations are to be expected. Nevertheless, the average error is very very small.

3D Reconstruction and Localization

In order to prove that the combination of our approaches for global registration and refinement could be used on a robotic platform for both mapping

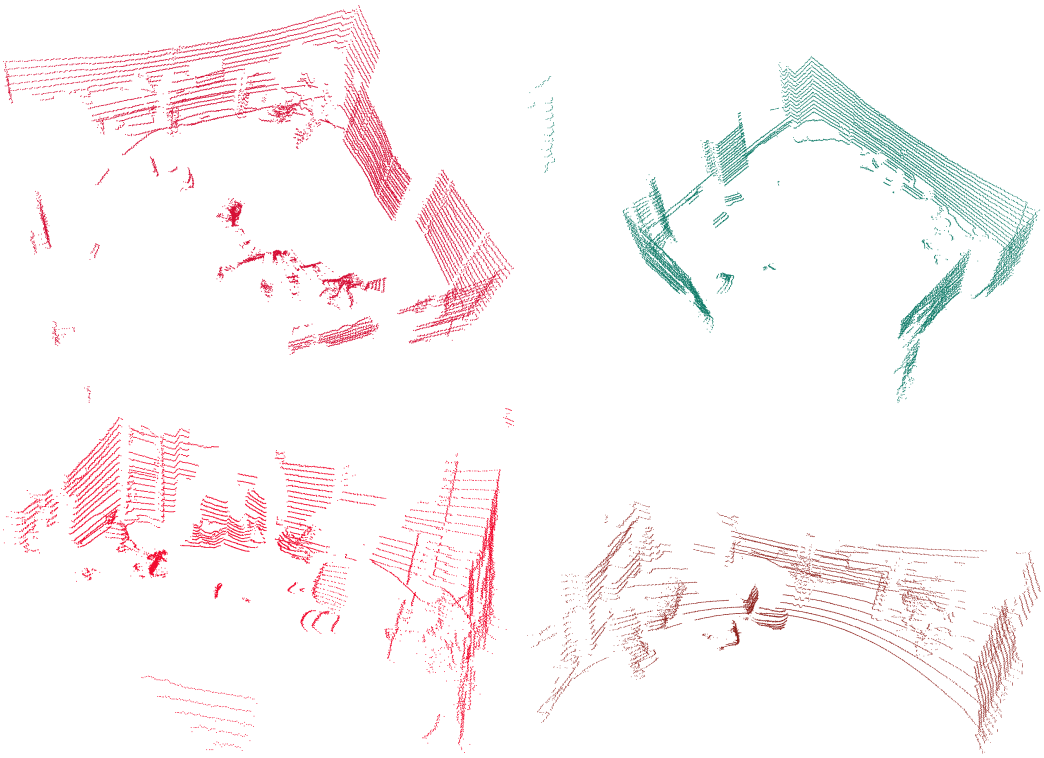


Figure 7.27: Some of the point clouds used to reconstruct the office.

and localization, we performed another experiment.

As first step, we collected a set of six point clouds in an office, Figure 7.27, using an hand-held Velodyne VLP-16, and we aligned them. The registration was performed incrementally: the second one was registered with the first one and we fused them. Then, the third one was registered with the result of the previous registration and so on.

During this alignment step we did not provide any initial guess to the registration algorithm. This process was performed using only our PSO-based algorithm, no further refinement via ICP or other variants was necessary, because the result was already good enough, Figure 7.28.

For this dataset no ground truth was available. The only feasible way to have a ground truth would have been to produce it manually. However, we were not able to produce one accurate enough. Indeed, with the algorithm we got results that we were not able to improve manually, thus using such a ground truth would have been meaningless. For this reason, only a qualitative evaluation is possible for the evaluation of the results on this dataset.

Once we performed the 3D reconstruction, we basically got a 3D map of the office. This map can be used for localization, with either point clouds

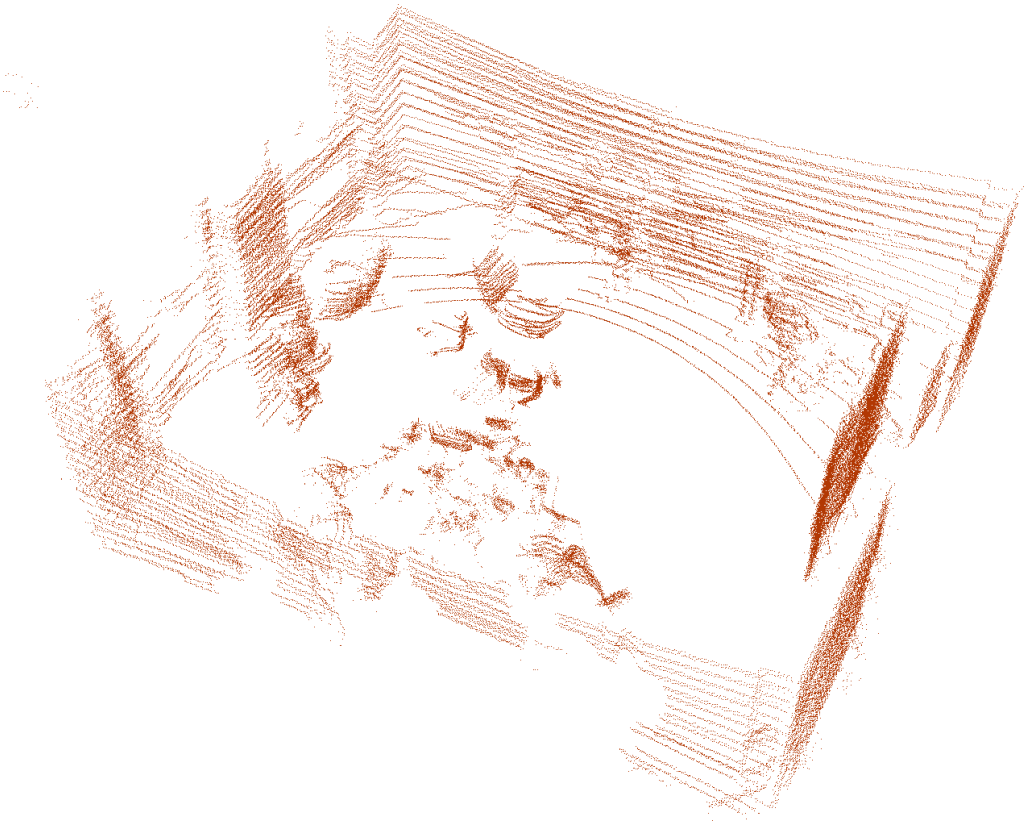


Figure 7.28: The full reconstruction of the office, performed using only our PSO-based algorithm.

Figure 7.29: A point cloud produced with a Kinect and its registration in the map produced with a LiDAR.

from the same kind of sensor used to produce it, or with a different one.

Since one of the topics covered in this work is dense-sparse registration, we decided to use for this localization experiment a sensor producing very dense point clouds. Therefore, as second step, we tried to localize a hand-held Kinect, by aligning its point clouds with the full 3D map of the office. A 3D localization system, such as a VICON motion capture system, could have been used to produce a ground truth for this set of experiments. However, none of this kind of systems was available to us. Therefore, also for the localization, only a qualitative evaluation in possible.

Between the collection of the data in the two steps (3D reconstruction and localization), the position of some of the furnitures were changed (notably the position of the chairs), and some objects were added (a bike in front of the closet). This setup was meant to simulate a real environment, where usually the map does not correspond perfectly to the working environment, because there could be many movable objects. The results are shown in Figures 7.29 and 7.30. In this case too, no initial guess was given to the registration algorithm. The registration was performed with the PSO-based algorithm and then refined using our probabilistic point clouds registration algorithm. Although no reliable ground truth was available, we can see that our registration pipeline was able to localize the Kinect into the room, without the necessity of any other data.

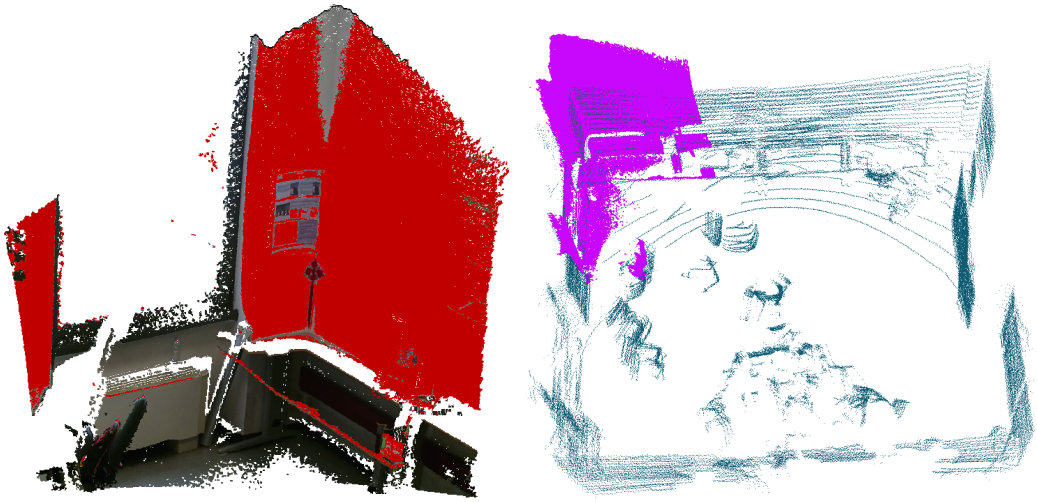


Figure 7.30: A second point cloud produced with a Kinect and its registration in the map produced with a LiDAR.

Chapter 8

Conclusions

We presented two novel algorithms for point clouds registration. The first one, we called *Probabilistic Point Clouds Registration*, is a variant of ICP with a different data association policy. It was derived applying statistical inference to a probabilistic model. Starting from a model based on a t-distribution, we derive, using Expectation-Maximization, an iteratively reweighted least squares problem. The error terms of that problem are very similar to those of ICP, with the difference that they are weighted in a way that reflects their accordance with the current estimate of the transformation. Practically, we give a high weight to the associations that are in accordance with the current estimate of the rototranslation and a lower to the others. This technique makes possible to implicitly deal with outliers, without the need of any explicit filtering step. Moreover, instead of using a single closest point association, we associate to each point in the source point cloud, a set of closest points in the target point cloud. Each of this associations is then weighted and their weights normalized.

Furthermore, we developed also a multi-iteration version of the Probabilistic Point Clouds Registration algorithm. Each iteration is composed of an iteratively reweighted least squares problem. In this way, an initial estimate of the transformation between the two point clouds is iteratively improved, until some convergence criterion is met, in a way analogous to ICP.

We took particular attention to the design of a termination criterion that could be used for any application, without the need to fine tune any parameter, in opposition to what happens for many other point clouds registration algorithms. For this reason, we conducted a series of experiments that resulted in a very general termination criterion.

We think that parameters' sensitivity is a very important issue for state-of-the-art point clouds registration algorithms. Therefore, we tested the sen-

sitivity of popular point clouds registration algorithms, namely ICP, G-ICP, along with our proposal. We show that the probabilistic point clouds registration algorithm shows a very low level of parameters' sensitivity. This is a very desirable behaviour, especially in robotics applications, because it means that it can be used in many different situations without re-tuning the parameters.

However, the Probabilistic Point Clouds Registration algorithm, much like other fine-registration techniques in the literature, still requires an initial guess in order to converge to a proper solution. Instead, we wanted a solution that could align two point clouds, regardless of their initial alignment.

For this reason we developed a global point clouds registration algorithm, aimed at finding a rough alignment, that could then be refined using other techniques. Our proposal is based on a soft-computing algorithm: Particle Swarm Optimization. Particular attention was given to the design of the function to optimize. Since the associations between points are not known a-priori, the *perfect* error function does not exists. Instead, we tested several approximations of this *ideal* function to discover the best one. Specifically, that function must satisfy two requisites:

1. The global minimum of the function must correspond to the best alignment.

2. The function must not have too many good local minima, otherwise the optimization algorithm will struggle to find the global minimum.

We found a function that satisfy our requirements and tested it on many different kinds of dataset. We show that the proposed algorithm is very efficient at finding a global alignment that, often, does not even require to be refined.

Bibliography

[1] Z. Zhang, "Microsoft kinect sensor and its effect," *IEEE multimedia*, vol. 19, no. 2, pp. 4–10, 2012.

[2] J. Han, L. Shao, D. Xu, and J. Shotton, "Enhanced computer vision with microsoft kinect sensor: A review," *IEEE transactions on cybernetics*, vol. 43, no. 5, pp. 1318–1334, 2013.

[3] D. Forsyth and J. Ponce, *Computer vision: a modern approach*. Upper Saddle River, NJ; London: Prentice Hall, 2011.

[4] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 3384–3391, IEEE, 2008.

[5] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 3212–3217, IEEE, 2009.

[6] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, "Fast 3d recognition and pose using the viewpoint feature histogram," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 2155–2162, IEEE, 2010.

[7] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard, "Narf: 3d range image features for object recognition," in *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, vol. 44, 2010.

[8] J. Jiang, J. Cheng, and X. Chen, "Registration for 3-D point cloud using angular-invariant feature," *Neurocomputing*, vol. 72, pp. 3839–3844, Oct. 2009.

[9] G. K. Tam, Z.-Q. Cheng, Y.-K. Lai, F. C. Langbein, Y. Liu, D. Marshall, R. R. Martin, X.-F. Sun, and P. L. Rosin, "Registration of 3d point clouds and meshes: a survey from rigid to nonrigid," *IEEE transactions on visualization and computer graphics*, vol. 19, no. 7, pp. 1199–1217, 2013.

[10] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," *International journal of computer vision*, vol. 13, no. 2, pp. 119–152, 1994.

[11] A. Segal, D. Haehnel, and S. Thrun, *Generalized-ICP.*, vol. 2. 2009.

[12] P. Biber and W. Straßer, "The normal distributions transform: A new approach to laser scan matching," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 3, pp. 2743–2748, IEEE, 2003.

[13] G. Agamennoni, S. Fontana, R. Y. Siegwart, and D. G. Sorrenti, "Point clouds registration with probabilistic data association," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pp. 4092–4098, IEEE, 2016.

[14] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[15] I. Sipiran and B. Bustos, "Harris 3d: a robust extension of the harris operator for interest point detection on 3d meshes," *The Visual Computer*, vol. 27, no. 11, pp. 963–976, 2011.

[16] C. Harris and M. Stephens, "A combined corner and edge detector.," in *Alvey vision conference*, vol. 15, pp. 10–5244, Manchester, UK, 1988.

[17] Y. Zhong, "Intrinsic shape signatures: A shape descriptor for 3d object recognition," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pp. 689–696, IEEE, 2009.

[18] F. Tombari, S. Salti, and L. Di Stefano, "Performance evaluation of 3d keypoint detectors," *International Journal of Computer Vision*, vol. 102, no. 1-3, pp. 198–220, 2013.

[19] R. Rusu, N. Blodow, Z. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008*, pp. 3384–3391, Sept. 2008.

[20] R. Rusu, N. Blodow, and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3d registration," in *IEEE International Conference on Robotics and Automation, 2009. ICRA '09*, pp. 3212–3217, May 2009.

[21] A. Sehgal, D. Cernea, and M. Makaveeva, "Real-Time Scale Invariant 3d Range Point Cloud Registration," in *Image Analysis and Recognition* (A. Campilho and M. Kamel, eds.), no. 6111 in Lecture Notes in Computer Science, pp. 220–229, Springer Berlin Heidelberg, 2010.

[22] L. A. Alexandre, "3d descriptors for object and category recognition: a comparative evaluation," in *Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal*, vol. 1, p. 7, 2012.

[23] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[24] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 14, pp. 239–256, Feb 1992.

[25] Y. Chen and G. Medioni, "Object modeling by registration of multiple range images," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pp. 2724–2729, IEEE, 1991.

[26] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[27] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[28] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP variants on real-world data sets," *Autonomous Robots*, vol. 34, pp. 133–148, Apr. 2013.

[29] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *International Conference on Robotics and Automation*, (Shanghai, China), 2011 2011.

[30] A. Segal, D. Haehnel, and S. Thrun, "Generalized-ICP.," in *Robotics: Science and Systems*, vol. 2, 2009.

[31] M. Magnusson, *The Three-Dimensional Normal-Distributions Transform: an Efficient Representation for Registration, Surface Analysis, and Loop Detection*. PhD thesis, Örebro University, 2009.

[32] D. Borrmann and A. Nüchter, "3d scans of the city center of bremen, germany." `http://kos.informatik.uni-osnabrueck.de/3Dscans/`.

[33] O. Wulf, "3d scans of the university campus of hannover." `http://kos.informatik.uni-osnabrueck.de/3Dscans/`.

[34] G. Turk, "The stanford bunny." `http://www.cc.gatech.edu/~turk/bunny/bunny.html`.

[35] S. Kotz and S. Nadarajah, *Multivariate t-distributions and their applications*. Cambridge University Press, 2004.

[36] W. Feller, *An introduction to probability theory and its applications: volume I*, vol. 3. John Wiley & Sons New York, 1968.

[37] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

[38] J. L. W. V. Jensen, "Sur les fonctions convexes et les inégalités entre les valeurs moyennes," *Acta mathematica*, vol. 30, no. 1, pp. 175–193, 1906.

[39] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis*, pp. 105–116, Springer, 1978.

[40] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, pp. 1942–1948, IEEE, 1995.

[41] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information processing letters*, vol. 85, no. 6, pp. 317–325, 2003.

[42] D. Holz, A. E. Ichim, F. Tombari, R. B. Rusu, and S. Behnke, "Registration with the point cloud library: A modular framework for aligning in 3-d," *IEEE Robotics & Automation Magazine*, vol. 22, no. 4, pp. 110–124, 2015.

[43] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," in *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pp. 120–127, IEEE, 2007.

[44] Y. Del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez, and R. G. Harley, "Particle swarm optimization: basic concepts, variants and applications in power systems," *IEEE Transactions on evolutionary computation*, vol. 12, no. 2, pp. 171–195, 2008.

[45] A. Engelbrecht, "Particle swarm optimization: Velocity initialization," in *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pp. 1–8, IEEE, 2012.