Exploring LiDAR Odometries through Classical, Deep and Inertial perspectives

Pierre Dellenbach

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à MINES Paris-PSL

Exploring LiDAR Odometries through Classical, Deep and Inertial perspectives

Exploration des odométries LiDAR à travers les perspectives classiques, inertielles et d'apprentissage profond

Soutenue par

Pierre DELLENBACH

Le 10 novembre 2023

École doctorale n°621

Ingénierie des Systèmes, Matériaux, Mécanique, Energétique

Spécialité

Informatique temps réel, robotique et automatique

Composition du jury :

David Filliat Professeur, ENSTA Paris	*Président*
Andreas Nüchter Professor, University of Würzburg	*Rapporteur*
Giorgio Grisetti Professor, Sapienza University of Rome	*Rapporteur*
Véronique CHERFAOUI Professeur des universités, Université Technologie de Compiègne	*Examinatrice*
François Goulette Professeur, ENSTA Paris	*Directeur de Thèse*
Jean-Emmanuel Deschaud Chargé de recherche, MINES Paris-PSL	*Examinateur*

MINES PARIS | PSL

Acknowledgements

I wish to thank my supervisors who made this thesis possible: Jean-Emmanuel Deschaud, François Goulette and Bastien Jacquet. I never intended to do a PhD in the first place, but the fact that I had worked with Jean-Emmanuel and François during an internship made me gladly accept this unexpected opportunity proposed by Bastien.

I am very grateful to Kitware, where I met incredible people, and had tons of fun working and sharing this experience with them: Julia Sanchez, Nick Laurenson, Arnaud Billon, Jerome Dias, Lea Vauchier for the computer vision dream team, and Raphaël Cazorla, the big boss. I also want to thank my colleagues at MINES Paris-PSL, who made this PhD a very pleasant experience: Jean-Pierre Richa, Jules Sanchez, Hugo Blanc and Louis Soum-Fontez.

My parents played a big role in this PhD, my mother for her constant support, and my father checking that I was not making the biggest mistake of my life with this PhD nonsense.

Finally, my Ritachka. Part of the decision to do a PhD, was so that we could do it at the same time. We struggled together, supported each other, (though I think you could still have been a bit more patient with me), and both gained some wrinkles and a few pounds in the process. And so, because I am pretty sure that without you, I would never have been a doctor, I am forever grateful to you.

Notations

R A rotation matrix $\in SO(3)$. 15, 43, 64, 125, 128, 129

t A translation vector $\in \mathbb{R}^3$. 15, 43, 64, 107, 116, 125, 126, 128, 129

T Expresses a pose (of a lidar, camera) in $SE(3)$. We propagate notations of a pose $\mathbf{T}_i^j$ to its translation $\mathbf{tr}_i^j$, and rotation $\mathbf{R}_i^j$ components. 15, 32, 38, 41, 43, 55–57, 63, 64, 67, 68, 108, 125, 126, 128, 129

RPE^{rot} Rotation component of the Relative Pose Error, expressed in percentage of the segment relative rotation, averaged over all segments. 14, 15

RPE^{tr} Translation component of the Relative Pose Error, expressed in percentage of the segment length, averaged over all segments. 14, 15, 18

ATE Absolute Trajectory Error, measuring absolute distance between the ground truth and the estimated poses (in meters). 15, 16, 131

ATE^{max} Absolute Trajectory Error Max, measuring the maximum absolute distance between the ground truth and the estimated poses (in meters). 16

RPE Relative Pose Error, measuring drift error averaged over all segments of a given length. 15, 16, 18, 72, 78, 79, 81, 83–87, 91, 109, 110, 112, 113, 115, 131

LO Acronym for 3D LiDAR Odometry. 32, 39, 41

LIO Acronym for 3D LiDAR Inertial Odometry. 32, 35, 39, 41

Contents

Chapter 1

Introduction

Résumé

L'objet de cette thèse est l'étude des algorithmes de SLAM pour les capteurs LiDAR 3D. Dans ce chapitre d'introduction, nous exposons le contexte de cette thèse, qui a été réalisée conjointement entre le laboratoire de robotique de l'école d'ingénieurs *MINES Paris-PSL*, et la société de logiciel *Kitware*. Nous présentons brièvement les capteurs LiDAR 3D et les capteurs IMU, avec lesquels nous travaillons dans cette thèse. Nous décrivons ensuite les motivations et les principaux enjeux derrière cette thèse, pour conclure par une présentation de l'organisation générale du manuscrit.

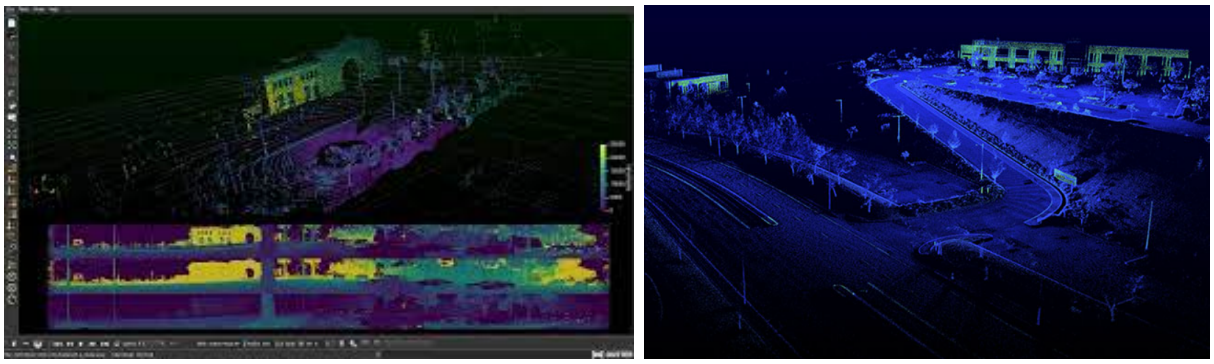


Figure 1.1: Left: a screenshot of OusterStudio, a specialization of LiDARView for Ouster sensors. Right: 3D reconstruction using the SLAM of LiDAR View developped by Kitware.

Context

This thesis is a joint work between the robotics laboratory of the *MINES Paris-PSL* engineering school, and the European antenna of the software company *Kitware*. *Kitware* is renowned internationally for its contribution to open-source, notably by C++ aficionados (which intersects largely with most robotics practitioners), for its excellent build manager *CMake*. However, *Kitware* also has strong expertise in scientific visualization and computer vision. The *Kitware Europe* (**KEU**) in particular has expertise in 3D Computer Vision, notably for LiDAR applications. Through their open-source project *LiDARView* 1.1, they monopolize the market of LiDAR visualizers and provide solutions for the principal manufacturers such as Ouster (*OusterStudio*), Velodyne (*VeloView*), etc...

On top of this expertise, they already had a strong experience with LiDAR SLAM, having developed their algorithm, integrated into *LiDARView*, and which is constantly refined and improved by the excellent computer vision team I had the honor to join during this thesis. It all began with a *LinkedIn* message from a famous influencer of the platform *Bastien Jacquet*, which also happened to be at the head of the Computer Vision team of **KEU**. I then contacted *Jean-Emmanuel Deschaud* and *François Goulette* who happened to be in Korea for a robotics conference, and we agreed to mount this thesis together, and together contribute to pushing the boundaries of 3D LiDAR SLAM algorithms.

What is 3D LiDAR SLAM ?

SLAM which stands for **S**imultaneous **L**ocalization **A**nd **M**apping is an important class of algorithms in the domain of robotics and computer vision. Given an acquisition platform equipped with sensors, capturing data moving through an unknown environment, *SLAM* aims to solve both the reconstruction of the environment and the localization of the sensor within this said environment. *SLAM* is related to many different fields in computer vision, notably 3D reconstruction, mapping, localization, etc... And many of the techniques are common and regularly exchanged between all these applications. One of the specificities of SLAM is that it is often used as a building block for larger robotics applications, notably for tasks involving robots either controlled or autonomous. This often puts constraints, or performance requirements for the SLAM to run online on a robotic platform, often with limited computation power.

SLAM is a generic term, but in practice, the algorithms adapted for a given application highly depend on the sensor suite of the hardware setup. In this work, we focus on SLAM for *3D LiDAR* sensors. However, there exists SLAMs for a variety of sensors (*2D LiDAR*, *RGB* images, *RGB-D*, etc..).

Figure 1.2: Examples of directional (top left), 2D (top right), 3D rotating (bottom left) and 3D solid-state (bottom right) LiDAR sensors. Images have been taken from the *Robotshop* website.

Each sensor has its specificities, and notably its weaknesses. To mitigate them, an interesting approach is to fuse the Data from multiple sensors to improve the performance of SLAM algorithms. In this work, we also incorporate Inertial Measurement Units (*IMU*) to improve the performance of our LiDAR SLAM.

3D LiDAR and IMU sensors

LiDAR sensors have existed for more than 30 years. LiDAR stands for **Li**ght **D**etection **A**nd **R**anging, and describes sensors measuring the distance traveled by the light emitted by a laser within the sensor. This allows measuring accurately the distance between the sensor and the point of the environment intersecting the ray of the laser. For time-of-flight technologies, the lasers send impulses that intersect with the environment. For diffusive materials, light is emitted back to the direction of the sensor and the time-of-flight is estimated by comparing the received signal with the original impulse, and the distance traveled can then be very accurately estimated.

Directional LiDARs have exploited this estimation of range for many applications notably obstacle avoidance. This technology has been integrated on rotating platforms to create *2D LiDARs*. These sensors rotate, capturing a trail of points intersected by their laser during the rotation. This stream of points localized very precisely *in the local frame* of the sensor, is often split into *frames* or *scans*. Typically, for rotating sensors, a scan corresponds to the complete revolution of the mobile unit within the sensor. These sensors have been exploited for more than 20 years with 2D SLAM algorithms to provide robots with the capacity to create 2D maps

Figure 1.3: Vacuum Cleaner Robot, possesses a 2D LiDAR which maps to establish the optimal trajectory to clean it.

of the environments. 2D LiDAR SLAM technologies are very mature, integrated within many industrial processes, and present in the home of many consumers (see figure 1.3).

2D Maps are interesting for many applications, however many others need to interact with complex 3D environments, and the ability to navigate in the 3-dimensional space is paramount. While this had been possible using images for some time, the apparition of 3D LiDAR sensors allowed the robotics community to push the boundaries in terms of the accuracy of 3D navigation. 3D LiDAR sensors (see figure 1 for examples of the different types of LiDAR sensors) leverage multiple lasers simultaneously to scan the environment in 3D dimension. Rotating units such as *Ouster* OS-0,1,2 or *Velodyne* VLP-16, HDL-32, HDL-64 sensors have multiple channels oriented at a different azimuth offset. Each sensor typically outputs frames at a resolution of $N_{channels} \times N_{width}$, where $N_{channels}$ is the number of channels (16 for VLP-16, 32 for HDL-32, 64, or 128, etc...), and N_{width} the number of points per channel (typically 1024 or 2048). The frame frequency, also defined by the frequency of a full revolution is typically set to 10Hz or 20Hz (the latter consuming more power).

Why are LiDAR needed, what are they useful for ? 3D reconstruction and 3D navigation have existed for nearly 20 years already, using cameras and building 3D structures from motion. So, why is 3D LiDAR getting so much attraction? One of the big drivers for LiDAR development has been the autonomous vehicle industry. While cameras are very important, LiDAR has unique complementary properties (signal even in a dark environment, very different noise model than cameras) which makes them very useful to be used along these cameras. In particular, images that are passive sensors need to recreate the 3D space from 2D light measurements. While this is perfectly possible using multiple cameras and stereo, more recently deep learning or a combination of both, the level of noise and precision for those methods cannot be compared to the accuracy 3D geometry provided by 3D LiDAR.

The field of SLAM is a great witness of this capacity: recently among the SLAMs tested on platforms with both cameras and LiDARs, the solutions using LiDARs are typically much more performant than those relying on cameras only. The reason is easy to understand: image-based

methods need to reconstruct the geometry of the environment, correlate this geometry with the image intensity/color values, and track changes in the image aspect to refine the geometry and estimate the motion. In contrast, LiDAR sensors already provide accurate geometry, so the "only" challenge is to track iteratively each new frame within the map.

Of course, this simplicity hides real challenges due to the specificity of these: the massive amount of points to process quickly, a non-negligible rolling shutter effect due to the continuous acquisition, noise and issues from highly reflective surfaces, etc... In this thesis, we will discover some of these challenges as we encounter them. But in short, the problem of 3D reconstruction with LiDAR simplifies to the problem of inserting each frame correctly concerning one another. This is the reason why sensors helping to estimate the trajectory are very complementary to LiDAR and are often part of state-of-the-art sensor-fusion solutions.

IMU sensors A **I**nertial **M**easurement **U**nit (IMU), is a sensor that measures values related to the motion of the platform it is rigidly attached to. A 6-axis IMU is composed of an accelerometer which measures the linear acceleration, and a gyroscope, measuring the angular velocity of the sensor. 9-axis IMU also has magnetometers that provide absolute orientation measurements, but in this thesis, we exclusively used 6-axis IMUs.

At our level, the principal variables determining the quality of the IMU are the measurement frequency and the level and characteristics of the noise. These two characteristics will determine the capacity of sensor-fusion systems (*ie* systems fusing multiple types of sensor information) to predict accurately the motion and leverage IMUs to put accurate constraints on the trajectory. Indeed, because IMUs only provide noisy derivatives of the motion of the platform they are attached to, the motion cannot be recovered from the IMU sensor alone and they are integrated into such sensor-fusion solutions.

Previously, mobile mapping was dominated by GPS-Inertial solutions. These measured the trajectory very accurately by fusing only GPS and IMU sensors and used the accurate trajectory to reconstruct the global point cloud of the environment by replacing scans (see figure 1.4). To be accurate, however, these platforms needed industrial-grade GPS and IMU sensors, which are still very expensive. Recently, there has been tremendous progress in Inertial LiDAR-SLAM solutions, which allow for new, cheaper and more flexible mapping alternatives. This is the subject of chapter 5.

Why study 3D LiDAR SLAM ?

I was often asked this question during this thesis. There was indeed a belief, perhaps because so much work had already been published on this subject, that the field was already completely saturated. Yet every year, more and more methods are still published and accepted in prestigious robotics conferences, why is that?

For many years, progress indeed seemed to have slowed down, notably due to the lack of publically available datasets to properly evaluate the research. But in the past couple of years, the lowering of the cost of LiDAR sensors as well as ambitious research projects have pushed existing algorithms to their limits and elevated the bar for the requirements of precision and performance expected of SLAM to unlock many industrial applications. LiDAR SLAM is living exciting days, and it was a privilege, during this thesis to have had a small part in this.

Contributions of the thesis and publications

In this thesis, we published two articles at top-tier international robotics conferences:

– *What's in my lidar odometry toolbox*, accepted at IROS 2021, see chapter 4

Figure 1.4: L3D2 acquisition platform constructed by the CAOR laboratory of MINES Paris-PSL. This platform relies on a costly industrial-grade GPS-INSS unit to construct a very accurate trajectory.

– *CT-ICP: Real-time Elastic LiDAR odometry with Loop Closure*, accepted at ICRA-2022, nominated for the *Oustanding paper award* 3

The main contributions of this thesis, validated by these publications are:

– A detailed comparison of Deep and Hybrid LiDAR odometry. *(IROS 2021)*

– A state-of-the-art LiDAR odometry, relying on an innovating motion distortion strategy of LiDAR frames. *(ICRA 2022)*

Both of these works also present secondary contributions, which will be detailed below. Furthermore, both have also been open-sourced and have been well-appreciated by the robotics community. At the time of this writing, the *CT-ICP* GitHub page [1], and `pyLiDAR-SLAM` GitHub page[2] where we published our work for the IROS2021 article have respectfully 481 and 225 stars.

Additionally, we are preparing work for publication, which extends *CT-ICP* as a LiDAR-Inertial system, and present this work in chapter 5.

Organization of this thesis

This document is organized as follows: First, in chapter 2 we present the methods, and datasets to evaluate our LiDAR odometry and SLAM methods. Secondly, in chapter 3 we present LiDAR Odometries, including our work *CT-ICP* and the related contributions. Then, in chapter 4, we focus on deep and hybrid LiDAR odometry, and we present the contributions of our article *What's In My LiDAR Odometry Toolbox*. Chapter 5 then presents LiDAR-Inertial odometry, including our work being prepared for publications.

[1] `https://github.com/jedeschaud/ct_icp`
[2] `https://github.com/Kitware/pyLiDAR-SLAM`

Chapter 2

Evaluation of LiDAR SLAM

Contents

Résumé

Beaucoup de travaux ont déjà été proposés dans le domaine des odométries LiDAR 3D. En effet, depuis le travail séminal de LOAM [135], plusieurs centaines d'articles ont été publiées, chacun proposant une propre approche singulière pour résoudre ce problème. Cependant, malgré tout ces travaux, il est souvent difficile d'extraire de fortes convictions sur les raisons sous-jacentes des performances d'une méthode.

Typiquement, chaque nouvelle méthode publiée évalue ses résultats, soit sur un jeu de données de référence public, soit sur un dataset privé, ou mis public pour occasion. Jusqu'à très récemment, il y avait très peu de jeux de données LiDAR 3D publics. Donc la plupart des méthodes étaient essentiellement évaluées sur le benchmark de KITTI [40], un agrégat de jeux de données, conçu pour faire avancer la recherche sur un large panel de tâches liées au véhicule autonome, et notamment l'odométrie LiDAR. Mais avec la forte réduction des prix des LiDARs, de plus en plus de jeux de données ont été rendus publics. Cela a permis une meilleure évaluation, du moins plus exhaustive, des odométries LiDARs.

Dans cette section, nous détaillons les méthodes et mesures d'évaluation des odométries LiDAR que nous utiliserons à pour l'ensemble de ce travail, ainsi que l'ensemble des jeux de données sur lesquels nous avons évalué nos algorithmes.

Le reste de ce chapitre est organisé de la manière suivante: tout d'abord en section 2.2 nous présentons donc les différentes mesures de précisions sur les trajectoires utilisés dans ce travail pour évaluer les différentes odométries LiDAR. Puis, dans la section 2.3, nous mentionnons la manière dont nous évaluons les performances computationnelles de nos algorithmes, et l'importance de concevoir des algorithmes rapides et efficaces. Finalement, dans la section 2.4 nous détaillons l'ensemble des jeux de données qui seront utilisés dans ce travail.

2.1 Introduction

As we presented in chapter 1, since the defining work of LOAM [135] in 2016, there has been a tremendous amount of work in the area of LiDAR odometry. All new methods need to prove a scientific contribution to be published. To prove these contributions, in the field of LiDAR SLAM and odometry, the algorithms are run on the sequences of public benchmarks ([40, 44, 136]) and datasets that either provide ground truth [13], scoring platforms, or both. While these evaluations provide objective results, from an industrial actor, or a researcher, ranking the relevance of these methods, or components of the methods for their specific use case remains challenging for multiple reasons.

On the one hand, old benchmarks [40] are saturated by novel methods, with minor performance improvements on the benchmark. The performance of these methods, sometimes, does not generalize well to other datasets, or in other contexts. For example, a SLAM performing well on KITTI [40], the most popular odometry benchmark for driving scenarios, might not perform as well when considering a sparser LiDAR (with 16 or 32 channels), or in other scenarios. On the other hand, while each method is supposed to explain its scientific contributions, in practice, it is extremely difficult to understand how each component of the overall method contributes to the presented performance. This is notably due to the growing complexity and refinement of the SLAM and odometry methods over the years. This is especially the case for a method that does not share its code online, in practice it is next to impossible to reconstruct a SLAM with on-par performance simply from an article alone, as many important details are often left out for these methods. *The devil is in the details* is particularly true when it comes to LiDAR odometries and SLAM.

Thankfully, as the interest in LiDAR-SLAM grew, so did the open-source methods publically available online. There is now an abundance of methods available ([25], [123], [84], [125], ...), which raises new, though much more comfortable problems: *How do we select/rank available methods objectively* ?

This is the subject of this chapter, where we aim to go deeper and present the relevant framework to evaluate the different methods against one another. We will use this framework to evaluate and compare each method in the remainder of this thesis.

The rest of this chapter is organized as follows, first in section 2.2, we present the different trajectory metrics used to evaluate slam and odometries, in section 2.3, we speak about the performance metrics to evaluate the runtime of the algorithms, and finally in section 2.4 the datasets which will be used to construct our evaluations.

2.2 Trajectory Precision Metrics

One of the main benefits of a LiDAR SLAM is its inherent precision, notably compared to visual SLAM. This is often demonstrated by looking at benchmarks such as KITTI's odometry Benchmark [40], where the top performing methods are LiDAR SLAM or odometries.

To demonstrate their scientific contributions, previous approaches have often focused on precision gains [135, 26, 131, 4], which are typically evaluated on standard datasets, predominantly the KITTI odometry benchmark [40]. The KITTI odometry benchmark focuses on odometry metrics, which aim to measure drift, typically without loop closure which we will cover in section 2.2.1.

However, for other tasks, it is the overall global accuracy of the trajectory which counts. This is typically the case for applications in a closed environment, where one typically expects multiple loop closures during the run of the SLAM. To evaluate the precision of SLAM for these use cases, global metrics are needed, and we present them in section 2.2.2.

2.2.1 Evaluating the drift: odometry metrics

When exploring open, new and unexplored spaces, all odometries drift over time. Measuring the amount of drift is paramount, as it gives an idea of the type of applications that a SLAM system can support. For example, if an odometry system has 500m of drift over a trajectory of 10km, the SLAM system would probably need sophisticated loop detection to close the loop, while for drift lower than 10m a simple ICP [5] based alignment might suffice. Figure 2.1 shows the drift accumulating over time.

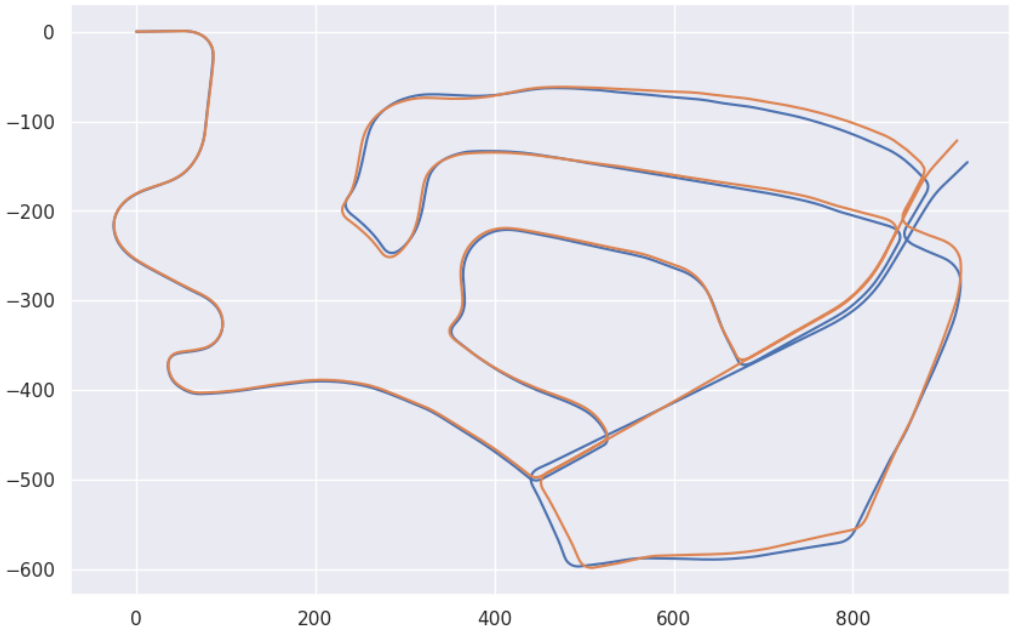


Figure 2.1: Drift accumulating over time for a LiDAR Odometry. The trajectory of the LiDAR Odometry (blue) accumulates errors over each frame registered, and this results in iteratively larger distance to the ground truth (orange) trajectory. *Dataset: KITTI-02* (see section 2.4).

The typical strategy to evaluate the drift is to compute a *Relative Pose Error* (**RPE**) over segments of fixed lengths, and average the results over all segments of this length in the trajectory. More precisely, as the relative pose error can be computed in rotation and translation, this gives two metrics: $\mathbf{RPE}^{tr}$ for the translation, expressed in percentage of the segment length, and $\mathbf{RPE}^{rot}$ for the rotation expressed in °/m, defined as follows:

$$\forall i \in \mathcal{I}^d, \quad \mathbf{T}^{gt}_{rel}(i) = \mathbf{T}^{gt}_{begin}(i)^{-1} * \mathbf{T}^{gt}_{end}(i) \tag{2.1}$$

$$\forall i \in \mathcal{I}^d, \quad \mathbf{T}^{odo}_{rel}(i) = \mathbf{T}^{odo}_{begin}(i)^{-1} * \mathbf{T}^{odo}_{end}(i) \tag{2.2}$$

$$\mathbf{RPE}^{tr} = \frac{1}{\#\mathcal{I}^d} \sum_{i \in \mathcal{I}^d} \frac{100}{\text{length}(i)} \cdot ||\mathbf{t}^{gt}_{rel}(i) - \mathbf{t}^{odo}_{rel}(i)||_2 \tag{2.3}$$

$$\mathbf{RPE}^{rot} = \frac{1}{\#\mathcal{I}^d} \sum_{i \in \mathcal{I}^d} \frac{|\text{AngularDist}(\mathbf{R}^{gt}_{rel}(i), \mathbf{R}^{odo}_{rel}(i))|}{|\text{length}(i)|} \tag{2.4}$$

Where $\mathbf{T}^*_*(i)$ denotes the pose at a beginning or end $(*_{begin}, *_{end})$ of a segment i among all segments (indexed by $\mathcal{I}^d$) of size at least d in the LiDAR trajectory, for the odometry or the ground truth $(*^{odom}, *^{gt})$.

The scale of the segments d needs to correspond to the relevant scale for the desired dataset. For mobile robotics, handheld acquisition setups and driving scenarios, the scale of the motion is typically very different. In section 2.4, we will precise for each dataset the length d of the segments considered for the evaluation, and we give in figure 2.1 the relevant scales for the different problems.

Application	**Driving/Autonomous Vehicle**	**Mobile Robotics**	**Indoor, Handheld Sensors**
Segment Length (m)	500m	100m	20m

Table 2.1: Segment length for the **RPE** metrics for the type of SLAM applications

While we provide the two metrics $\mathbf{RPE}^{tr}$ and $\mathbf{RPE}^{rot}$ to evaluate the drift, in practice $\mathbf{RPE}^{tr}$ is mostly sufficient to evaluate and rank odometries, as local rotation errors also lead to translational drift. So in most cases, we principally present results for the $\mathbf{RPE}^{tr}$ metric, unless the $\mathbf{RPE}^{rot}$ becomes necessary for finer analysis related to rotations.

2.2.2 Absolute Trajectory metrics

For some problems or use cases, global trajectory metrics are more relevant than relative pose metrics. This is the case, for example when navigating in closed environments, where we expect many loops, such as a warehouse, a building, or a taxi circuit.

Thus the typical evaluation for full LiDAR SLAM systems (which also integrates loop closure), is the *Absolute Trajectory Error* (**ATE**), which computes the distance between the estimated pose of the LiDAR and the ground truth. As some rotation error in the first few frames may have a significant impact on these metrics, a typical approach computes an optimal rigid transform between the ground truth and the LiDAR trajectory by least square minimisation, and the **ATE** is computed on this modified trajectory, as follows:

$$\mathbf{T}_{rigid} = \underset{\mathbf{T} \in SE(3)}{\text{argmin}} \sum_{i \in \mathcal{I}^{lidar}} ||\mathbf{T} * \mathbf{t}^{slam}_i - \mathbf{t}^{gt}_i||^2_2 \tag{2.5}$$

$$\forall i \in \mathcal{I}^{lidar}, \ \mathbf{T}^*_i = \mathbf{T}_{rigid} * \mathbf{T}^{slam}_i \tag{2.6}$$

$$\mathbf{ATE} = \frac{1}{\#\mathcal{I}^{lidar}} \sum_{i \in \mathcal{I}^{lidar}} ||\mathbf{t}^*_i - \mathbf{t}^{gt}_i||^2_2 \tag{2.7}$$

The **ATE** measures the mean of the absolute pose error for each pose of the LiDAR (which is typically defined as the pose at the beginning or the end of a LiDAR frame). Also relevant

is the $\mathbf{ATE}^{max}$, which expresses the maximum trajectory error between the ground truth and the corrected trajectory (also in meters). For example, for robots navigating in a warehouse, the precision specifications for a SLAM system will typically be expressed as a threshold on the $\mathbf{ATE}^{max}$ rather than the **ATE**.

Even more than the **RPE**, the **ATE** requires *accurate ground truth* to be informative. If the ground truth is established with noisy GPS measurements, for example, the **ATE** will be directly impacted. By comparison, the noise can be compensated in the **RPE** by selecting longer segments (this is notably why the KITTI benchmark averages the **RPE** over segments of lengths varying from 100m to 800m). In any case, the precision is a matter of scale. For very long trajectories over multiple kilometers, a noisy GPS ground truth precise up to 1-5 meters, might be sufficient to correctly evaluate the quality of the SLAM. Thankfully, there has been a tremendous effort from the research community to propose benchmarks and datasets with increasingly accurate ground truth, which will drive the evaluation of our methods, and that we will present in section 2.4. In this section, we focus on the evaluation of LiDAR odometries, to evaluate full SLAM (with GPS or loop closure constraints), [59] offers a more comprehensive approach.

2.3 Performance

In the previous section (2.2), we mentioned the precision metrics for our evaluation of LiDAR odometry methods. Another important aspect is the runtime performance of the algorithm. As we mentioned earlier, LiDAR SLAM is often used in live contexts, running on robotics platforms with computational constraints (such as our small vacuum robot presented in figure 1.3). For these contexts, LiDAR SLAM needs to be fast and precise, though a tradeoff is often necessary between the two objectives. Thus, in this work, we will also mention as an important criterion the execution speed of the LiDAR SLAM algorithm we evaluate.

LiDAR sensors provide a stream of LiDAR frames at a certain frequency (typically 10Hz or 20Hz, the latter being highly recommended). Thus, for an algorithm to run live, it needs either to run at higher frequencies or be tolerant to dropping frames. Indeed, if the LiDAR odometry runs at a lower frequency than the sensor output, if all frames waiting to be treated were kept in memory, the memory of the platform would explode over time.

In this work, we will thus focus on the average runtime of our LiDAR odometries, which gives a good first-order indication of the potential of a method for online use. Indeed, an average runtime below the frame acquisition time should guarantee that keeping a buffer of frames waiting to be processed should not explode over time.

To evaluate the runtime of each algorithm, we will use a laptop doted with an Inter Core i7-9750H CPU@2.50GHz with 6 cores and 12 threads, and equipped with an NVidia 2080 Max-Q graphics card. When evaluating the runtime of LiDAR odometries, we only measure the runtime of the algorithm and not data loading, or the conversion which can occur. This includes preprocessing, registration, map updates, etc..

2.4 Datasets

While in previous sections (2.2, 2.3), we detailed the metrics to evaluate our different methods, we now present the datasets on which they will estimate these metrics.

While for a long time, there has been a shortage of publically available LiDAR datasets, with the KITTI [40] remaining the main dataset of interest, in recent years, there has been an emergence of publically available LiDAR Datasets to foster research in multiple domains including object detection, segmentation, and also for evaluating odometry and SLAM [136]. Among

the publically available datasets, we made a selection based on multiple criteria including: the availability and accuracy of the ground truth, the diversity of the scenarios, environment and motions, the challenging aspect of the dataset, and finally their popularity.

We grouped the datasets into three main categories, corresponding to the three categories presented in figure 2.1, and each associated with a relevant *distance* or *segment length*. Thus, we first present in section 2.4.1 the driving datasets we work with. For these datasets, the sensor placed on a car is moving fast, in open environments, but with relatively slow rotations and accelerations compared with the sensor acquisition speed.

Then, in section 2.4.2 we present the mobile robotics datasets. This time the sensors are placed on smaller robots, which typically move in smaller, sometimes closed environments. The speed of the sensors is typically much slower than a car, but accelerations can be much greater, especially when considering rotations.

Finally in section 2.4.3 we introduce handheld datasets. The sensors are placed on a platform that is handheld and moves with the person carrying the platform. This often leads to slow-moving sensors, but typically even jerkier motions than for mobile robots, with also great accelerations and fast rotations.

2.4.1 Driving Dataset

The quest for the autonomous vehicle has been a driving factor for the acceleration of both the industry and the research community's interest in LiDAR solutions. In this context, the LiDAR can typically be used for object detection, but also navigation or localization, for which SLAM and odometry are often important building blocks of much larger complex industrial data processing pipelines.

Pushed by this desire to accelerate the research for the autonomous vehicle, the KITTI [40] benchmark has long been a reference for evaluating odometry and SLAM. It is still widely used today and remains one of the most popular benchmarks, and we present it in section 2.4.1.1. We selected also a synthetic dataset generated using the CARLA simulator which we present in 2.4.1.2.

2.4.1.1 The KITTI Odometry Benchmark [40]

Initially published in 2013, by the Karlsruhe Institute of Technology, the *KITTI Vision Benchmark Suite* proposes a series of benchmarks for different relevant tasks in the context of autonomous vehicles. Thanks to a set of sensors mounted on a car, including a Velodyne HDL-64 sensor (see figure 2.2), and multiple cameras, multiple sequences we acquired of varying lengths, which were then labeled with ground truth for multiple tasks (by hand for segmentation tasks, object detection, and using GPS data for odometry).

Some of the labeled sequences were made publically available, while for others the labels are hidden, and are used to rank methods for the series of live benchmarks which are still open today. Among them, the *odometry benchmark* is the one relevant for our application and has been the reference benchmark for evaluating LiDAR odometry and SLAM methods since LOAM [135] appeared.

The benchmark contains 11 sequences with ground truth and 11 more sequences with hidden ground truth (used for the benchmark evaluation). Thus, we only consider the sequences for which we have access to the ground truth, *ie* the $\sim$ 20000 frames of the 11 initial sequences. These acquisitions comport different urban scenarios, mostly in suburban areas, with a sequence on a highway, the sequence `01`.

Another important point is that for KITTI's benchmark, the LiDAR frames have previously been distorted to compensate for the motion. For most of the sequences considered (everyone except the sequence `03`), we have access to the *raw* KITTI frames, *ie* without distortion. Using

Property	Abbrev.	Description
Environment		
Rural	R	Vegetation, linear and flat environments.
Suburbs, Residential	S	Small buildings, narrow roads, low driving speed.
City	C	Tall buildings, narrow or wide streets, many stop at lights, low driving speed.
Road, Highway	H	Few geometric features (linear environments), high driving speed.
Outdoor	O	Open environments, tree and vegetation, geometric features with large scale.
Indoor	I	Indoor environments, geometric features of smaller scale, entry to unexplored environments, new rooms.
Small Scale Indoor	SI	

Table 2.2: Table of abbreviations of the different properties of a dataset sequence used in this thesis

these "*raw*" point cloud frames, which only have the x, y, z properties, we can approximate the timestamps of each point by linearly interpolating based azimuth (or *yaw* value). This will be important to demonstrate on KITTI the relevance of compensating the motion distortion in Chapter 3.

More recently, the same team published a new benchmark named KITTI-360 [68], acquired with the same platform as KITTI's original benchmark. The benchmark contains 8 additional sequences with ground truth, with notably some much longer sequences (ranging from 3000 to 15000 frames), for these sequences the motion was not compensated, so similarly to the *raw* sequences, we interpolate the timestamps using the azimuth angle.

Finally, it is well known that the LiDAR for KITTI [40] and KITTI-360 [68] have calibration issues. So for every algorithm, and on each frame from the dataset, we apply the same correction which we presented in our article [25], and which corrects the calibration of the KITTI LiDAR.

We summarize the content of the dataset in table 2.3.

On KITTI Odometry benchmark's metric The KITTI odometry benchmark evaluates algorithms using its metrics $\mathrm{tr}_{err}, \mathrm{rot}_{err}$ which are variants of the **RPE**. More precisely, these metrics are the averages of the $\mathbf{RPE}^{tr}$ for different segments lengths ranging from $100m$ to $800m$, (averaged over all segments of all lengths). This rather convoluted metric was designed to assign a unique score to methods using a sometimes imprecise GPS ground truth. We will sometimes use this metric (notably in chapter 4), to compare a wide range of methods on KITTI without running the algorithm again, as most methods present their results using this metric for better comparison.

2.4.1.2 KITTI-CARLA [27]: A KITTI-like synthetic Dataset

Our second driving dataset was acquired using the CARLA simulator [29]. The CARLA simulator is designed to advance the research for autonomous vehicles. Constructed on top of the Game Engine Unreal Engine [34], it is essentially a Video Game, in which a comprehensive set of sensors is implemented, ranging from realistic RGBD cameras (with photorealistic effects), LiDAR, radar, event cameras, etc…

Figure 2.2: KITTI[40] dataset acquisition platform (Top Left). The LiDAR for the acquisition sensor for the setup is an HDL-64 Velodyne (Top Right), placed in position *Autonomous vehicle*, *ie* with intersecting mostly the ground and points below the sensor. The bottom image shows an illustrative LiDAR frame colored by the timestamp of each point.

Sequence	Properties	Number of Frames
KITTI		
00	Suburbs,Outdoor	4541
01	Highway,Outdoor	1101
02	Suburbs,Outdoor	4661
04	Suburbs,Outdoor	271
05	Suburbs,Outdoor	2761
06	Suburbs,Outdoor	1101
07	Suburbs,Outdoor	1101
08	Suburbs,Outdoor	4071
09	Suburbs,Outdoor	1591
10	Suburbs,Outdoor	1201
KITTI-360		
00	Suburbs,Outdoor	11501
02	Suburbs,Outdoor	19231
03	Suburbs,Highway,Outdoor	1030
04	Suburbs,Outdoor	11400
05	Suburbs,Outdoor	6723
06	Suburbs,Outdoor	9698
07	Rural,Highway,Outdoor	3161
09	Suburbs,Outdoor	13955
10	Suburbs,Outdoor	3743

Table 2.3: Properties of each sequence of the KITTI and KITTI-360 datasets

Figure 2.3: Images describing the sensor output of the CARLA dataset. **Left:** image from a simulated photorealistic RGB sensor. **Middle:** a sample LiDAR frame simulated with the engine. **Right:** the road layout of the map `Town10`.

The sensor outputs are accessible via an API and allow the simulation of a parametrizable sensor suite on top of a vehicle that drives autonomously in one of the 8 maps predefined in the simulator. Figure 2.3 presents the sensor output and the environment of CARLA.

The great feature of using a synthetic simulator is the access to the exact ground truth (for the CARLA simulator, this includes ground truth for segmentation data, object detection and also for poses). However, it does have drawbacks. First of all, currently, the vehicle model does not describe the accurate physics of vehicles on a road (Jerky movements, aggressive rotations, no simulation of the wheel's distortion, etc…). Secondly, the environment in which the vehicle navigates is extremely simplistic, it consists of repeatable structures, often extremely planar, with little variation, and of overall small scale for the context of driving (as seen notably by the largest map available in figure 2.3).

KITTI-CARLA Thus our dataset KITTI-CARLA [27], is therefore constructed with the CARLA simulator (version `0.9.10`) by designing a sensor suite copying almost identically KITTI's vehicle setup. The dataset has 7 sequences for 7 of the 8 default maps available in the simulator (`Town01-07`). For each sequence, the default autonomous agent is driving in the map for a total of 5000 frames, with a 64 lasers lidar simulated at 10Hz. The distortion of the scan due to the motion is simulated, so each point of each frame is timestamped accurately.

2.4.2 Mobile Robotics Dataset

After the autonomous vehicle industry, another area that has been more and more vested in LiDAR SLAM and mapping is the mobile robotics domain. Robots are becoming more and more available, and their use in the industry has grown exponentially in recent years. Robots are revolutionizing the retail industry for the automatic management of warehouses with various degrees of autonomy, for performing safety inspections, for the Defense area, etc…

For most applications, and as the unit price of LiDAR sensors is decreasing steadily, they are becoming a viable choice in the localization pipeline of the sensor. We consider a large-scale mobile robotics dataset in this thesis: NCLT [13] which we present in section 2.4.2.1.

2.4.2.1 NCLT [13]

The dataset was published by the *Michigan State University* in 2013, to advance localization in a predefined environment across multiple seasons. It consists of 27 sequences acquired with a sensor acquisition system built on top of a controllable segway, including a LiDAR, an IMU and a 360 camera. The complete system is described in figure 2.5.

The 27 sequences consist of long runs (up to a couple of hours of sensor acquisition), within the *Michigan State University Campus*, across multiple seasons. It was designed for studying Long-Term localization, in varying environments (appearance of snow, different lighting conditions, etc …). Figure 2.4 shows multiple runs of the sensor overlaid on top of each other.

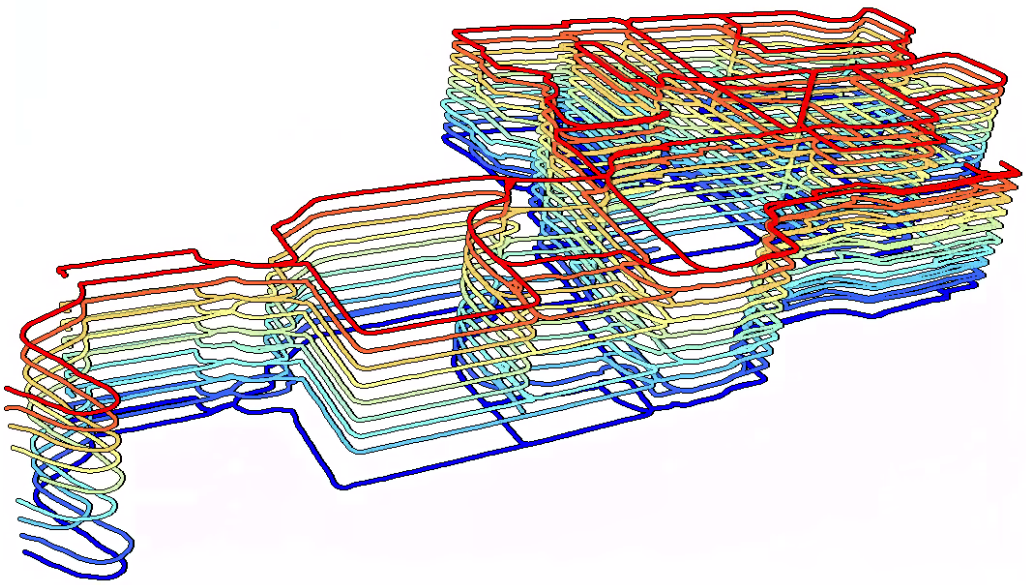


Figure 2.4: Multiple trajectories of NCLT's[13] platform across the Michigan University Campus, overlaid on top of each other.

The typical trajectories cross multiple types of environments (Roads surrounded by vegetation, small buildings, outside of large buildings, parking lots, etc…) This is in itself challenging, as the localization algorithms need to adapt to environments of different scales. Another difficulty of the dataset is the segway the platform is mounted on. It leads to trajectories with jerky or awkward motion especially, where each terrain irregularity is felt, with additional strong rotation in roll (during acceleration, or deceleration phases) and yaw (when the segway is turning around itself).

Finally, the RTK GPS does provide a very accurate ground truth for each of the 27 sequences. The sequences will be labeled by their date of acquisition in tables, (for example `2012-01-08` is the first sequence, and `2013-04-05`, the last).

2.4.3 Handheld Datasets

The final category of datasets describes a new use case, that of the handheld sensors suite. Mobile mapping solutions carried by personnel have been around for some time, with projects such as Google's Cartographer [45] enabling the use of backpack solutions to map environments so far only accessible by foot.

Current survey/mapping solutions are long and expensive, as traditionally static scanners need to stay still until a scan is completed before moving it, mapping an environment one scan at a time. But 3D LiDARs offer an alternative. Mounted on a device carried by a person (backpack, stick or handheld), the data can be continuously acquired while crossing the environments, leading to much faster acquisition time.

In this context, multiple datasets and benchmarks have been proposed to evaluate the capabilities of LiDAR SLAM. Notably, Oxford University proposed the *Newer College Dataset*

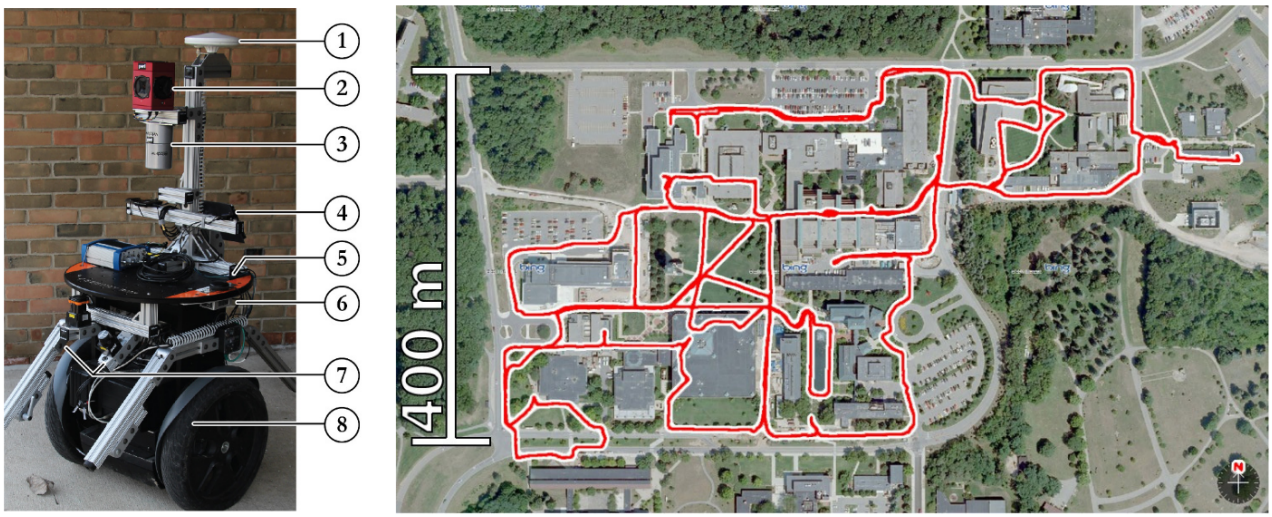


Figure 2.5: The Segway platform used for collecting the NCLT dataset [13]. Among the sensors are present: an RTK GPS (1), a 360 camera (2), an HDL-32 Velodyne 3D LiDAR (3), an IMU (4) and a consumer-grade GPS (5). On the right, a sample trajectory overlayed on top of a satelite image of the campus. (Images are from the original article [13]).

[98], we present it in section 2.4.3.1. More recently, the *HILTI SLAM Challenge* presented in 2021 at *IROS* and in 2022 at *ICRA* for the second edition aimed to evaluate the precision capacities of state-of-the-art SLAM systems, with very high-quality ground truth. We present this benchmark in section 2.4.3.2.

2.4.3.1 Oxford Dataset [98]

The *Newer College Dataset* was acquired within and outside of the *Oxford University Campus*. The sensor suite has a 3D LiDAR (an Ouster sensor, with either 64 or 128 lasers), an IMU and a stereo camera system. The system is mounted at the end of a stick, and held by a human operator. Figure 2.6 illustrates the whole system with the operator.

The dataset itself consists of multiple runs of the operator across the campus' courtyard and neighboring streets. The environment comports different types of areas, including open and confined and vegetated areas. The sequences themselves have different lengths, often with loop closures, as the campus itself is rather small.

The challenge resides more with the motion than the environment. The environment itself is rather simple, but because the system is handheld on a stick, the sequences presented have irregular, jerky motions and fast rotations. To make the dataset even more challenging, some sequences were purposefully made difficult by swinging the stick around, to challenge visual and LiDAR SLAM systems.

The ground truth was computed accurately, using a survey-grade laser scanner to obtain map with millimetric precision. The ground truth poses were then obtained by registering the Ouster point cloud within the point cloud map using an Iterative Closest Point algorithm.

2.4.3.2 *HILTI* Challenges [44, 136]

Presented initially in 2021 at the *IROS* conference, the goal of the challenge was to test and advance SLAM and mapping solutions in the construction industry. From *HILTI*'s point of view, the construction sector could tremendously benefit from very accurate mapping solutions, for building inspections, progress monitoring, site safety verifications, etc...A second edition

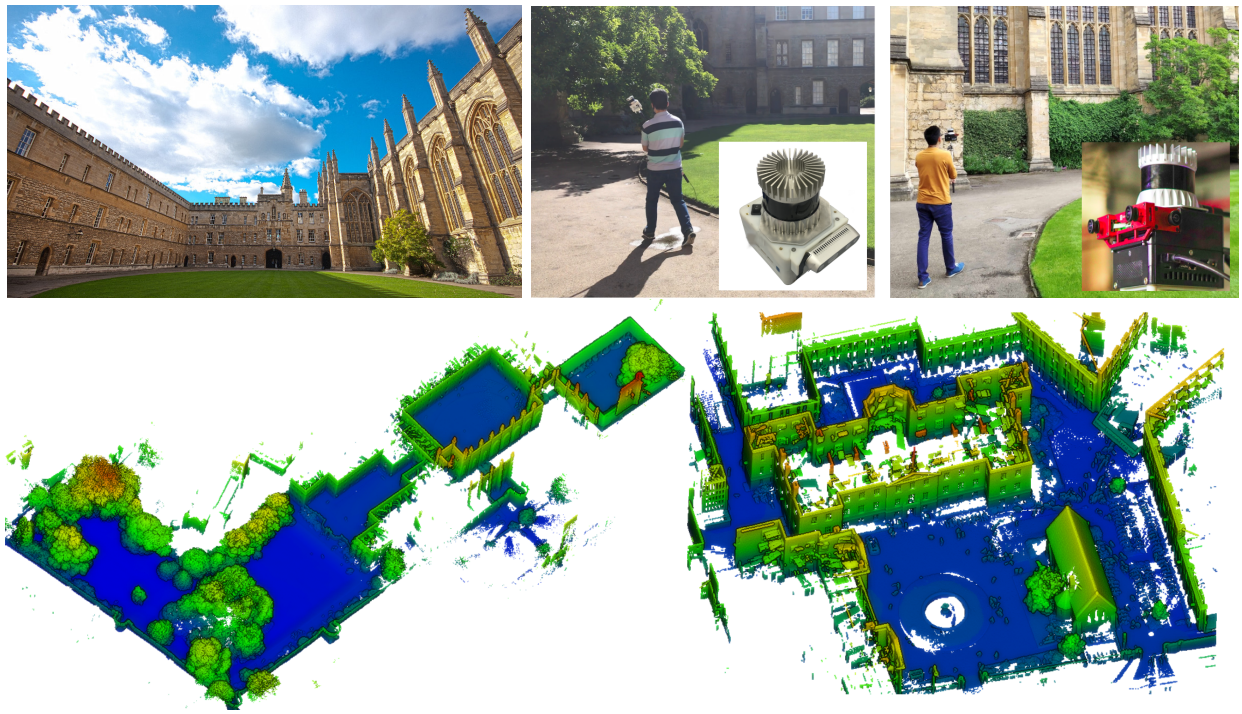


Figure 2.6: *The Newer College Dataset.* **Top Left:** the inner courtyard of the Oxford University campus. **Top right:** the handheld sensor acquisition setup. **Bottom:** the reconstructed ground truth map scanned with survey grade LiDAR. Images are from the dataset's web page `https://ori-drs.github.io/newer-college-dataset/`.

occurred in 2022 with an even more challenging set of sequences, in partnership with Oxford University [136].

For both competitions, a set of sequences is acquired using a handheld sensor suite comprising a LiDAR, an IMU and a system with multiple cameras. Some of the sequences are provided with ground truth, and others have a hidden ground truth and are used for ranking methods in a live benchmark. The ground truth themselves have a millimetric level of precision. Similarly to the previous dataset (see section 2.4.3.1), a survey-grade fixed LiDAR scanner was used to generate a highly precise point cloud, and the sensor is placed in millimeter accurate reference points placed along the trajectory. For the rest of the poses, between the reference points, the poses were estimated precisely using the reflective targets placed on top of the sensor.

The 2021 dataset contains 12 sequences in varied environments, ranging from empty parking lots to offices, basements or construction sites. Only 6 sequences have ground truth poses. The acquisition system for this dataset consists of a 64 laser Ouster sensor, with a directional Livox sensor for LiDAR sensors, as well as a camera system with an IMU (see figure 2.7). This dataset had many challenges, including complicated environments, notably open and featureless environments like empty parking lots.

The 2022 competition was designed to be even more challenging than the previous one. The setup is different, using a *Hesai XT-32*, which provides very precise measurements, mounted on a small platform with a multi-camera system and an IMU. The system is deployed for multiple sequences in a construction site, and to map both the outside and the inside of the Sheldonian Theater in Oxford. In total, the dataset comports 16 sequences, 8 for the challenge and the live leaderboard, and an additional 8 with ground truth poses. Some sequences are particularly challenging, both because of the environments (long corridors, close confined spaces, stairs, changing in scale) and because of the motion of the sensor, similarly to previous handheld datasets is jerky with great acceleration but with relatively low speed.

These datasets were designed to test *Sensor-Fusion* SLAM Algorithms, as its sensor suite includes vision, LiDAR-based and inertial sensors. Interestingly, the results show that the best-performing methods do not even need to leverage cameras, as Inertial Sensors and LiDAR are sufficient to dominate both benchmarks. We will mainly use this dataset in chapter 5.

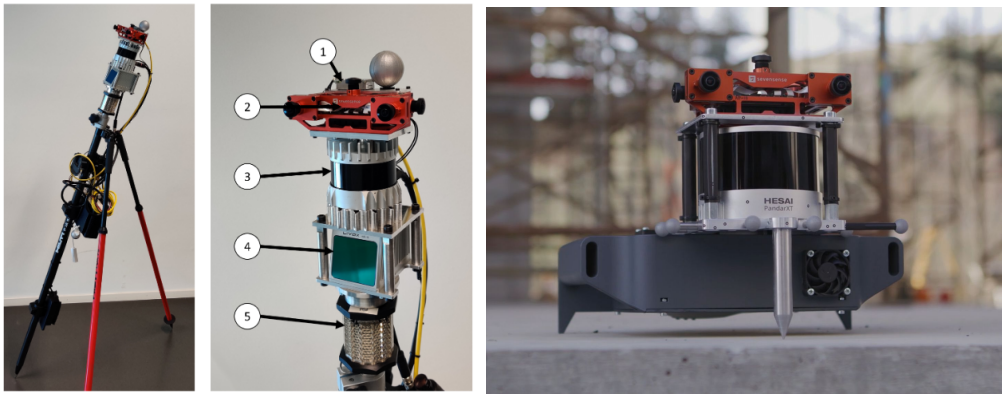


Figure 2.7: The sensor setups for the *HILTI* 2021 [44] and 2022 [136] competitions. On the left the Phasma stick of the 2021 competition, carrying an Ouster 64 (3), a Livox (4) and an Alphasense camera system (2) with an integrated IMU. On the right, the system was reduced in size for the 2022 competition, to be held in front of the operator, and carries a HESAI XT-32. Images were taken from the challenge's web page `https://hilti-challenge.com/index.html`.

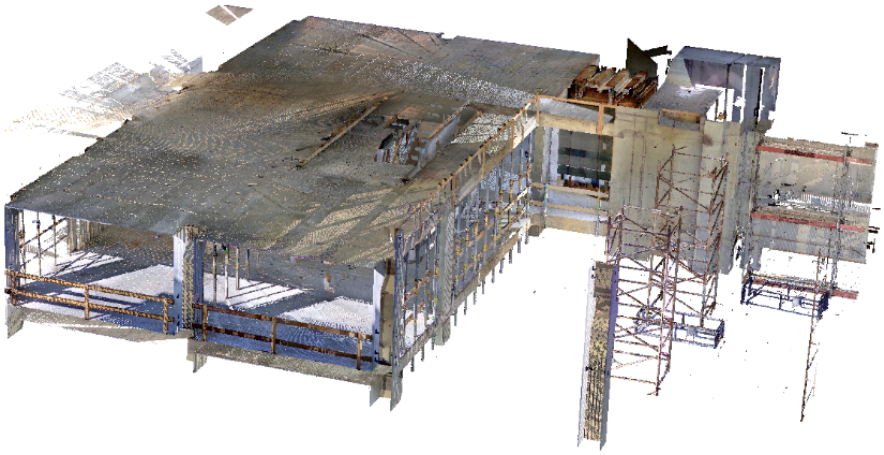


Figure 2.8: **HILTI 2022**'s construction environment for sequences `exp04,exp05,exp06`

2.5 Conclusion

We presented in the chapter the main tools we will use throughout this thesis to evaluate, rank and analyze the different odometry and SLAM methods that we will present, either external methods we will run, or our own. First, in section 2.2 and 2.3, we presented the different metrics, and values that we shall track and look for in algorithms.

Secondly, we presented a large number of datasets that provide an extensive (though not exhaustive) test bed to evaluate, compare and rank different methods we will present in a variety of different conditions, environments and trajectories. While we will not run each algorithm on each sequence of each dataset and will allow us some shortcuts when relevant, we will exploit their full range to make the most relevant analyses and comparisons and draw deep insights and conclusions of the inner workings of a multitude of methods.

Thus, we are now equipped to analyze and understand LiDAR odometries performances, and we start in the next chapter with classical LiDAR odometry methods.

Chapter 3

LO: LiDAR(-only) Odometry

Contents

Résumé

Les odométries LiDARs forment une partie essentielle de la plupart des systèmes de SLAM et de mapping pour ces types de capteurs. Nous pouvons définir une odométrie LiDAR comme un algorithme qui recale itérativement chaque nouveau nuage de point fourni par un capteur LiDAR (aussi appelé "frame") sur une carte reconstruite au fur et à mesure des insertions. L'aspect important, est qu'une odométrie est la meilleure estimation d'un système en l'absence de fermeture de boucle, et en l'absence de contraintes absolues sur la trajectoire (type GPS). Une odométrie est typiquement intégrée dans un système plus global définissant un SLAM, en fournissant des contraintes de trajectoire relatives entre les frames consécutives. Ces contraintes peuvent ensuite être intégrées à des modélisations probabilistiques de trajectoire, utilisant typiquement des "Pose Graph" [61], et fusionnées avec d'autres types de contraintes pour optimiser globalement la trajectoire.

Dans ce chapitre, nous nous concentrons sur les odométries LiDAR pures. C'est à dire, sur le problème d'estimation de trajectoire n'utilisant que le flux de nuage de point brut fourni par un capteur LiDAR, sans référencement GPS, ni fermeture de boucle.

Le reste de ce chapitre est organisé de la manière suivante: Tout d'abord, nous présentons dans la section 3.3 une odométrie LiDAR de référence, simple, implémentée en python, et qui fait partie de notre projet open-source `pyLiDAR-SLAM`. Puis, en section 3.4, nous introduisons notre méthode *CT-ICP*: une odométrie LiDAR état de l'art en terme de précision, avec une efficacité lui permettant d'être utilisée en temps réel. Finalement, dans la section 3.5, nous présentons une batterie d'expérimentations et d'analyses nous permettant de comprendre avec précision les raisons des performances, mais aussi des limites des deux méthodes proposées.

Nous détaillons en dessous les contributions principales présentées dans ce chapitre:

Contributions Principales

- Une odométrie LiDAR simple, implémentée en python, atteignant néanmoins l'état-de-l'art sur KITTI [40].
- Une nouvelle méthode repoussant l'état-de-l'art d'odométrie LiDAR sur un large panel de jeux de données: *CT-ICP*.

3.1 Introduction

LiDAR odometries are an essential building block of many SLAM and mapping systems. A LiDAR odometry can be defined as the algorithm which iteratively registers LiDAR frames onto a map built online. Another definition could be the frontend of a LiDAR SLAM system without loop closure or absolute positioning information. Indeed, a LiDAR odometry is typically integrated within a larger SLAM system by providing relative trajectory constraints to a backend, typically a pose graph [61] or a factor graph [23], which optimizes globally the trajectory and then the map can be updated if necessary.

While this simple view is true in many cases, in reality for many others, things get a little more blurry as subsystems can be much more coupled to each other. Indeed, it is not always possible to properly isolate an odometry component, as the map is sometimes updated after the pose graph optimization. Others perform a sliding-window-based optimization on multiple frames or have a tighter coupling between the backend and the frontend. Nevertheless, we will adopt this view in this work in the interest of clarity.

Thus, this chapter focuses on LiDAR-only Odometries, *ie* the problem of estimating the trajectory of the LiDAR sensor given only its stream of outputs, and without any loop closure. We first present related work in section 3.2. Then we present a first baseline LiDAR-Odometry, implemented in python as part of our `pyLiDAR-SLAM` project, in section 3.3. Then we introduce our state-of-the-art solution: *CT-ICP*, which is a superior LiDAR odometry method, with more complexity. *CT-ICP*'s performance, including on raw sensor data, is notably due to its motion distortion strategy, and we present this work in extensive details in section 3.4. Finally, in section 3.5, we experiment thoroughly with both methods and through a detailed ablation study, clarify the contribution of different aspects to the overall performance of the *CT-ICP*.

Both the methods proposed in the chapter, allow a locally very precise estimation of the trajectory, which in turn allows the precise estimation of geometry at a large scale, as illustrated in figure 3.1. We summarize below the principal contributions presented in this work:

Principal Contributions

- A simple LiDAR Odometry implemented in python, which still reaches state-of-the-art performance on KITTI [40].
- A state-of-the-art LiDAR Odometry, *CT-ICP*, for a wide range of datasets, thanks to its distortion handling procedure.

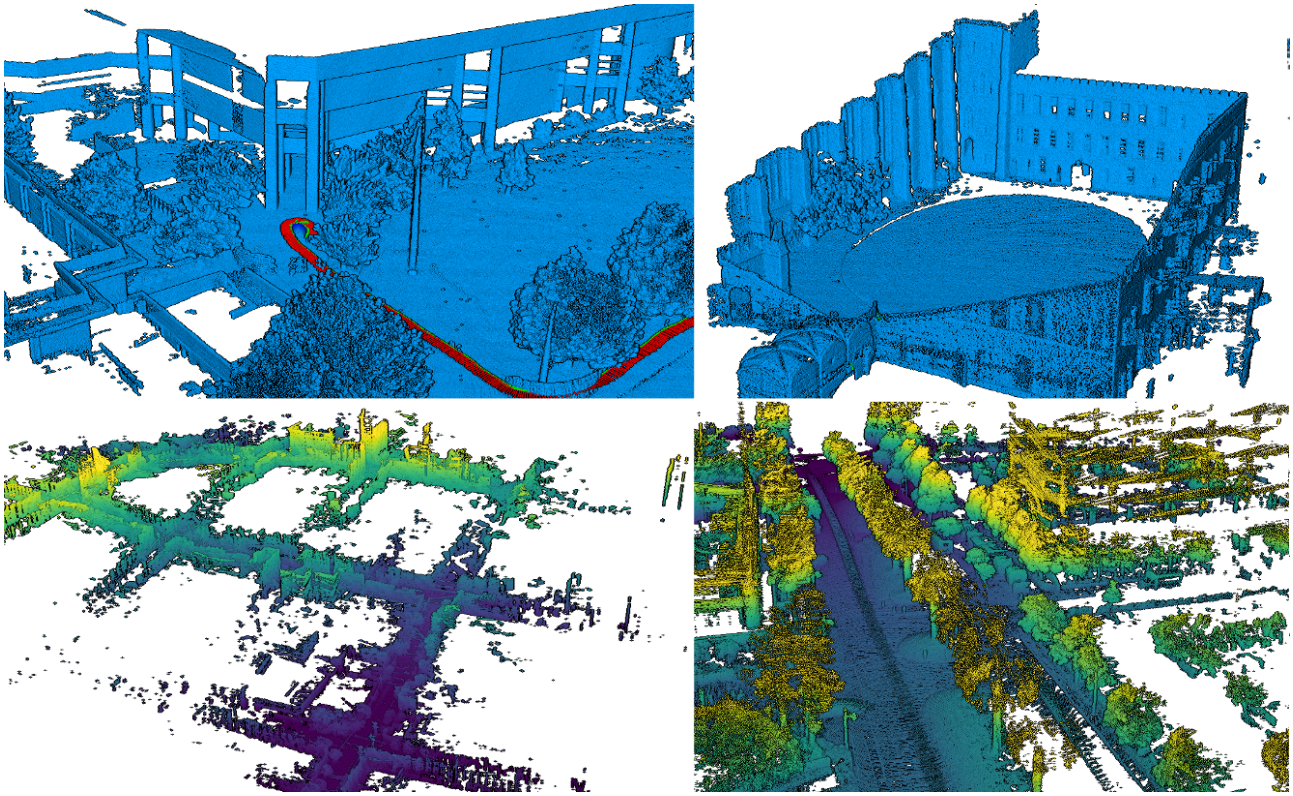


Figure 3.1: Dense point clouds reconstructed by aggregating frames using the trajectory estimated by our `pyLiDAR-SLAM` odometry.

3.2 Related Work

As previously mentioned, there is a massive body of work focusing on LiDAR odometries. Early methods such as [114, 10, 82, 14] made the transition from 2D (3DoF) towards full 3D (6DoF) trajectory estimation, using emerging 3D LiDAR technology, or assembling 3D scans by integrating 2D lasers mounted on moving structures. This paved the way for modern LiDAR odometries designed for modern LiDAR sensors, of which LOAM [135] is without contest the seminal work. Yet, on the one hand, LOAM [135] has been extended and iterated on by countless other methods [111, 134, 123, 69, 133, 47, 102]. On the other hand, many approaches have been proposed with significantly different base formulations [26, 4, 30, 84]. But, on a high level, all methods have essentially the same construction.

Initially, the stream of LiDAR points is cut into frames, each frame has a considerable amount of points (up to 10^6 points per second for a HDL-64). For rotating LiDAR sensors (Velodyne VLP-16, HDL-32 or 64, Ouster sensors, etc..), a frame corresponds typically to a 360° revolution of the unit, for an acquisition frequency of 10 to 20Hz. See figure 3.2 for examples of frames for the different sensors. This operation is typically done not by the algorithm itself, but by the driver provided by the manufacturer. So the input of each LiDAR odometry is a sequence of LiDAR frames.

Each frame is pre-processed to reduce the number of points to handle, and the computation burden that comes with it (typically this step can include sampling, key points extraction, semantic classification, etc...). The frame is then registered within the current map using a variant or extension of the Iterative Closest Point algorithm (ICP) [5]. The output of the registration is the pose of the new frame located in the world (typically referenced by the pose of the initial frame when no GPS is involved). Then the frame is inserted in the map, and the motion model which predicts the initial guess before the registration is also updated. This is summarized in figure 3.3.

The most significative element of the pipeline is the pair *Map/Registration Algorithm*. The two are strongly coupled: if the map stores a set of features, a dense point cloud or distribution information, the corresponding procedure registration will differ. We first look at the related work from this perspective in section 3.2.1.

Another important aspect is the trajectory representation. This includes many aspects, from the motion model to the point cloud distortion or even the registration procedure. We present the related work through this lens in section 3.2.2.

Finally, in section 3.2.3, we present different acceleration strategies which were proposed to improve the performances of the SLAM (including the different research structures proposed for neighborhood computation).

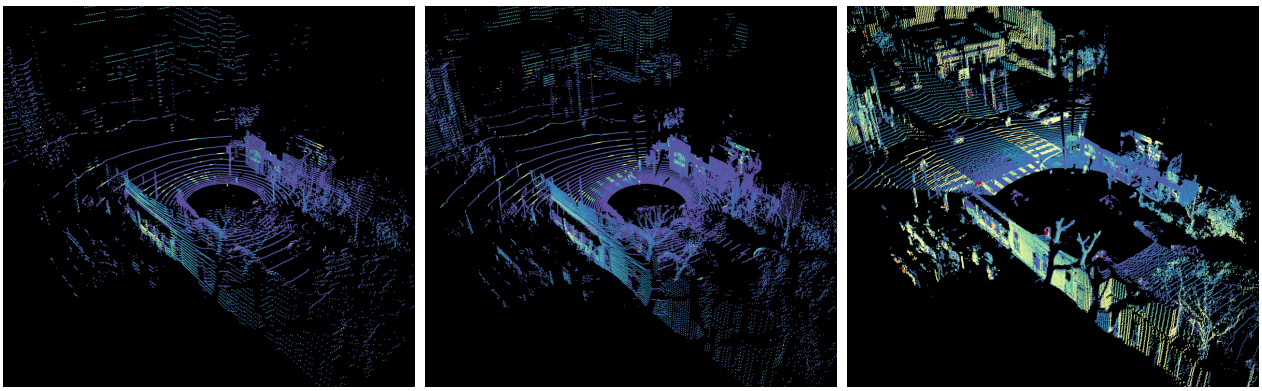


Figure 3.2: Sensor frame for Ouster with 32 channels (left), 64 (middle) and 128 (right).

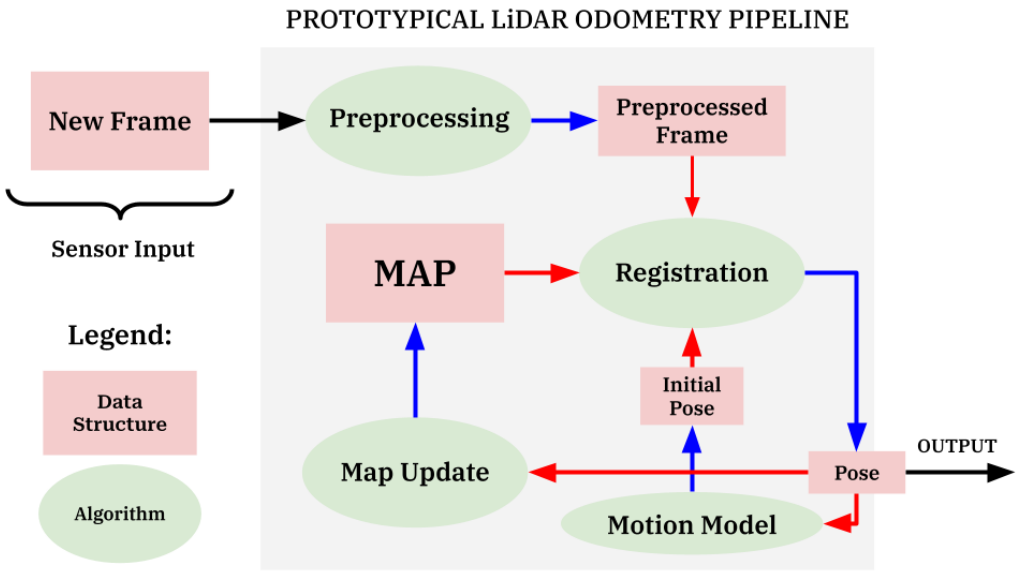


Figure 3.3: A prototypical simplified LiDAR odometry pipeline.

3.2.1 Registration and Maps

Map representation and registration methods are intrinsically linked. As mentioned above, and at the high-level most methods follow a procedure similar to the ICP [5]. More specifically, for each scan, a pose is estimated by iteratively refining an initial pose estimate. For this refinement, an optimal step is computed by minimizing a cost function, which is typically the sum of some type of distance between a sample of points or features from the new scan, and the map which stores the relevant information extracted from the registered scans to efficiently build these cost functions. After each step in this iterative procedure, the data association between points of the scan and the map is updated with the new estimate, before a new optimal step is computed or convergence criteria are reached.

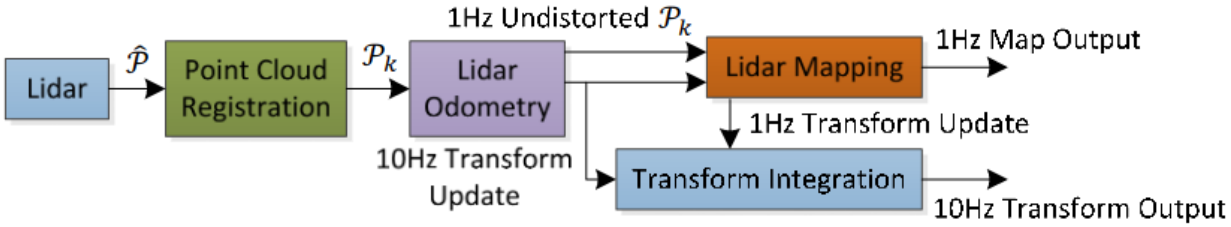
For performance and precision, many different strategies have been proposed and we detail in the following sections the principal groups of methods. On a side note, though we cover **LIO** methods in chapter 5, we mention some of them here, but with a focus on their registration procedure.

3.2.1.1 Geometric Feature Based

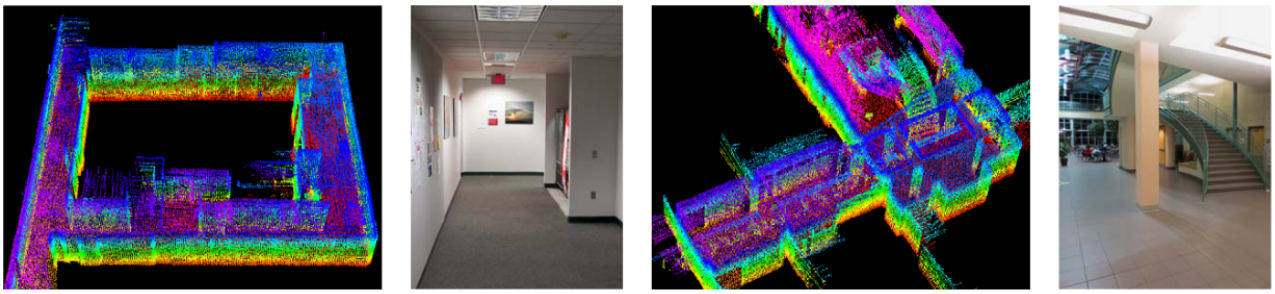
LOAM and Derived work While not the first 3D LiDAR SLAM, LOAM [135] (see figure 3.4) has certainly been the most influential **LO** method in the robotics community. The method processes each scan, and extracts feature points from individual scan lines of a rotating LiDAR sensor. These feature points correspond to either planar or edge features. Their method is built in two steps. First, a frame-to-frame alignment produces poses in real-time at 10Hz. Then, a slower mapping phase produces a more precise pose estimate using a frame-to-map alignment at 1Hz.

For both the frame-to-frame alignment and the frame-to-model, the registration procedure minimizes iteratively an optimization cost consisting of the sum of point-to-line and point-to-plane residuals:

$$E_{LOAM}(\mathbf{T}) = \sum_{k \in \mathcal{P}} \delta^2_{plane}(\mathbf{T} * \mathbf{p}^{\mathbf{pl}}_{\mathbf{k}}, \mathbf{n}^{\mathbf{pl}}_{\mathbf{k}}) + \sum_{l \in \mathcal{E}} \delta^2_{edge}(\mathbf{T} * \mathbf{p}^{\mathbf{e}}_{\mathbf{l}}, \mathbf{n}^{\mathbf{e}}_{\mathbf{l}}) \tag{3.1}$$



(a) Block diagram describing the LOAM method.



(b) Reconstruction of environments using the LOAM method.

Figure 3.4: Images taken from the LOAM paper [135], describing the method.

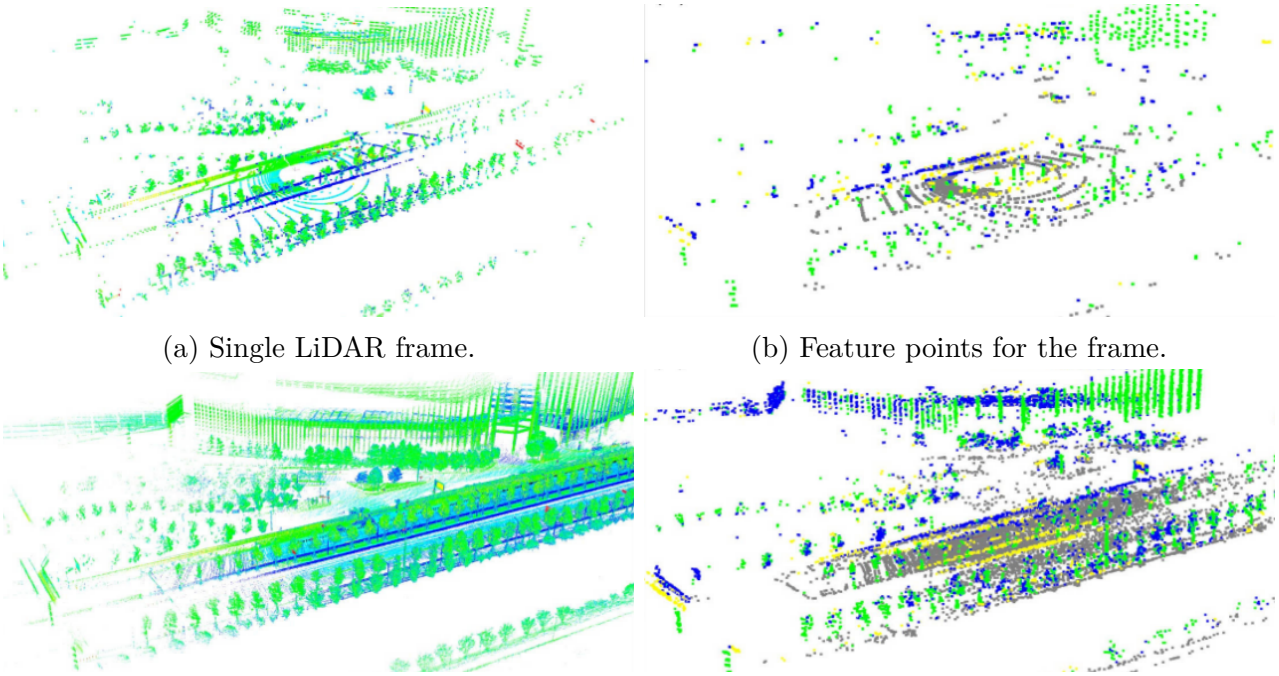
Where $\mathbf{p_k^-}$ and $\mathbf{n_k^-}$ designate respectively feature points from the new scan and feature points from the previous scan or the map. The features of a registered scan are stored in the map which consists of a dual voxel grid of edge and planar features. The data association for the "mapping" step is performed by constructing for each scan a kd-tree for all the feature points in the neighborhood of the new scan's pose estimate. One of the important reasons for the success of LOAM is its accuracy (measured in drift). It was indeed one of the first methods to outperform significantly other camera-based systems, notably on the most popular odometry benchmark of the time: the KITTI benchmark [40].

Since its publication, many works have extended and improved the original method. Notably, Lego-LOAM [111] proposes an additional ground segmentation using an image-based method [6] on the range image. A ground plane alignment is then used to estimate half of the pose parameters (Z, θ_{roll} and θ_{pitch}), and the LOAM keypoints alignment then estimates the remaining parameters (θ_{yaw}, X, Y). Other works also integrate a similar idea to incorporate ground constraints such as [71, 63, 133] based on the LOAM algorithm.

V-LOAM [134] uses a camera to estimate the frame-to-frame odometry and estimate poses at a high frequency (30Hz), and then uses them to distort the LiDAR frame and improve the pose estimate before the mapping stage.

Since the feature extraction of LOAM was designed for rotating LiDAR sensors with regular firing patterns, some methods have adapted the algorithm to work for other sensor modalities, notably for Livox sensors [69, 64] which have non-repetitive scanning patterns. Many more derived works have been proposed over the years [102, 47, 123, 16], which keep the main components of LOAM and its specific feature extraction strategy.

Other feature-based method Some approaches have proposed different feature extraction strategies. For example, MULLS [84] (see figure 3.5) extracts features of the following classes: facade, pillar, beam and vertex (MULLS is designed for urban scenarios, which contain many such structures). Their ICP variant solves at each iteration, and after a data association within feature classes, a multi-metric linear least square objective function, with appropriate costs for



(a) Single LiDAR frame.

(b) Feature points for the frame.

(c) Aggregated point cloud of local map.

(d) Map of feature points.

Figure 3.5: Images describing the MULLS [84] map and feature points. Images are from the original article [84]. The feature points are colored following: (ground, facade, pillars, beams)

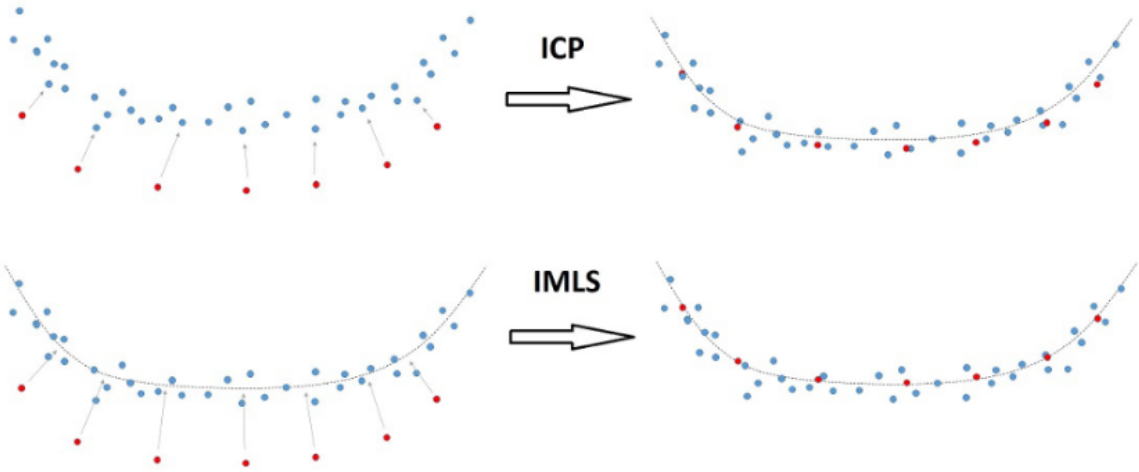
each class of feature.

Other approaches extract planes from the point cloud as features [52, 139, 138, 41]. Planes play an important role in point cloud processing, especially for indoor environments [7, 14]. These methods represent the map as a set of parametrized planes, and will typically update the poses but also the plane parameters as new scans are registered. Planes extracted from a new scan, typically with a range-image-based segmentation, are then tracked within the map [138, 139] and a plane-to-plane or point-to-plane is used for an initial pose estimate. One of the neat features of the infinite plane parametrization (compared with surfel elements or planar features for example), is that it makes a batch update for poses and planes more convenient. So most of these methods present some aspect of Bundle-Adjustment, which we cover in more detail in section 3.2.1.4.

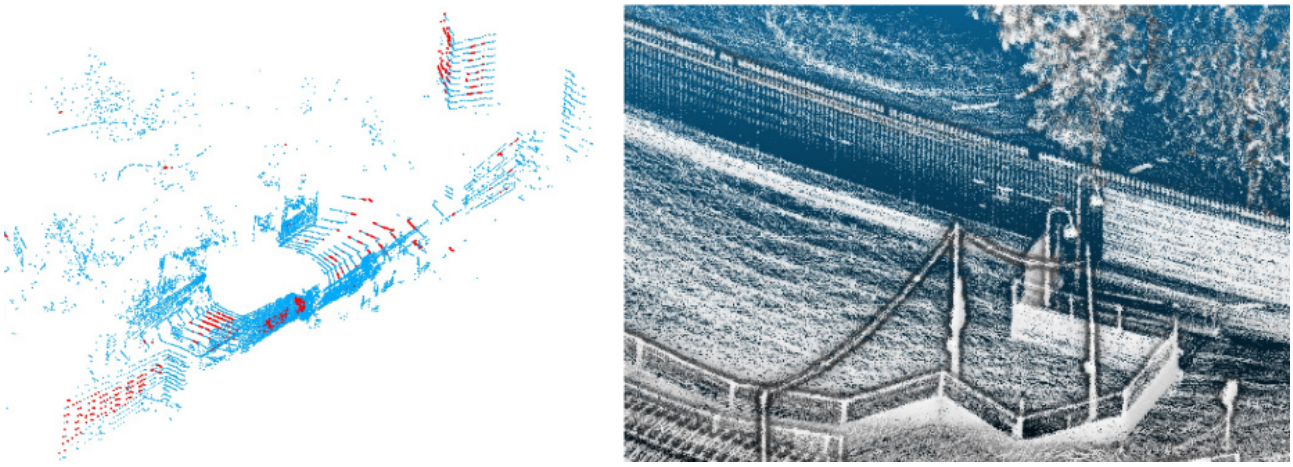
3.2.1.2 Dense Point Cloud Based

While feature extraction certainly has been relevant, they lead to a loss of information from the raw point cloud. Other approaches have been proposed and shown to obtain high levels of precision by keeping a dense point cloud representation of the map [26, 121, 127, 125, 126, 124]. This dense point cloud is obtained from iteratively inserting consecutive scans transformed with the estimated pose. The challenge of these approaches is to deal efficiently with the sheer quantity of data returned by the sensor, for which acceleration strategies are needed to reach the desired runtime.

IMLS-SLAM First to demonstrate the possible precision gains obtained with this approach, *IMLS-SLAM* [26] showed that retaining dense point cloud information in the map allows outperforming LOAM on the KITTI benchmark [40]. IMLS-SLAM implements a frame-to-model approach, where the model is a dense point cloud, consisting of a sliding window of consecutive frames. For each frame, a kd-tree is built on the window of aggregated frames, and the normals



(a) IMLS registration compared to the standard ICP. IMLS's registration iterates until the key points are aligned with the implicit surface and not the closest neighbors like the ICP.



(b) **Left:** sampled key points (red) from a single frame (blue), used to register the frame. **Right:** aggregated point cloud using IMLS's trajectory, shows the precision of the method, through the precision of the small objects.

Figure 3.6: Description of IMLS [26]'s method.

are computed. Then, a sample of points is selected and registered against the local map by minimizing the distance to the implicit surface of the point cloud. Figure 3.6 presents the sample points as well as a schematic comparison of the IMLS's registration procedure, compared to the standard ICP. This method, when it was published, reached state-of-the-art results on the KITTI odometry benchmark [40]. However, it is too slow for real-time application of SLAM (it runs at 1Hz), the main bottleneck being the construction of the kd-tree and the estimation of the normal.

Other methods Some **LIO** methods have also proposed dense point cloud map representation while keeping real-time performances [127, 125, 126]. Using the optimized search structure *ikd-Tree* [12], an iterative kd-tree, they store a dense point cloud, and can query for each key point neighbors in the map at a high frequency for their registration algorithm, without being delayed by map updates. We will cover more precisely their registration method, which is intrinsically linked to their LiDAR-Inertial framework in chapter 5.

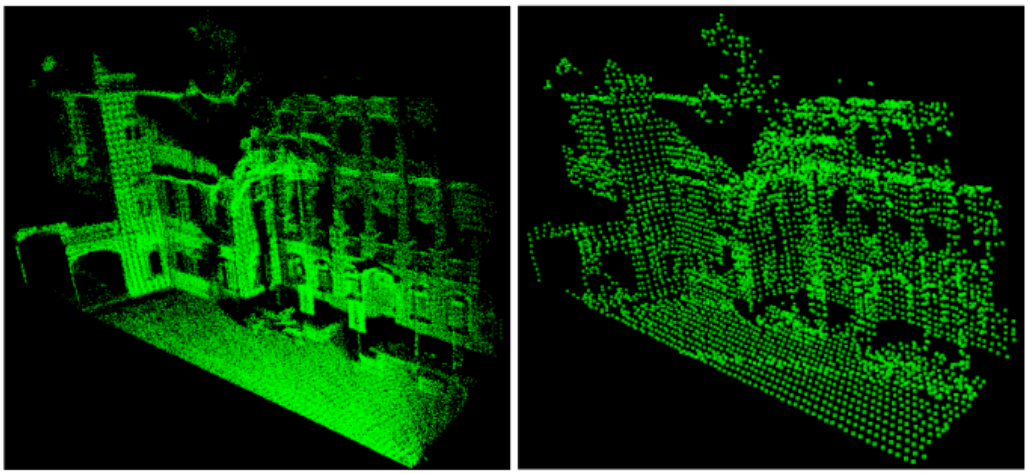


Figure 3.7: Dense point cloud (left), point cloud sampled (middle) Images are taken from ikd-tree [12].

3.2.1.3 Probabilistic distribution/Surfel variants

While dense point cloud approaches have tried to obtain improved accuracy by keeping more information within the map, other approaches have taken a different road and decided to lower the quantity of data to handle while maintaining information by modelling the surface of the environment with probabilistic distributions [39, 58, 56, 131, 132, 83, 99] or surfel elements [8, 11, 32, 87, 86, 31, 30, 95, 4, 17].

Surfel-based representation Though linked through the representation of the environment as a set of surfaces, surfel elements differ from an infinite plane representation [52] presented in section 3.2.1.1. Surfels are indeed small and localized surface elements as seen in figure 3.9 and model a small region of the environment for a much lesser cost than the set of individual points they were constructed from. Surfel maps are often built from and stored in voxel grids [8, 11, 32, 87, 86, 31, 30, 95]. A surfel is typically constructed from the covariance of a cluster of points and thus summarizes the information for this cluster. SuMA [4] stands out by using projective data association to compute neighborhoods as well as point clusters. It stores surfels on the GPU and uses rasterization to construct a projective-based point association between surfel elements from the map and the new scan.

Depending on the method, each scan can either be summarized as a surfel map [95], or directly registered with a point cloud to surfel map alignment. In the first case, a surfel map to surfel map alignment is performed. While the other is obtained by minimizing a point-to-plane distance between points and their associated surfel elements [4, 17, 86].

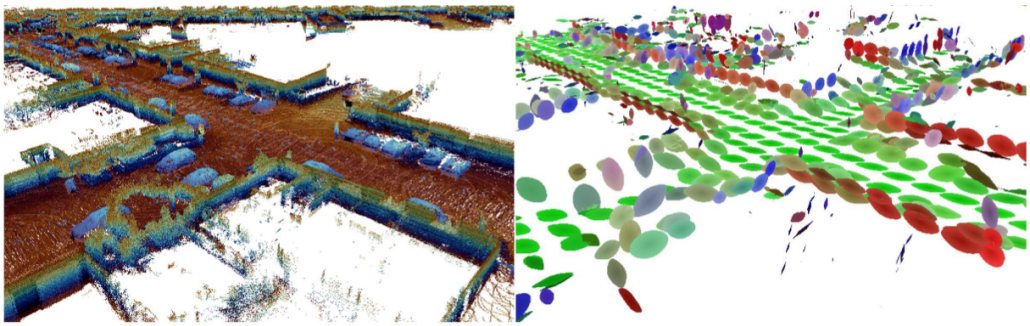


Figure 3.8: The Litamin map (on the right), summarizes the dense point cloud (on the left) as a voxel grid of distributions. Image taken from the original article [132].

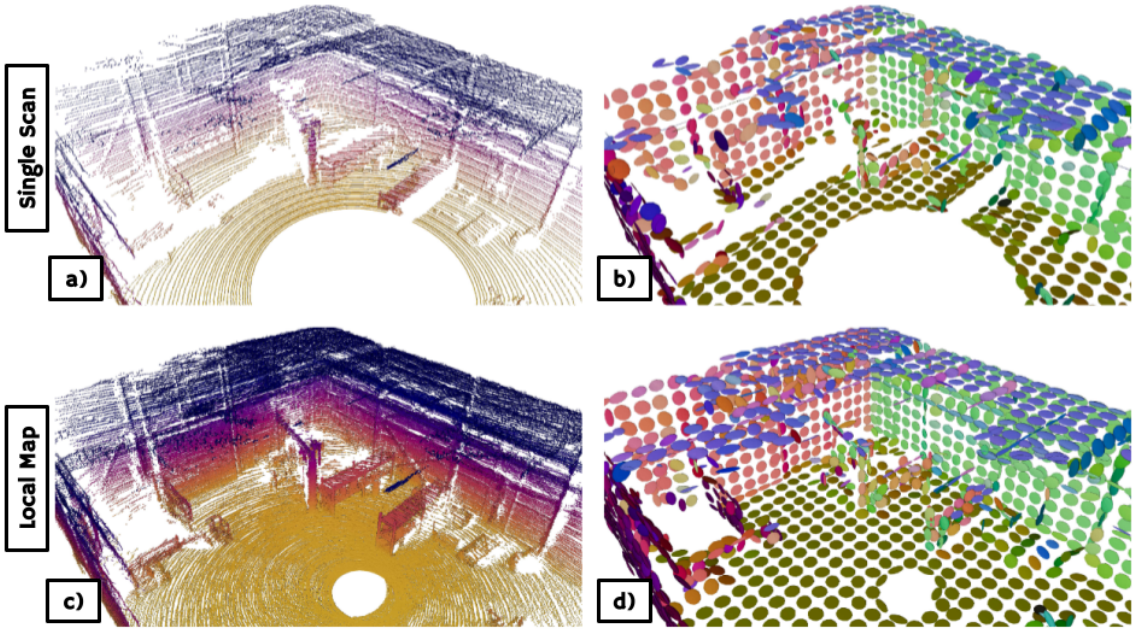


Figure 3.9: Surfel map (b) representation from a single scan (a), and surfel map (d) from a dense map (c) generated and stored within a voxel grid by [95]. All pictures are from the original article [95].

Distributions: point to distribution, distribution to distribution Another strategy to summarize the surface information is the probabilistic approach. Instead of flattening the covariance like surfels, the information within a region of a map or scan is summarized as a probabilistic distribution [39, 58, 56, 131, 132, 83, 99]. A first approach uses a normal distribution transform (NDT) [76] as a scan registration strategy. The NDT approach splits the space using voxels and stores within each voxel the mean (μ) and covariance (Σ) of the corresponding point cluster, and models with a normal distribution $p(\mathbf{x}) \sim \mathcal{N}(\mu, \Sigma)$ the distribution of points within the voxel. This map ($\mathcal{M}$) is then used to register a scan ($\mathcal{S}$) using a point-to-distribution approach. Data association is performed using the voxel grid, and iteratively a pose estimate is refined by maximizing the log-likelihood of the points belonging to the selected voxels:

$$\underset{\mathbf{T}\in SE(3)}{\operatorname{argmax}} \log \mathcal{P}(\mathcal{S}, \mathbf{T}|\mathcal{M}) = \underset{\mathbf{T}\in SE(3)}{\operatorname{argmax}} \; -\sum_i (\mathbf{T} * \mathbf{p}_i - \mu_i)^T \Sigma_i^{-1} (\mathbf{T} * \mathbf{p}_i - \mu_i) \tag{3.2}$$

This 3D NDT scan registration is directly used by multiple methods to construct LiDAR Odometries [56, 39, 16].

Another popular approach is based on the Generalized-ICP [108] registration method. This approach generalizes both the original ICP [5], which minimizes a point-to-point distance between points, and its point-to-plane extension. In its original formulation, each measurement both from the point cloud to register and the reference point cloud is modeled as sampled from a normal distribution. So, for a point cloud $\mathcal{PC}^A$ to be registered on a point cloud $\mathcal{PC}^B$, the Generalized ICP refines poses by solving the following objective:

$$\underset{\mathbf{T}\in SE(3)}{\operatorname{argmax}} \mathcal{P}(\mathcal{PC}^A, \mathbf{T}|\mathcal{PC}^B) = \underset{\mathbf{T}\in SE(3)}{\operatorname{argmax}} \log \mathcal{P}(\mathcal{PC}^A, \mathbf{T}|\mathcal{PC}^B) \tag{3.3}$$

$$= \underset{\mathbf{T}\in SE(3)}{\operatorname{argmin}} \; \sum_i (\mathbf{T} * \mu_i^A - \mu_i^B)^T (\Sigma^B + \mathbf{T}^T \Sigma^A \mathbf{T})^{-1} (\mathbf{T} * \mu_i^A - \mu_i^B) \tag{3.4}$$

The Generalized ICP is a framework to register point clouds. Though in their original paper, they present an enhanced plane-to-plane method, with a specific noise model of the covariance, other models of the covariance could be used. Implemented within the Point Cloud Library (PCL) [103], and facilitated by the library's ROS integration, the Generalized ICP has been the basis of multiple **LO** and **LIO** odometries [83, 99, 110, 100].

Recently [58] proposed a voxelized variant of GICP. This algorithm essentially combines the NDT and GICP algorithms for faster execution. GICP relies on the costly nearest neighbor association to determine each point covariance, while the NDT summarizes per voxel the distribution. In their approach, VGICP combines the two. A preprocessing step extract for each scan of the covariance for each point, using an efficient and parallel implementation of a kd-tree. Then, the average covariance of each point is saved within each voxel (which ensures that even for voxels with only one point, the covariance is not degenerate). This essentially allows keeping more information from the raw point cloud, without the cost of an expensive nearest neighbor search for each scan registration.

This is particularly helpful in the context of **LO**, as this leads to easier map updates within the map after a scan is registered. Some very efficient **LO** have been built on top of this registration [131, 132]. Figure 3.8 shows a representation of the LiTAMIN map as a grid of distributions.

3.2.1.4 Local Bundle Adjustment (BA)

As we saw in previous sections, there is a large variety of registration methods and map representation, each with its specificities, strengths and weaknesses, and will study some of them in section 3.5. Most odometry methods presented above have however one thing in common after a scan is registered, the corresponding geometry of the map is never modified by the odometry module, but only after loop closure, typically by modifying the map after pose graph updates. This means that registration errors or approximations are never updated in light of new measurements, which leads to the accumulation of drift which is typically never corrected before loop closure.

Some approaches have been proposed to address this problem by proposing methods inspired by Bundle Adjustment formulation of visual SLAM and structure from motion [72, 139, 49, 138, 35]. Compared to Pose Graph optimization, which optimizes a set of poses by minimizing constraints on relative poses, bundle adjustment approaches optimize poses by minimizing geometric constraints between poses and the environment. This allows them to refine poses while staying faithful to the geometry, which a pose-graph only allows through covariance estimations. Figure 3.10 shows an example of a LiDAR Bundle Adjustment formulation from the article [72].

BALM [72], built on a LOAM framework, optimizes on a sliding window the poses for 20 scans, by minimizing edge and planar constraints between features observed from multiple scans. Other approaches have focused on planar features [138, 35], and from a set of planar constraints optimize a window of poses by maximizing the planarity of those planes. Very recently, [28] proposes a unified LiDAR and RGBD Bundle Adjustment, which performs on par or better than current state-of-the-art SLAM and BA approaches.

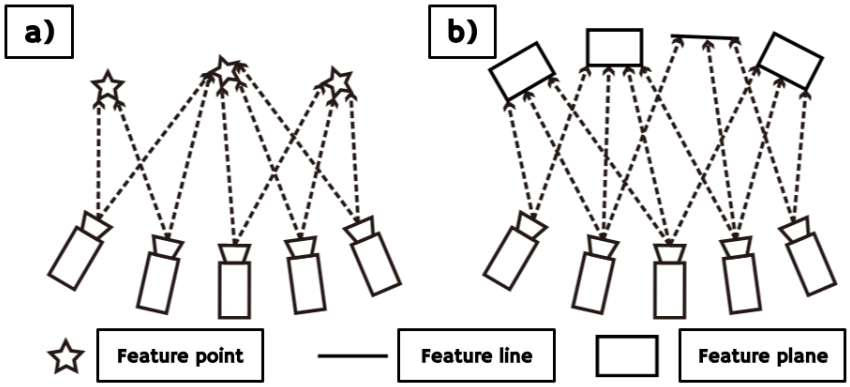


Figure 3.10: Visual Bundle Adjustment vs LiDAR Bundle Adjustment Formulation proposed by BALM [72]. Image is taken from [72] original paper.

3.2.1.5 Conclusion

Though we tried to present a detailed picture of the principal LiDAR odometries approaches, we have not been completely exhaustive. Other interesting approaches have presented more complex map representation using volumetric representation with a large scale TSDF [60], an implicit representation [48], or using a mesh representation for the map [120]. And many more variations exist within the presented categories. We refer to [50] for a more detailed survey of existing LiDAR odometries.

In section 3.5 we selected a few of the publically available methods to construct benchmarks on the wide range of datasets presented in chapter 2.

3.2.2 Trajectory Representation/Motion Model

After the pair registration method/map representation, another important aspect of LiDAR Odometries is their trajectory representation and motion model. The motion of the vehicle or robot carrying the LiDAR sensor is continuous, yet it is parametrized by a discrete set of parameters. Furthermore, the LiDAR stream is often split into individual point cloud frames (this is typically done at the driver level, for all LiDAR sensors), and the task for SLAM and **LO** approaches is to output poses for the frame at the same frequency than the frames.

Though this is true for all types of sensors, including for cameras, this is particularly significant for LiDAR due to the slow acquisition time of these sensors. Indeed, most LiDARs produce scans at a frequency of 10 to 20 Hz. During this time the sensor is typically moving, and possibly at a high speed by comparison to this frequency. This creates a distortion of the LiDAR measurements, also called the "rolling shutter effect" (referencing cameras' rolling shutter), which needs to be compensated for optimal registration and odometry quality. While some approaches simply ignore this problem [131, 132], aided notably by the KITTI benchmark which produces motion compensated frames [40], there is a large body of work focusing on this problem [81, 8, 11, 30].

We can separate the problem into three components. First, the motion model, which we present in section 3.2.2.1 describes how the motion is initialized before registration. This is a very important, yet often overlooked aspect, as ICP-based registration typically has very small convergence radiuses, and easily falls into local minimums. Secondly, comes the question of how the trajectory is represented, modeled and parameterized (see figure 3.11). We present a review of the approaches in section 3.2.2.2. Finally, using both the motion and the trajectory representation, we present the different strategies for handling the distortion of the LiDAR frame in section 3.2.2.3.

3.2.2.1 Motion Model/Initialization

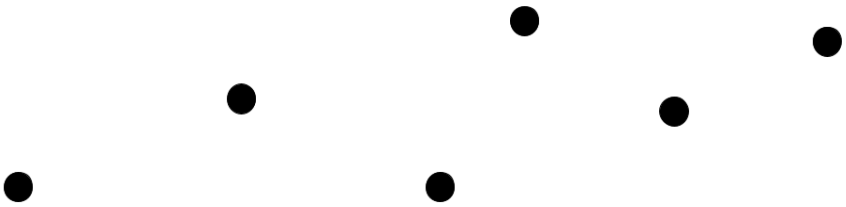
As mentioned above, ICP-based methods, which rely on point association between two clouds, need good initializations not to fall in local minimums. Strategies to predict the motion of future frames based on the past registered trajectory is often called the motion model for the platform. A motion model can also be used to define additional constraints on the registration procedure, to force coherence with the model [25, 111].

Most **LO** methods rely on the constant velocity (CV) model [135, 25, 24, 123, 121, 8, 93, 139, 49]. This simple model simply considers the velocity as fixed between consecutive frames, and thus initializes the pose $\mathbf{T}_{i+1}^{init}$ for a new frame by applying the previously estimated relative pose $\mathbf{T}_{i,i+1}^{*}$ to the pose of the previous scan $\mathbf{T}_{i}^{*}$ as follows:

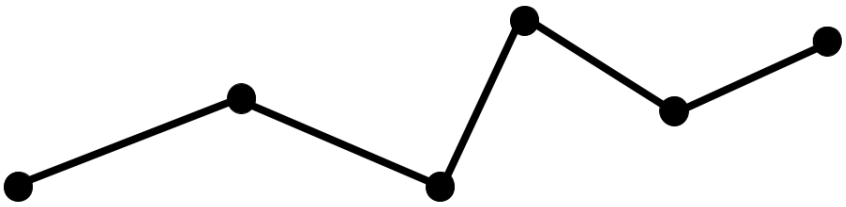
$$\mathbf{T}_{i+1}^{init} = \mathbf{T}_{i}^{*} * \mathbf{T}_{i,i+1}^{*} \tag{3.5}$$

The constant velocity model is particularly relevant in the context of road vehicles, which have high inertia, and high-speed but slow angular acceleration. Though it is still relevant, as shown recently in some challenging mobile robotics contexts [25, 121], when the motion has fast acceleration, notably high angular acceleration (notably for handheld datasets [44, 136]).

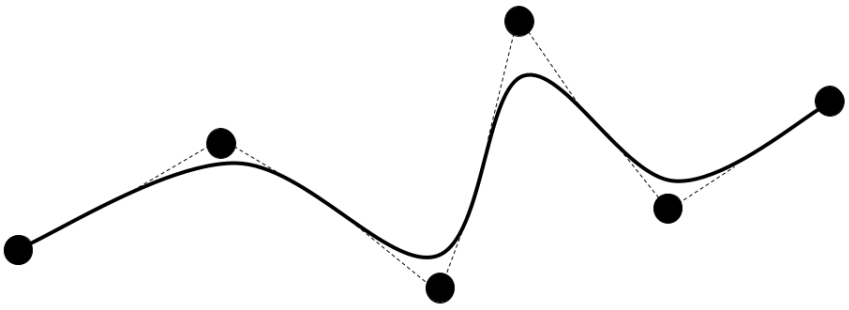
For these scenarios, an Inertial Measurement Unit (IMU) is often considered necessary. As presented in chapter 2, IMU sensors measure linear acceleration and angular velocity, typically at a much higher frequency than LiDAR frames. This information is integrated by **LIO** methods to predict the motion, and put constraints on the optimization procedure, and it typically removes the need for a motion model altogether. **LIO** is the focus of chapter 5, so we do not detail further these methods here.



(a) Rigid poses trajectory model.



(b) Constant velocity trajectory model, implemented by linear interpolation between rigid poses.



(c) B-Spline continuous trajectory model, parametrized by discretized knots.

Figure 3.11: Representation of the different trajectory models for robotics application. Black dots represent parameter poses and the lines, the estimated trajectory.

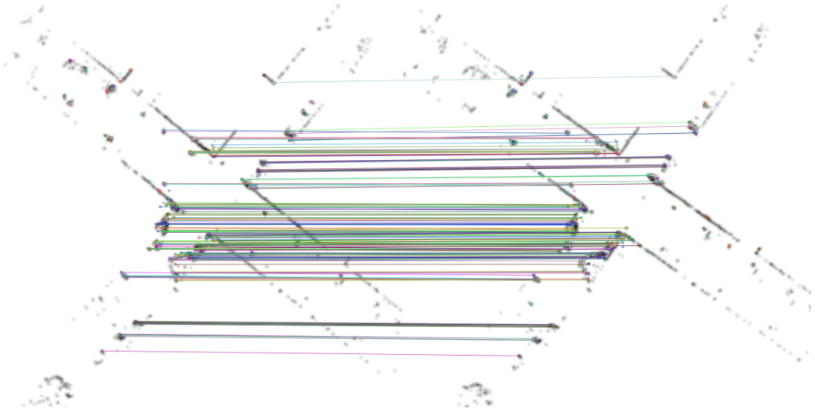


Figure 3.12: Elevation-Image registration based, motion initialization method presented in [24]. The image represents two consecutive scans projected on the z-axis aligned using a geometrically verified 2D image based alignment.

Other approaches have been proposed, instead of, or completing the constant velocity model. Notably, LOAM [135] initializes the motion with a frame-to-frame alignment before its precise registration procedure (called "mapping" in the original article). This approach has been extended by [16], which proposes an NDT-based frame-to-frame registration before calling LOAM's scan-to-map registration.

In our work, [24], where we study the performance of different initialization strategies, we propose a novel Elevation-Image based alignment strategy. This strategy estimates the 2D motion of the sensor, by projecting consecutive scans on the z-axis, and using an image matching technique to estimate the 2D rotation (in *yaw*) and the 2D motion (see figure 3.12). We present this work in more detail in section 3.3.1.

On a similar note, LeGO-LOAM [111] uses a ground detection to constrain the motion. As noted above, their registration procedure approach estimates the 6D motion in two steps. First estimating the z-axis and *roll* and *pitch* angles of the rotation using a ground alignment procedure, before estimating the remaining parameters with the standard LOAM optimization.

3.2.2.2 Representation/Parametrization of the trajectory

Most methods model the trajectory as a set of poses, where one pose corresponds exactly to the pose of the sensor at the time of a single scan acquisition ([135, 84, 24, 111, 123, 26, 83]...). This approach is convenient in the context of SLAM, as it integrates easily within the Pose Graph framework. If poses are needed at higher rate, notably to distort the LiDAR frame [123], they are typically sampled at a higher frequency with a linear interpolation between the reference frames. More specifically, given a pose $\mathbf{T}^A = (\mathbf{R}^A, \mathbf{t}^A)$ at a time t_A and a pose $\mathbf{T}^B = (\mathbf{R}^B, \mathbf{t}^B)$ at a time t_B, a linear interpolation allows to estimate a pose at a time $t \in [t_A, t_B]$ with:

$$t_\alpha = \frac{t - t_A}{t_B - t_A} \in [0, 1] \tag{3.6}$$

$$\mathbf{t}(t) = t_\alpha \cdot \mathbf{t}_B + (1 - t_\alpha) \cdot \mathbf{t}_A \tag{3.7}$$

$$\mathbf{R}(t) = \text{Slerp}(\text{Quat}(\mathbf{R}_A), \text{Quat}(\mathbf{R}_B), t_\alpha) \tag{3.8}$$

$$\mathbf{T}(t) = (\mathbf{R}(t), \mathbf{t}(t)) \tag{3.9}$$

We note here Slerp(.) the **S**pherical **L**inear Int**erp**olation operation on quaternions, and Quat(.) the operator which transforms a rotation to a representative unit quaternion.

B-spline trajectory representation While this representation has the advantage of simplicity, one thing to notice is that it is not continuously differentiable. Thus, it is not ideal to model smooth trajectories, which are often relevant, notably in driving scenarios or for mobile robotics. More complex modelization of trajectories is sometimes needed, notably when accurate trajectory positions are needed at a high frequency, for example when fusing data with higher frequency sensors like IMU or cameras. This has motivated a large body of work to propose alternate and more complex representations, allowing more precise pose estimations at high sampling rates. A popular solution is to use B-splines [8, 11, 32, 106, 30, 95, 97] to model the trajectory, and build a continuous and differentiable trajectory representation.

A general B-spline (or **basis spline**) of order n, is a piecewise polynomial function of degree $n-1$ in a variable t, which serves as a basis function to define spline functions. Given $m+1$ knots $t_0, t_1, ..., t_{m+1}$, a B-spline of order n is defined (recursively and by construction) as:

$$B_{i,1}(t) := \begin{cases} 1 & \text{if } t_i \leq t < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

$$B_{i,n}(t) := \frac{t - t_i}{t_{i+n} - t_i} B_{i,n-1}(t) + \frac{t_{i+n+1} - t}{t_{i+n+1} - t_{i+1}} B_{i+1,n-1} \tag{3.11}$$

These basis functions are used to define more complex interpolation, notably to define smooth continuous curves. Given $m+1$ control points $\mathbf{P_0}, \mathbf{P_1}, ..., \mathbf{P_m} \in \mathbb{R}^d$ (for each of the $m+1$ knots), the spline of order n supported by the B-spline and these control points is defined as:

$$S^n(t) = \sum_{i=0}^{m-n} B_{i,n}(t) \cdot \mathbf{P_i}, \text{ defined for } t \in [t_n, t_{m-n}] \tag{3.12}$$

Because a B-spline of degree n is a polynomial of degree $n-2$, it belongs to the class C^{n-2}. Thus to obtain a twice continuously differentiable function, an order of 4 is necessary. This definition is not directly applicable to model trajectories due to the non-additive nature of $SO(3)$. To resolve this, the rotation and translation are often decoupled [113, 92, 42], and the translation component can be expressed using equation 3.12. The rotation component is typically expressed using cumulative B-spline curve (CuBsp) [113, 92, 42]. The idea stems from a reformulation of equation 3.12 using cumulative basis functions:

$$S^n(t) = \mathbf{P_0} + \sum_{i=1}^{m} \widetilde{B}_{i,n}(t)(\mathbf{P_i} - \mathbf{P_{i-1}}) \tag{3.13}$$

$$\widetilde{B}_{i,n}(t) = \sum_{j=i}^{m} B_{j,n}(t) \tag{3.14}$$

This equation can be applied to $SO(3)$ and its associated Lie algebra. For $\mathbf{X_0}, \mathbf{X_1}, ..., \mathbf{X_m} \in SO(3)^{m+1}$, and given the logarithm $\log : SO(3) \to \mathbb{R}^3$ and exponential map $\exp : \mathbb{R}^3 \to SO(3)$, [55] proposed the following formula to define a curve in SO(3) using the cumulative B-spline functions:

$$S^n(t) = \mathbf{X_0} \prod_{i=1}^{m} \exp[\widetilde{B}_{i,n}(t) \log(\mathbf{X_{i-1}}^{-1} * \mathbf{X_i})] \tag{3.15}$$

This approach was optimized by [113, 92], and we refer the reader to these approaches for more details on the method. Figure 3.13 shows an illustration of the usage of B-Spline by CLins [75], a LiDAR-Inertial Odometry method.

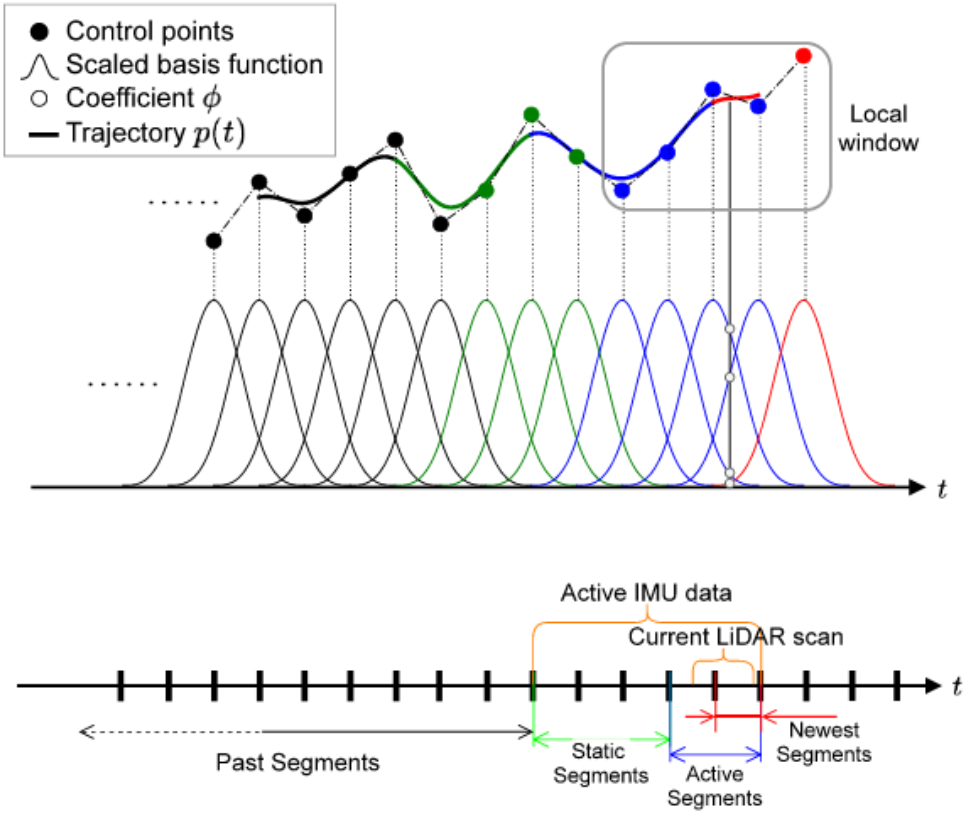


Figure 3.13: CLins [75] uses a B-Spline to represent a continuous trajectory. They optimize Knots parameters on a sliding window. This continuous modeling, notably allows them to construct IMU residuals using the derivatives of the trajectory. Images from [75].

3.2.2.3 Handling the distortion of the LiDAR

The final aspect of the trajectory representation is how the distortion of the point cloud is handled. As we will show in section 3.5, the choice of strategy to handle this motion distortion has a strong effect on the precision of odometries. The principal reason is that the motion distortion can create bad point associations, which leads to convergence towards a bad objective by ICP methods.

Despite its importance, this singular problem has often only been proposed as a small feature within a more global system and rarely studied in itself. Many methods, indeed, notably those essentially evaluated on KITTI which provides motion corrected frames[135, 84, 39, 57], do not consider this problem.

Typically, there are currently four approaches. The first approach is to treat this as a preprocessing problem. Given an initial estimation of the motion, using a constant velocity model or if available the prediction from an IMU, the point cloud is undistorted using this estimate, and then registered as is in the map [139, 112]. Other approaches use a two-step method, where after an initial undistortion is applied and after the scan is registered, the distortion is updated using the novel, refined estimate [123, 49]. While this method corrects the distortion given a better estimate, the registration itself uses the same initial distortion, even after the pose is updated at the end of an iteration, before a new data association is performed. To go one step further, the distortion can be corrected at the end of each iteration of the registration procedure before the data association is updated [93].

All of the above approaches use linear interpolation to perform the distortion of the point cloud (though, if an IMU is available, the poses can be sampled at higher frequency [97]). However, the registration remains a rigid transformation, and the distortion is not modeled in the optimization objective. So the final step is to model the continuous trajectory with an extended set of parameters and optimize the objective on these parameters. This is the approach taken by many methods modeling the trajectory with B-splines [11, 32, 106, 30]. These methods typically estimate trajectory segments corresponding to multiple scans with a small set of control points, which tends to smooth the trajectory [96]. In our work [25], which we present in section 3.4, we propose an alternative distortion method, notably inspired by [9].

3.2.3 Acceleration strategies

For many robotics applications, SLAM algorithms need to be fast. As the computational resources often need to be shared by multiple processes for multiple tasks (path planning, object recognition, obstacle avoidance, etc...), there is also an incentive for low memory, power and computation consumption. This is typically a challenge, notably due to the immense quantity of data typically received from the sensors. For reference, the Velodyne Alpha Prime with its 128 channels, produces around 4.8 million points per second, which amounts to at least 54Mo per second (using 32bit floating point representation).

Using all this information should lead to more precise algorithms, however, in practice, to construct real-time SLAM solutions fed with this data some acceleration strategies are required. One key aspect of the different strategies we present in this section is to choose an advantageous tradeoff between on the one hand the loss of information and the loss of precision associated with it and on the other the performance gains. Counterintuitively perhaps, this tradeoff is not always straightforward, as often the precision is not always associated with the quantity of information available, but rather with selecting the useful information.

In this section, we present the most important strategies to gain performance in LiDAR SLAM. First, reducing the amount of data in the LiDAR frame is necessary, the challenge is to adopt strategies losing the least amount of information for a performance requirement. We present these sampling and summarization techniques in section 3.2.3.1. Another typically costly operation is the point association. In the context of an ICP-based registration, point associations are updated after each inner iteration of the algorithm. Thus to accelerate the nearest neighbor search, some data structures are necessary, and we present some work focusing on this aspect in section 3.2.3.2.

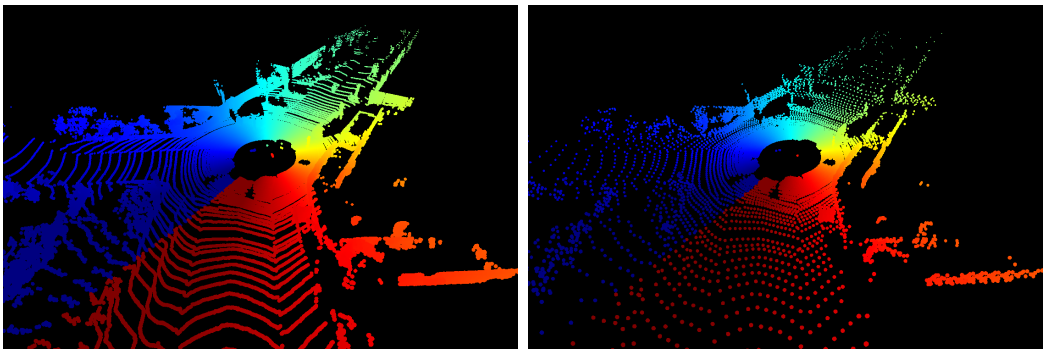
3.2.3.1 Sampling and Summarizing

We already mentioned multiple summarization strategies in section 3.2.1, some of which are intrinsically linked with the associated registration method (e.g. surfel extraction, features extraction LOAM [135], voxelization NDT [76]). Summarizing and sampling can occur at two stages, first as a general pre-processing step, to lower the number of points to process, and then during the registration procedure, by selecting a smaller amount of key points to register.

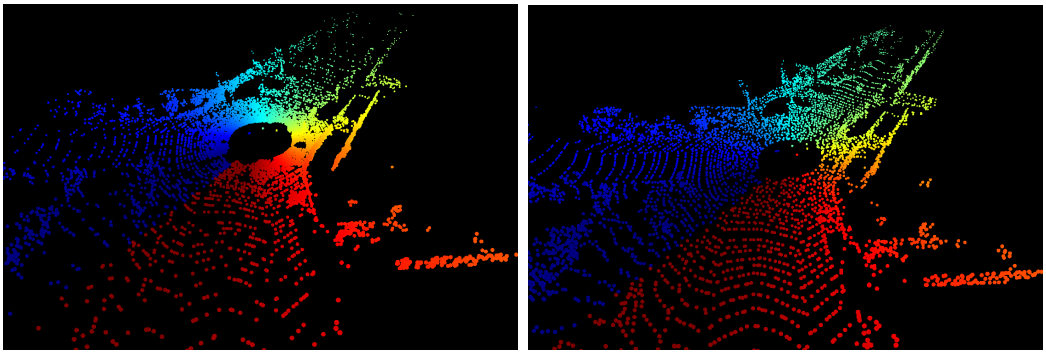
Features Sampling We already mention LOAM-based methods in 3.2.1. One of the goals of this dual feature extraction is to reduce the computational load. This preprocessing step selects points in the input scan which have either high (planar points) or low (edge points) local smoothness. In the original paper [135], points are grouped by scan lines, we note $\mathbf{p}^{(m,n)}$ the n^{th} point of the m^{th} scan line. The local smoothness is computed as follows, where $\mathcal{N}^{(m,n)}$ denotes the surrounding neighbor points of $\mathbf{p}^{(m,n)}$:

$$\sigma^{(m,n)} = \frac{1}{\#\mathcal{N}^{(m,n)}} \sum_{\mathbf{p}^{(m,j)} \in \mathcal{N}^{(m,n)}} (||\mathbf{p}^{(m,j)} - \mathbf{p}^{(m,n)}||_2^2) \tag{3.16}$$

Two thresholds σ_{edge} and σ_{planar} control the number of edge and planar features extracted. The performance gains from this sampling occur mainly from limiting the size of the optimization problem to solve (measured by the number of residuals to evaluate) and of the neighbor association step in the ICP.



(a) Original frame (b) Regular sampling of 10^4 points.



(c) Random sampling of 10^4 points. (d) Grid sampling with voxel of 0.5m

Figure 3.14: Geometric sampling for different strategies.

Geometric Sampling Outside of LOAM-based methods, notably for dense point cloud alignment methods (see 3.2.1.2), the most popular strategy is to sample points to have a satisfying distribution of points in space. The goal of some strategies is to correct the resolution gap between the vertical and horizontal axes. Rotating 3D LiDARs sweep the environment at high frequency, using N^{laser} (with N^{laser} ranging from 16 to 128 depending on the sensor). So for each frame, they produce a point cloud organized as a 2D tensor of dimension $N^{laser} \times N^{fire}$, where N^{fire} typically equals 1024 or 2048. Thus, a strategy is to reduce the number of the horizontal dimension of the laser, this can either be done by projecting the laser in a smaller range image [4, 24], or by subsampling columns from the tensor image.

While this approach has many advantages, including simplicity, speed, and the capacity to be parallelized (notably on the GPU [4, 24, 15]), it does not adapt to the geometry captured. A typical problem is that it can suppress points far away while retaining many points belonging to surfaces close to the sensor. So a final possibility is to sample points with a fixed size voxel grid [121, 25, 49]. For each frame, a single point is kept per voxel, so the quantity of information retained depends on the selected voxel size. Figure 3.14 shows different sampling strategies of LiDAR Frames implemented in *CT-ICP*. One of the problems of this approach is that the number of points depends on the environment. In large and open environments, this leads to a significantly larger sample, and thus slower processing time (both for the registration and map management). We investigate this in section 3.5.

Finally, a simpler method is to sample points randomly with a uniform distribution. This approach is notably used in LOCUS [83] which keeps only around 10% of the points of incoming frames.

Summarizing A last alternative we already mentioned above is to summarize each frame using a voxelized distribution. This is the approach selected by [131, 132], where each frame is voxelized, and the mean and covariance of points falling in the voxel are stored for each voxel. Similarly, surfel methods can transform each frame into a surfel map, by computing the surfel parameters from the mean and covariance of points attached to each surfel (for example using a voxel grid) [95].

3.2.3.2 Neighbor association and search data structure

A costly step, in terms of runtime and computations, is the neighbor association. In an ICP-based scheme, each point association is recomputed at each iteration, typically by finding its nearest neighbor in the map. This typically requires using spatial query acceleration schemes, to avoid iterating through all the points of the map.

Kd-Tree, Octree The most popular approach is to use space partition data structures, such as kd-trees or octrees. Both of these approaches partition the space, though differently. A kd-tree [38, 81] is a balanced tree that splits the space at the median for a given axis, iteratively permuting axes for each node. The construction has a complexity of $O(n\log(n))$ (in the 3D case), and the expected complexity of nearest-neighbor queries is in $O(\log(n))$ [38].

By contrast, an octree [33] splits the space into regular grids, and depending on the point's repartition in space, is often not balanced. Traversing the tree requires $O(K)$ operation, where K is the number of subdivisions. Most methods relying on octrees typically set a minimum resolution (0.001m for LOCUS 1.0 [83], and from 0.1m to 0.001m for LOCUS 2.0 [99]). Figure 3.15 illustrates the different division of space of the two methods

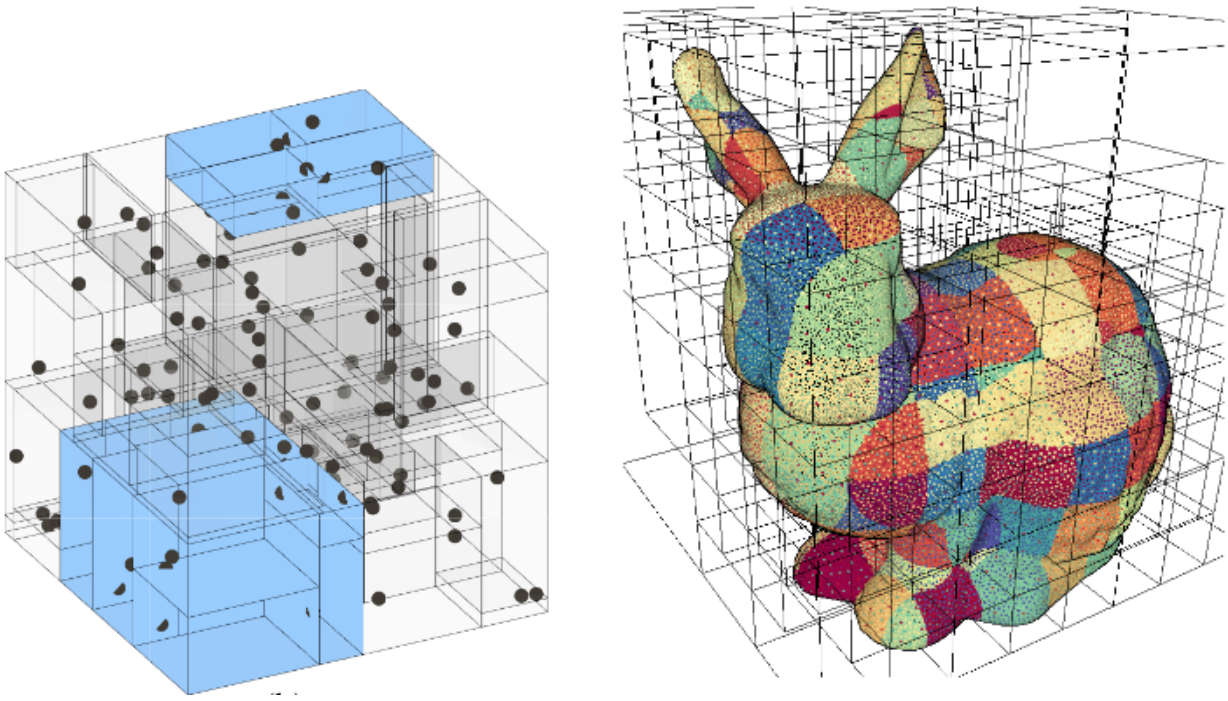


Figure 3.15: Separation of space for a Kd-Tree (left) and an Octree (right). Both tree data structures separate the space to accelerate spatial queries. But use different methods to do so: the octree splits the space equally at each node, while a kdtree computes the optimal separating plane to tidy points. Images from [12] and [107].

The issue with these two data structures is that they are not designed for a dynamic 3D environment such as the ones created by SLAM algorithms. To be efficient, SLAM algorithms need to perform map management operations such as removing old data or duplicated data or updating the structure after a trajectory update. To get around this problem, multiple strategies were employed, some more or less efficient.

LOAM and derived work [135, 123], store the point cloud map in a voxel grid. When a scan is registered against the map, a set of features in the surrounding environment of the scan is extracted, and a kd-tree is built specifically for these features. It is then used by the registration

algorithm and destroyed when it finishes. Similarly, IMLS-SLAM [26] iteratively constructs a new kd-tree for the previous $N = 50$ frames for each scan registration, which negatively impacts the runtime in this case.

Recently, ikd-Tree [12] proposed an iterative kd-Tree which augments a static kd-Tree with thread-safe management operations (insertion, deletion, downsampling) in $O(\log(n))$. The data structure management operates on a separate thread, and tree rebalancing is triggered each time an update operation is required. Following this, LOCUS 2.0 [99] proposed a multi-threaded octree and achieves similar performances as [12]. The system maintains an octree centered around the robot, with regular insertions. And periodically, the construction of a novel octree is triggered and computed on a separate thread. Once the computation terminates, the two octrees are simply swapped at the end of the construction.

Projective association One of the problems of tree structures, is the limit in their parallelization capacities for a single query, as tree traversal cannot be parallelized. By contrast, projections are very parallelizable, and thus can efficiently be exploited by GPU for parallel processing. This is the strategy adopted by SuMA [4, 17] to produce an odometry that processes high-resolution scans (with a resolution of 900×64) in real-time. The point association for their point-to-plane approach is performed by pixel-wise association within a spherical image. More precisely, the surfel map is projected in the same spherical image as the input cloud. Points of the new scan are associated with surfel falling in the same pixel for a point-to-plane alignment. Another approach, [109], also uses a projective data association within a registration procedure inspired by GICP[108] and NDT[76]. In our work [24] we also exploit this projective data association, and present it in section 3.3.

Voxel Hashing Another increasingly popular acceleration strategy is to use voxel hashing [78, 126, 121]. The general idea is to store voxels in a hash table, and for the neighbor association, to find neighboring occupied voxels using hash comparisons in the table. We detail this approach in section 3.4, when we present our work [25] which incorporates this data structure.

3.2.4 Conclusion

In this section, we presented in detail the most important aspects of a LiDAR-only odometry pipeline. As we saw, there is a large variety of approaches for each step of the pipeline, and though most new methods make some effort to isolate the proposed contributions of each new component through ablation studies, it is still difficult to extrapolate statements outside of the specific LiDAR Odometry pipeline presented. For example it is impossible to say which registration method is fundamentally the best based on the current related work alone, there is no consensus on which sampling strategy is better, which method to handle the distortion works best, etc... In section 3.5, we aim to help answer some of these questions by performing a deep and detailed analysis both between systems and within our LiDAR Odometry system, through a large variety of benchmarks.

But before this, we present the two LiDAR Odometry pipelines we developed during this thesis in section 3.3 and 3.4.

3.3 `pyLiDAR-SLAM`: A basic classical pipeline

In our work [24], we performed a detailed comparison of multiple LiDAR Odometry approaches, focusing on the difference between deep and classical LiDAR odometries, which we present in section 4.3. This comparison is made possible by the implementation of a classical pipeline, which serves as a reference to derive our analysis. In this section, we detail this classical pipeline, which was designed to be a simple implementation of point-to-plane LiDAR odometry. Yet, despite its simplicity, this method nonetheless obtains near state-of-the-art results on the KITTI odometry benchmark.

This work is integrated within the modular python toolbox `pyLiDAR-SLAM`. This toolbox is designed by breaking the LiDAR Odometry into different modules, each module having multiple implementations, including deep learning blocks, which can be combined into different pipelines. Figure 3.16 presents an overview of the modular pipeline, and the different modules implemented. In green are all the classical modules we address in this section. The toolbox is publicly available on GitHub at `https://github.com/Kitware/pyLiDAR-SLAM/wiki/SLAM-LiDAR-Toolbox`.

So, in the remaining of this section, we present this work, which will serve as a good baseline when we later introduce our main work CT-ICP in section 3.4.

Detailed Contributions The contributions of the work presented in this section are the following:

- The implementation of a modular point-to-plane odometry package, which reaches near state-of-the-art performance on KITTI despite its simplicity.
- The use of elevation images to initialize the 2D motion of the sensor.
- A novel projective frame-to-model odometry, though assembled from existing components.
- A lazy computation of the map normals for better performance.

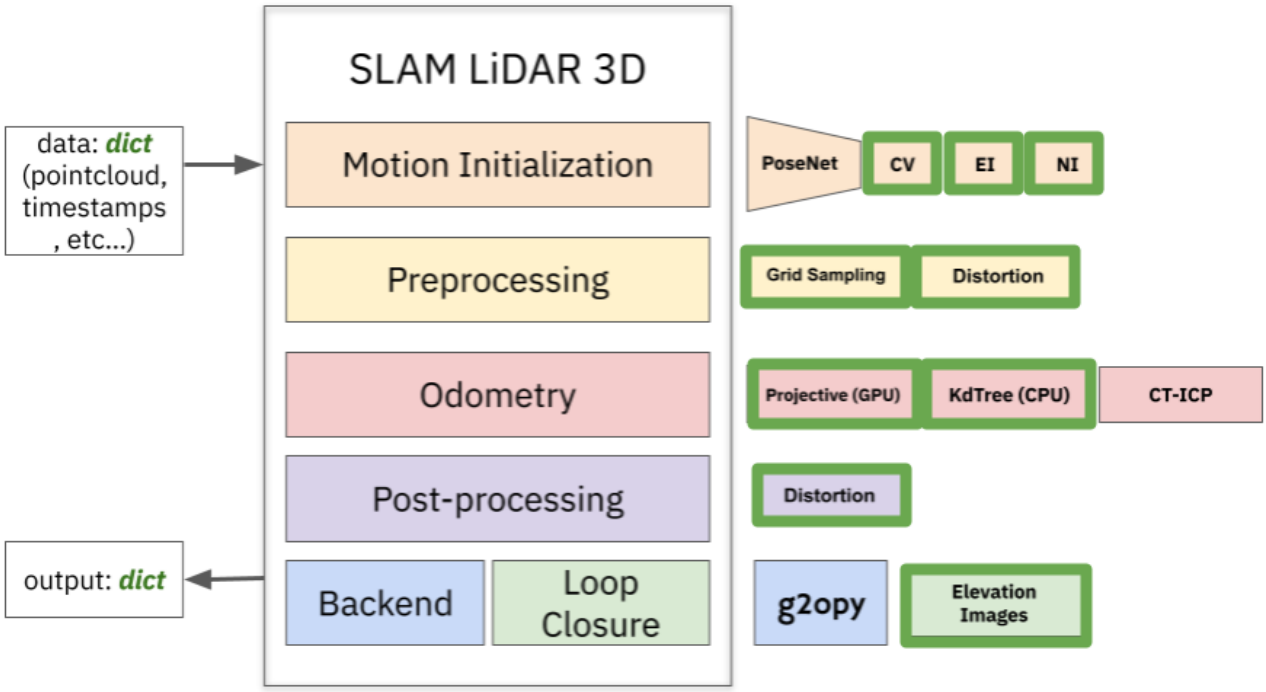


Figure 3.16: Overview of the `pyLiDAR-SLAM` python package. The toolbox is designed as a modular pipeline, where each module has multiple implementations, which allows testing different combinations of modules.

3.3.1 Motion Initialization and Preprocessing

First, before a frame is given to the odometry module, a preprocessing step is applied and an initial pose is estimated. The importance of each of these two steps was already mentioned in depth in section 3.2.

Motion Initialization (*NI,CV,EI*) Additionally to the basic common approaches of using a Constant Velocity model (**CV**), or no initialization (**NI**), we introduce a novel Elevation Image (**EI**) based initialization procedure. Elevation images have already been used notably in loop closure context [74], and overall we will see in section 3.5 that it does not yield significative improvements over other methods. Using elevation-based alignment in this context is nonetheless, to the best of our knowledge, a new approach that achieves similar goals as the two steps alignments of Lego-LOAM [111] presented above.

For each new frame, we project into a 2D occupancy grid along the z-axis (this projection assumes that the sensor is oriented vertically with respect to the ground, which is typically the case for a sensor on a vehicle for instance). The grid has a resolution of 800×800, where each pixel represents a 30cm $\times$ 30cm patch of the ground for driving scenarios (though this parameter should be adapted in other contexts). The grid is then treated as an image, ORB [101] features are extracted (we chose ORB features for their efficiency and rotation invariance). To estimate the motion of the new frame, we perform a 2D image alignment between the new frame and the previous frame. Using the RANSAC algorithm [36], we fit an image homography, and if a threshold number of inliers is reached, the motion is initialized with the 2D transform computed from the homography. See figure 3.12 for an illustration of the elevation image registration process. This method, within the `pyLiDAR-SLAM` framework slightly outperforms other models **CV, NI** as we will show in section 3.5.

Preprocessing (*Grid Sampling, Distortion*) As a preprocessing step, we apply either a grid sampling step or a spherical image based sampling (which were already presented in the above section 3.2.3.1), followed by a distortion procedure. By default, we proposed in our work [24] a voxel size of 0.4m for the grid sampling, and a resolution of 64×720 for the spherical sampling. However, in section 3.5 we present the effect of each of these parameters on the performance of the algorithm.

3.3.2 Registration procedure

After the motion initialization and the frame new frame is preprocessed, we register it within the map using a point-to-plane variant of the ICP algorithm [5]. A prototypical ICP algorithm is presented in Algorithm 1. The ICP is an iterative procedure that computes incremental pose updates, by minimizing the distance between the transformed point cloud and a reference point cloud. The distance in question depends on the variant, as was seen above in section 3.2.1. In this work, we used the point-to-plane [88] variant and detail it below.

Another crucial point of the registration is the type of point association used. In our case, for the point-to-plane variant, we need to compute at each iteration the nearest neighbor and its normal in the point cloud. We present in section 3.3.3 the strategies considered in this work.

Point-to-plane Error As presented in algorithm 1, incremental pose updates are obtained by minimizing the distance between key points of the preprocessed frame, and the point cloud of the map. In this work, we estimate increments by minimizing the distance between a key point and the tangent plane of the environment supporting its closest neighbor. Keeping the same notations as Algorithm 1, where $\mathcal{PC}_1$ is the target point cloud, and $\mathcal{PC}_2$ the map point cloud, and $\mathbf{n}_i$ the normal of the tangent plane passing through $\mathbf{q}_i$, the error minimized is defined

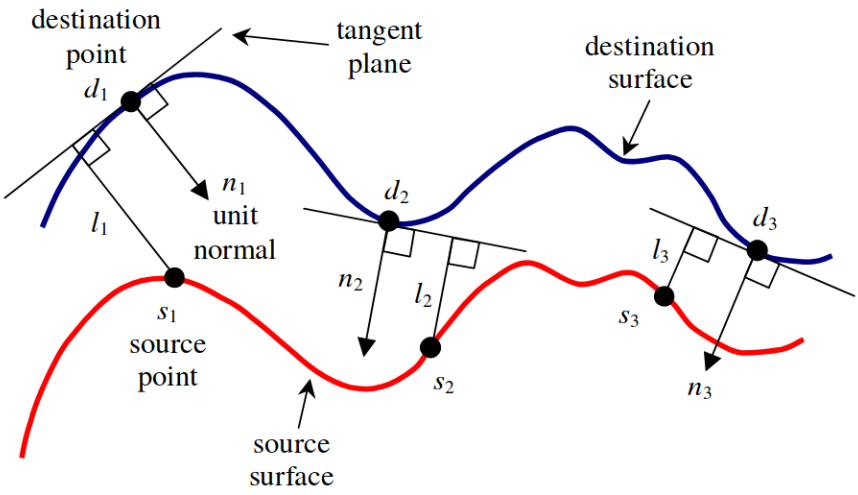


Figure 3.17: Point-to-plane error between two surfaces. Image is from the technical report [73].

as the sum of point-to-plane residuals for the set of appariements $\mathcal{N}_{1,2} := \{(p_i, q_i)\}$. This error, which we note $E_{\text{pt-to-pl}}(\mathbf{T})$, is expressed as follows:

$$E_{\text{pt-to-pl}}(\mathbf{T} \in SE(3)) = \frac{1}{2} \sum_{i=1}^{\#\mathcal{NN}_{1,2}} \rho((|\mathbf{T} * \mathbf{p}_i - \mathbf{q}_i) \cdot \mathbf{n}_i|^2) = \frac{1}{2} \sum_{i=1}^{\#\mathcal{NN}_{1,2}} \rho(r_i^2) \tag{3.17}$$

The point-to-plane is often preferrable to the original point-to-point distance [4, 17, 78, 26]. The intuition behind this is that a point-to-plane metric allows the scan to slide on the dominant planes, which leads to larger convergence areas (*ie* larger areas in the parameter space which leads to correct convergence).

Robust Estimator In the equation 3.17, ρ denotes a robust loss function that is used to mitigate the effect of outliers. Outliers occur when points association fail to associate a key point to a point (or a plane) which will bring it closer to its real position on the map. This occurs for multiple reasons, relatively large initial motion, mobile objects, or simply due to the specific geometry of the environment. Robust estimators will cancel the quadratic contributions in a standard least-square scheme of wrong associations which have residual growth as the solution gets closer to the real value.

We can solve equation 3.17 with a least square solver, by introducing the weights $w_i = \sqrt{\rho'(r_i^2)}$. Then, minimizing 3.17 is, with some approximations, equivalent to minimizing (which we show below):

$$E^{WLS}_{\text{pt-to-pl}}(\mathbf{T}) = \frac{1}{2} \sum_{i=1}^{\#\mathcal{NN}_{1,2}} w_i^2 \cdot r_i(\mathbf{T})^2 \tag{3.18}$$

This least-square expression is non-linear due to the rotation component of the pose parameters. Thus to minimize it, an iterative least-square solver is required. At each step, of the iteration, an optimal step $\delta\mathbf{T}^*$ is computed, and then residuals and weights are updated. While many popular robust estimators exist, in this work we used the Cauchy estimator $\rho_{cauchy}(r^2, \sigma) = \sigma^2 \log(1 + \frac{r^2}{\sigma^2})$, parameterized by the distance parameter σ. For this estimator, $\rho'_{cauchy}(r^2, \sigma) = \frac{1}{1+r^2/\sigma^2}$. This estimator, which is asymptotically logarithmic allows us to

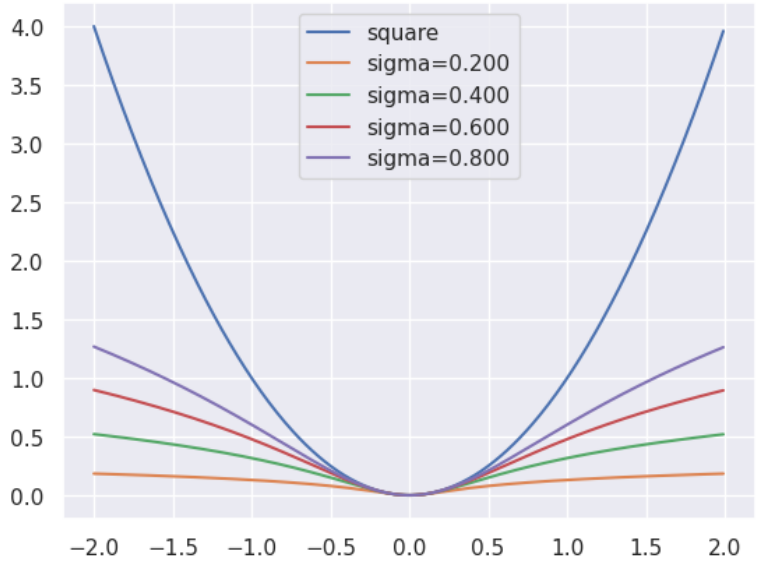


Figure 3.18: Cauchy loss function for different values of σ, compared to the function square (without robust estimators). For residuals far from zero (where the "far" is controlled by σ), the error contribution to the objective is logarithmic.

strongly eliminate large outliers without clamping, and thus we found, leads to more refined estimates. Figure 3.18 shows the Cauchy loss function for different values of σ.

Gauss-Newton In this work, we use the Gauss-Newton algorithm to minimize the expression above. First, we show that solving 3.17 and 3.18 is equivalent. From the taylor expansion of equation 3.17, we have:

$$E_{\text{pt-to-pl}}(\delta\mathbf{T} * \mathbf{T}) = E_{\text{pt-to-pl}}(\mathbf{T}) + J(\mathbf{T}) * \delta\mathbf{T} + \frac{1}{2} \cdot \delta\mathbf{T}^T * H_E(\mathbf{T}) * \delta\mathbf{T} + o(\|\delta\mathbf{T}\|^2) \tag{3.19}$$

Where $J(\mathbf{T}) \in \mathbb{R}^6$ and $H_E(\mathbf{T}) \in \mathbb{R}^{6\times6}$ are respectively the jacobian and hessian of $E_{\text{pt-to-pl}}$ with respect to $\mathbf{T}$. Minimizing this expression can be done by setting the derivatives to zero, which leads to:

$$J(\mathbf{T}) + H(\mathbf{T}) * \delta\mathbf{T}^* = 0 \tag{3.20}$$

$$\delta\mathbf{T}^* = -H_E(\mathbf{T})^{-1} * J(\mathbf{T}) \tag{3.21}$$

This is the expression of the Newton algorithm, of which the Gauss-Newton is an approximation. For equation 3.17, the jacobian and hessian are expressed as:

$$J(\mathbf{T}) = \frac{1}{2}\sum_k \frac{\partial\rho(r_k^2)}{\partial\mathbf{T}} = \frac{1}{2}\sum_k 2r_k \cdot \rho'(r_k^2) \cdot \frac{\partial r_k}{\partial\mathbf{T}} = \sum_k r_k \cdot \rho'(r_k^2) \cdot \frac{\partial r_k}{\partial\mathbf{T}} \tag{3.22}$$

$$H_E(\mathbf{T}) = \frac{1}{2}\left(\sum_k \frac{\partial^2\rho(r_k^2)}{\partial\omega_i\partial\omega_j}\right)_{ij} = \frac{1}{2}\sum_k \left(\frac{\partial^2\rho(r_k^2)}{\partial\omega_i\partial\omega_j}\right)_{ij} \tag{3.23}$$

$$= \sum_k \left(\rho'(r_k^2) \cdot \frac{\partial r_k}{\partial\omega_i}\frac{\partial r_k}{\omega_j} + \rho'(r_k^2) \cdot r_k \cdot \frac{\partial^2 r_k}{\partial\omega_i\partial\omega_j} + 2\rho''(r_k^2) \cdot r_k^2 \cdot \frac{\partial r_k}{\partial\omega_i}\frac{\partial r_k}{\omega_j}\right)_{ij} \tag{3.24}$$

For $E_{\text{pt-to-pl}}^{WLS}$ of equation 3.18, we apply the standard Gauss-Newton derivation from the Taylor expansion, starting from the lifted form of the error expressed using residual vectors:

$$E^{WLS}_{\text{pt-to-pl}}(\mathbf{T}) = \frac{1}{2} \cdot R(\mathbf{T})^T * R(\mathbf{T}), \text{ where } R(\mathbf{T}) = \Big(w_i \cdot r_i\Big)_i \tag{3.25}$$

$$\begin{aligned} E^{WLS}_{\text{pt-to-pl}}(\delta\mathbf{T} * \mathbf{T}) &= \frac{1}{2} \cdot R(\mathbf{T})^T * R(\mathbf{T}) + \cdot R(\mathbf{T})^T * J_R(\mathbf{T}) * \delta\mathbf{T} \\ &+ \frac{1}{2}(\delta\mathbf{T}^T * J_R(\mathbf{T})^T * J_R(\mathbf{T}) * \delta\mathbf{T} + 2 \cdot R(\mathbf{T})^T * H_R(\delta\mathbf{T}, \delta\mathbf{T})) + o(\|\delta\mathbf{T}\|^2) \end{aligned} \tag{3.26}$$

Where $H_R(\delta\mathbf{T}, \delta\mathbf{T})$ is the hessian of the residual vector $R(\mathbf{T})$, and $J_R(\mathbf{T})$ its jacobian. The Gauss-Newton method consists of approximating $H_{E^{WLS}}$ to $J_R(\mathbf{T})^T * J_R(\mathbf{T})$, then we optimal step for the minimization of E^{WLS} is:

$$\delta\mathbf{T}^{WLS} = -[J_R(\mathbf{T})^T * J_R(\mathbf{T})]^{-1} * [R(\mathbf{T})^T * J_R(\mathbf{T})] \tag{3.27}$$

$$\text{replacing: } J_R(\mathbf{T}) = \Big(w_i \cdot \tfrac{\partial r_i}{\partial \mathbf{T}}\Big)_i = \Big(\sqrt{\rho'(r_i)} \cdot \tfrac{\partial r_i}{\partial \mathbf{T}}\Big)_i, \text{ leads to:} \tag{3.28}$$

$$\delta\mathbf{T}^{WLS} = \Big(\sum_k \rho'(r_k) \tfrac{\partial r_k}{\partial \omega_i} \tfrac{\partial r_k}{\partial \omega_j}\Big)^{-1}_{ij} * [\Big(\sqrt{\rho'(r_k)} r_k\Big)^T_k * \Big(\sqrt{\rho'(r_k)} \tfrac{\partial r_k}{\partial \mathbf{T}}\Big)_k] \tag{3.29}$$

$$= -\Big(\sum_k \rho'(r_k) \tfrac{\partial r_k}{\partial \omega_i} \tfrac{\partial r_k}{\partial \omega_j}\Big)^{-1}_{ij} * [\sum_k \rho'(r_k) \cdot \frac{\partial r_k}{\partial \mathbf{T}}] \tag{3.30}$$

Looking more closely at (3.24) we note that if we make the following approximations (which are often made on the assumption of small residuals):

$$\rho'(r_k^2) \cdot r_k \cdot \frac{\partial^2 r_k}{\partial \omega_i \partial \omega_j} \approx 0 \tag{3.31}$$

$$2\rho''(r_k^2) \cdot r_k^2 \cdot \frac{\partial r_k}{\partial \omega_i} \frac{\partial r_k}{\omega_j} \approx 0 \tag{3.32}$$

Then we see that the optimal steps (3.27) and (3.21) are equal, which justifies our reformulation as a weighted least square problem.

Algorithm 1: Prototypical ICP Algorithm.

Input : $\mathcal{PC}_1$, $\mathcal{PC}_2, \mathbf{T}_{1\to2} \in SE(3)$
Output: $\mathbf{T} \in SE(3)$; `/* The optimized pose */`

$\mathbf{T} \leftarrow \mathbf{T}_{1\to2}$
Kpts $\leftarrow \mathbf{T} * \mathcal{PC}_1$
$k \leftarrow 0$
while $k \leq N_{max}$ *or Convergence Criterion Reached* **do**
  $\mathcal{NN}_{1,2} \leftarrow$ Nearest Neighbors of Kpts in $\mathcal{PC}_2$; `/* Neighbor Association */`
  $\mathcal{NN}_{1,2} = \{(\mathbf{p}_i \in \text{Kpts}, \mathbf{q}_i \in \mathcal{NN}_{1,2}), \forall i \in [0, \#\text{Kpts}]\}$
  $\delta\mathbf{T}^* = \underset{\delta\mathbf{T}\in SE(3)}{\text{argmin}} \sum_{i=1}^{\#\mathcal{NN}_{1,2}} Dist(\delta\mathbf{T} * \mathbf{p}_i, \mathbf{q}_i)$; `/* Pose Optimization */`
  $\mathbf{T} \leftarrow \delta\mathbf{T}^* * \mathbf{T}$; `/* Pose Update */`
  Kpts $\leftarrow \mathbf{T} * \mathcal{PC}_1$; `/* Keypoints Update */`
  $k \leftarrow k + 1$;
end

3.3.3 Point Association and Neighborhood construction

As mentioned above, a critical step of the registration procedure is the neighbor association method. As our work is a point-to-plane based association, and as seen in algorithm 1, for each iteration of the ICP, for each keypoint we need to compute the nearest neighbor and its associated normal in the map. This requires giving the map a point-association method, and in this work, we used two of them, which we compare against each other. First a classical kd-tree-based method (similar to [135, 26]), and then a spherical projection data association. We detail both methods here.

Kd-Tree In this work, and similarly to [26], at each iteration the map consists of the point cloud obtained by the concatenation of n consecutive frames, and a Kd-Tree is built on the concatenated point cloud. Then for a given query point, the exact nearest neighbor in the map is found by descending through the graph structure [38], which take $O(\log(N_{map}))$ where N_{map} is the number of points in the map. Additionally, for the point-to-plane association, we need to build the normal for each query point. For a given point of the map, the normal is estimated using its neighborhood's covariance. More precisely, for a point $\mathbf{p}_{map}$ of the map a neighborhood $\mathcal{N}(\mathbf{p}_{map})$ of its k nearest points in the map are selected, and the normal $\mathbf{n}_{map}$ is set as the eigenvector with smallest eigenvalue of the covariance matrix:

$$\mathbf{\Sigma}(\mathbf{p}_{map}) = \sum_{\mathbf{p}_k \in \mathcal{N}(\mathbf{p}_{map})} (\mathbf{p}_k - \mathbf{p}_{map})^T \cdot (\mathbf{p}_k - \mathbf{p}_{map}) \in \mathbb{R}^{3\times 3} \tag{3.33}$$

$$\mathbf{n}_{map} = \underset{\mathbf{v}\in\mathbb{R}^3, \|\mathbf{v}\|_2=1}{\operatorname{argmin}} \|\mathbf{\Sigma}(\mathbf{p}_{map}) \cdot \mathbf{v}\|_2 = \mathbf{e}_3, \quad \text{where} \tag{3.34}$$

$$\mathbf{\Sigma}(\mathbf{p}_{map}) \cdot \mathbf{e}_3 = \lambda_3 \mathbf{e}_3, \text{ and } 0 \leq \lambda_3 \leq \lambda_2 \leq \lambda_1 \text{ are the eigen values of } \mathbf{\Sigma}(\mathbf{p}_{map}) \tag{3.35}$$

In IMLS-SLAM [26] this operation is precomputed for every point of the map, at each insertion in the map, and is a particularly costly operation responsible for the long execution time of the algorithm. However, due to the coarseness of the sampling of IMLS-SLAM, most neighbors in the map are not even considered during the registration procedure, which results in a lot of wasted computation time. By contrast, in this work, we use a lazy approach. We compute normals during the registration procedure, for the nearest neighbor selected, and store this normal in the map. For the following iterations of the ICP, the normal computation is only performed for the queried nearest neighbors which have not yet been computed. This simple lazy approach considerably improves the run-time compared to the naive strategy used in [26], as we show in section 3.5.

Projective Another approach is to use a projective data association. The general idea of this approach is to project the 3D point cloud of both the map and the target into an image and associate target points and map points by pixel association. This idea is similar to SuMa [4], though they projected a surfel map into an image instead of a point cloud. The image generated, which stores in each voxel the x, y, z coordinates is called a vertex map ($\mathcal{VM}$) and is constructed using a spherical projection $\mathcal{SP}$, as follows (also described in figure 3.20):

$$\mathcal{SP} : \mathbf{p} = (x, y, z)^T \in \mathbb{R}^3 \setminus \{0\} \rightarrow \mathbf{px}^n(\mathbf{p}) \in [0, 1]^2, \text{where}$$

$$FoV = |FoV_{down}| + |FoV_{up}| \tag{3.36}$$

$$r = \|p\|_2,\ \theta = -\arctan(y/x),\ \phi = \arcsin(z/r) \tag{3.37}$$

$$\mathbf{px}^n_y = 0.5 * (\theta/\pi + 1.0), \quad \mathbf{px}^n_x = 1.0 - \frac{(\phi + FoV_{down})}{FoV} \tag{3.38}$$

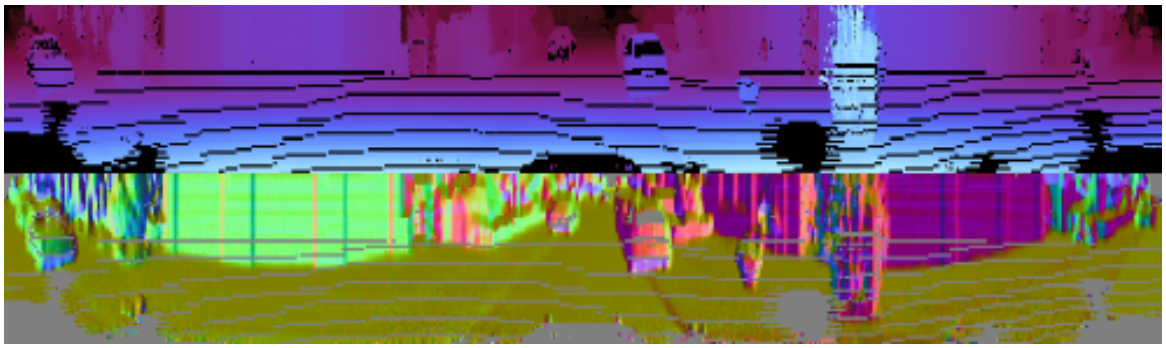


Figure 3.19: Vertex Map (top) and Normal Map (bottom) computed from a single scan of KITTI's dataset (top).

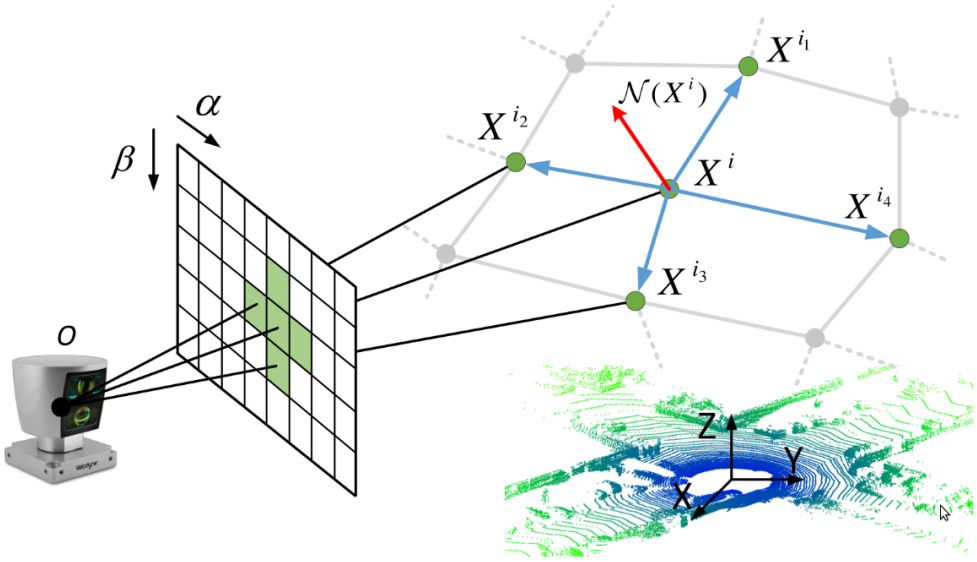
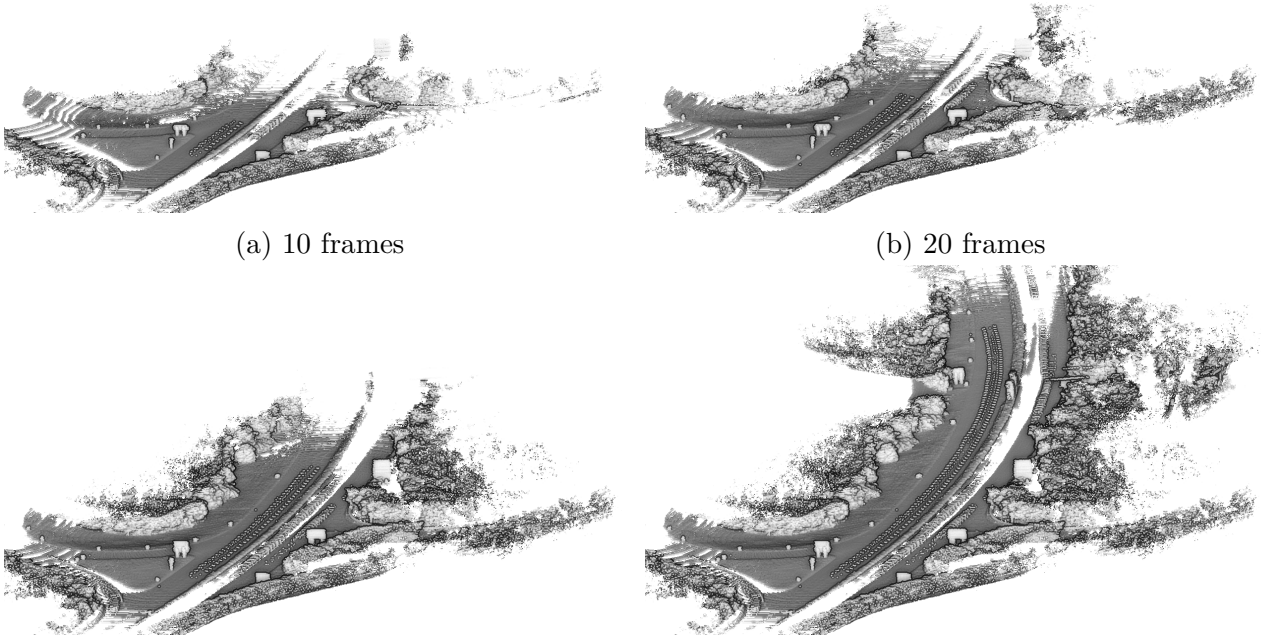


Figure 3.20: Figure describing the vertex map construction process. Image from the *LO-Net* [65] article.

Here $\mathbf{px}^n = (\mathbf{px}_x^n, \mathbf{px}_y^n)$ describes the normalized coordinates of the projection. We obtain the pixel coordinates of a $\mathcal{VM}$ of resolution $Height \times Width$ by multiplying the normalized pixels by the resolution : $\mathbf{px}_x = \lfloor \mathbf{px}_x^n \times Height \rfloor$, $\mathbf{px}_y = \lfloor \mathbf{px}_y^n \times Width \rfloor$. Here, we considered 360 rotating lasers and thus restricted the field of view only vertically using parameters FoV_{up} and FoV_{down}. Figure 3.20 describes this $\mathcal{VM}$ formation process. Given a vertex map $\mathcal{VM}$, we compute a normal map $\mathcal{NM}$ following [2]'s fast method for range images, using a box filter. Figure 3.19 shows a vertex map computed on a single frame from KITTI's dataset.

This method gives inexact neighbor association, and approximate normal (compared to using a Kd-Tree), but the interest of this method lies only in its computational efficiency. Not only does it require only $O(1)$ operations per query, it can be parallelized efficiently on a GPU, which is impossible for tree traversal. And while SuMa used OpenGL shaders to compute efficiently the projection on the GPU, we used the PyTorch library with `pyLiDAR-SLAM` package.



(a) 10 frames (b) 20 frames

(c) 50 frames (d) 100 frames

Figure 3.21: Aggregated maps for different sliding window sizes.

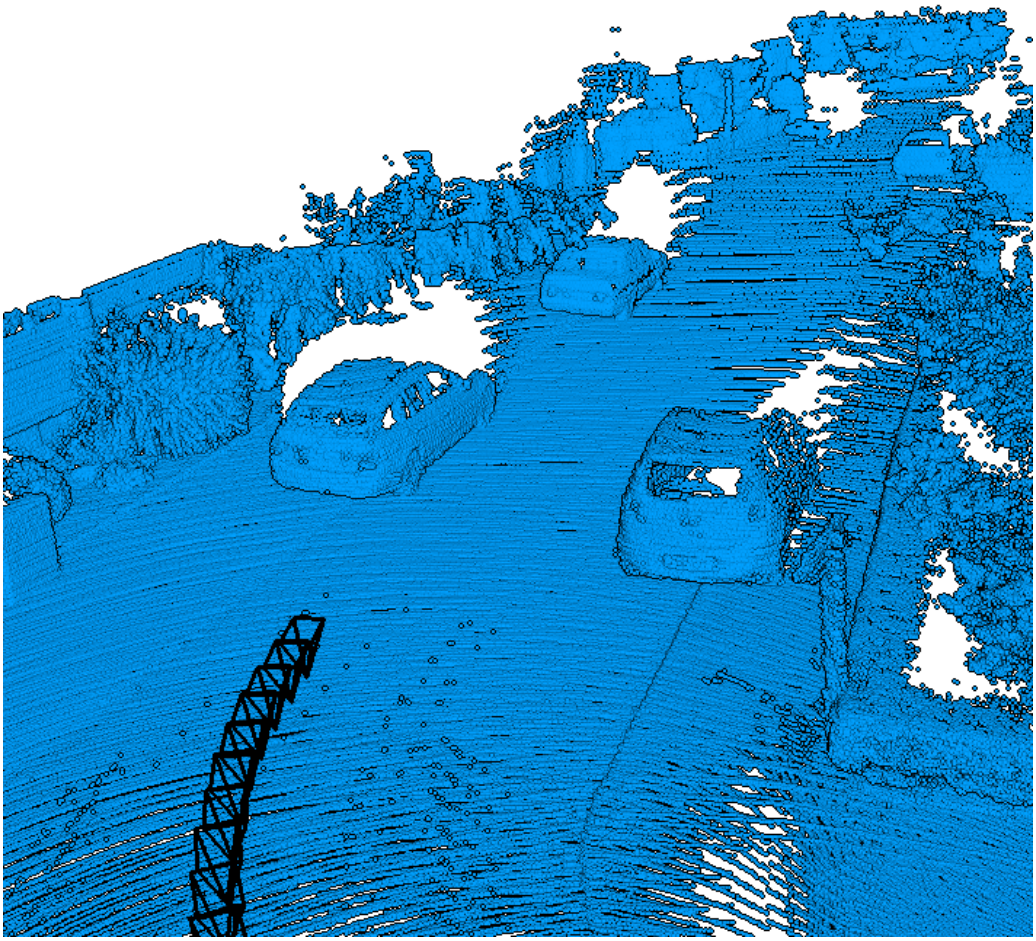
3.3.4 Map management

Once the registration is completed, the frame is inserted into the map. In this work we adopt a simple scheme similar to [26]. Our map is set as a sliding window of the N previous frames, so an insertion first requires the addition of the new frame to the sliding window along the associated pose, and the removal of older frames. The frames inserted are themselves sampled according to the problem's scale.

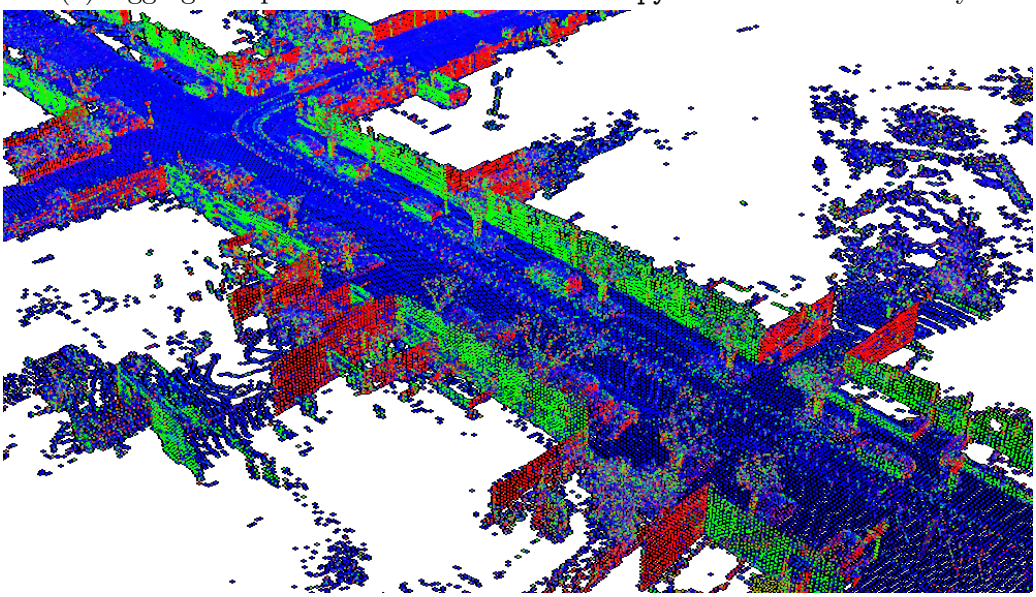
After each addition, each frame is put in the reference frame of the last inserted pose and then the neighbor association model is updated. For the projective association, this implies projecting the map into a vertex map, while for the Kd-Tree model, a new Kd-Tree is constructed and stored in the map.

The density of the map point cloud depends on the size of the sliding window, the larger the window, the higher the density. To be more precise, the density of the point cloud depends on the motion of the sensor. As the sensor moves its lasers intersect unseen parts of the environment and add information to the map, while a static sensor always intersects the same points. To mitigate this, we only add frames to the map if the cumulated distance from the last insertion is at least 50cm or 10° in driving scenarios, and $10cm$ for other scenarios.

Globally, a higher density leads to more information and thus should lead to a higher precision of the registration. We show in figure 3.21 the aggregated point cloud for different sizes of sliding windows. However a larger map also leads to a larger memory footprint and runtime, and as no voxelization is performed, duplicate information is stored in the map when very close points are inserted. The performance of the map depending on its parameters (sliding window size, preprocessing's sampling size) are studied in section 3.5. We show a representation dense point cloud reconstructed by this odometry and show a local map for the KD-Tree registration procedure in figure 3.22.



(a) Aggregated point cloud constructed with `pyLiDAR-SLAM`'s odometry.



(b) Local map for `pyLiDAR-SLAM` odometry, colored by normal direction

Figure 3.22: Aggregated point cloud, and map for `pyLiDAR-SLAM odometry`.

3.3.5 Conclusion

In this section, we presented the classical LiDAR Odometry pipeline implemented within `pyLiDAR-SLAM`. We will show section 3.5 that despite the simplicity of its design this LiDAR odometry reaches near state-of-the-art performance on the KITTI dataset.

The work presented here, and the associated experiments (see section 3.5) was also part of the work, published at the *IROS-2021* international robotics conference: *What's In My LiDAR Odometry Toolbox?*. Notably, the classical pipeline presented above serves as a baseline classical approach to compare with hybrid and deep methods. We present the other and principal aspects of this work in chapter 4.

The classical pipeline we just introduced, serves as a good introduction to point-cloud-based LiDAR odometry approaches. Still, this approach has some issues preventing its practical use in real-life scenarios. The first problem is the overall precision of the method on real data. We mentioned that this approach has near-state results on KITTI. However, the KITTI dataset is a particularly easy dataset for LiDAR odometry, notably, because it removes the frame distortion problem by providing motion-compensated scans, also because the environment presents very few challenges, except for one or two sequences (e.g. sequence `01`). When texting in different contexts we show unsatisfactory results for this method.

The last problem is the overall runtime performance of our approach. We are limited in this work, by the map management approach but also by the python language, which prevents us to treat each point individually, and restricts us to batch operations implemented in the standard linear algebra libraries (`numpy, pyTorch`).

For these reasons, we proposed a novel LiDAR Odometry method, *CT-ICP*, implemented in C++ which addresses all these points. We present this work in the next section.

3.4 *CT-ICP:* State-Of-The-Art Real-Time Odometry

Following the limitations of `pyLiDAR-SLAM`'s LiDAR Odometry presented above, and of Jean-Emmanuel Deschaud's IMLS-SLAM [26], we propose a novel real-time, state-of-the-art LiDAR Odometry.

This new approach notably builds on [26], which was when it was written already a state-of-the-art method on KITTI [40], though with very slow runtime (1Hz). This work by contrast, additionally to setting a new state-of-the-art, is faster than real-time, on a personal computer. Key to the precision of this approach is the novel method to handle the frame distortion of CT-ICP. Furthermore, the code is released in open-source with a publically permissive license, and available at `https://github.com/jedeschaud/ct_icp`.

The remainder of the section is organized as follows: first in section 3.4.2 we present our principal contribution, the elastic registration procedure which compensates during the optimization of the frame distortion. Then in section 3.4.3 the neighborhood construction process and the map management are proposed. And finally in section 3.4.4 we present the initialization and sampling approach proposed in CT-ICP.

3.4.1 Contributions

The list contributions of this work are the following:

- A novel method to handle the distortion of a point cloud, based on an elastic registration procedure presented in section 3.4.2.
- A state-of-the-art performant LiDAR Odometry method tested on a large variety of datasets.
- An open-source implementation made publically available.
- A neighborhood construction with an on-the-fly normals computation.
- A two-step grid-sampling approach, which significantly improves the speed of the odometry, without sacrificing the quality of the registration.

3.4.2 Continuous-Time Registration

We previously presented the necessity of handling the distortion of the point cloud, as well as the different methods existing in section 3.2.2.3. Our registration method aims to propose an alternative solution to this issue. The general idea is to formulate a cost function that will encourage the distortion of a frame to adhere to the environment's structure. The crucial difference between our method and the previous ones evoked in section 3.2.2.3 is that we set the distortion as an objective of the optimization step, rather than as a preprocessing or post-processing step.

The way we do this is by introducing 6 additional degrees of freedom which provide the frame's elasticity, and model the trajectory continuously. The resulting 12 degrees of freedom correspond to a pair of the 6 parameters of $SE(3)$. More precisely, introducing $\mathbf{X} = (\mathbf{T}_b, \mathbf{T}_e) \in SE(3)^2$, and provided each key point of the new frame $\mathcal{PC}^{kpts}$ has accurate timestamps, our formulation estimates, for each residual, the pose at a given timestamp $\alpha_i \in [0, 1]$ by linearly interpolating between $\mathbf{T}_b : \alpha_i = 0$ and $\mathbf{T}_e : \alpha_i = 1$. So the overall geometric cost function, which essentially incorporates the distortion to a point-to-plane cost, is as follows:

$$E_{\text{CT-ICP}}(\mathbf{X}) = \frac{1}{2}\sum_i \rho(r_i^2(\mathbf{X})), \quad \text{where} \tag{3.39}$$

$$r_i(\mathbf{X}) = a_i \cdot (\mathbf{p}_i^W(\mathbf{X}) - \mathbf{q}_i^W) \cdot \mathbf{n}_i \tag{3.40}$$

$$\mathbf{p}_i^W(\mathbf{X}) = \mathbf{R}^{\alpha_i}(\mathbf{X}) * p_i^L + \mathbf{t}^{\alpha_i}(\mathbf{X}) \tag{3.41}$$

$$\mathbf{R}^{\alpha_i}(\mathbf{X}) = \text{Slerp}(\mathbf{R}_b, \mathbf{R}_a, \alpha_i) \tag{3.42}$$

$$\mathbf{t}^{\alpha_i}(\mathbf{X}) = (1-\alpha_i)\mathbf{t}_b + \alpha_i * \mathbf{t}_e \tag{3.43}$$

$$\mathbf{q_i}^W := \text{Nearest neighbor of } \mathbf{p}_i^W \text{ in the map} \tag{3.44}$$

$$a_i := \text{Weight defined from neighborhood's covariance} \tag{3.45}$$

We incorporate the weight $a_i = (\sigma_2 - \sigma_3)/\sigma_1$ where σ_i is the i^{th} square root of the eigen value of $\mathbf{p}_i^W$'s neighborhood in the map. This weight, already introduced in [25] favors planar neighborhoods, and mitigates linear (e.g. small posts) of volumic (e.g. vegetation) neighborhoods which negatively impact the point-to-plane error.

While this is rarely the case, this error alone can lead to divergence during the optimization. Indeed, for points having a timestamp around the middle of the frame $\alpha_i \approx 0.5$, a continuous pair of poses ($\mathbf{T}_b$, $\mathbf{T}_e$) can be interpolated to give the same pose $\mathbf{T}_i$ for time α_i. Thus when a frame is in an environment where there are no strong geometric constraints for points at the beginning and end of the frame, this can cause some problems. To mitigate this, we introduce some additional constraints on the trajectory $C_{\text{loc}}(\mathbf{X}), C_{\text{vel}}(\mathbf{X})$ which are defined as follows:

$$C_{\text{loc}}(\mathbf{X}) = \|\mathbf{t}_b - \mathbf{t}_e^{prev}\|_2^1 \tag{3.46}$$

$$C_{\text{vel}}(\mathbf{X}) = \|(\mathbf{t}_e - \mathbf{t}_b) - (\mathbf{t}_e^{prev} - \mathbf{t}_b^{prev})\|_2^2 \tag{3.47}$$

Where $\mathbf{t}_*^{prev}$ is the respective translation component of the previous pose. So the overall optimization objective of CT-ICP is:

$$\delta\mathbf{X}^* = (\delta\mathbf{T}_b^*, \delta\mathbf{T}_e^*) = \underset{\delta\mathbf{X}}{\text{argmin}}\, E_{\text{CT-ICP}}(\delta\mathbf{X} * \mathbf{X}) + C_{\text{loc}}(\delta\mathbf{X} * \mathbf{X}) + C_{\text{vel}}(\delta\mathbf{X} * \mathbf{X}) \tag{3.48}$$

In this work, we solve this objective using the Gauss-Newton algorithm, similarly as presented above in section 3.3.2. The overall registration algorithm, the optimization apart, is identical to the ICP-based algorithm presented in section 3.3.2: iteratively, the neighbor associations are updated, and the above error cost is minimized. To compute the neighbor associations, each point is expressed with the same linearly interpolated pose as presented in equation 3.41. Figure 3.23 presents a schema of the registration procedure within the odometry.

Why not set $\mathbf{T}_b$ to $\mathbf{T}_e^{prev}$? Taking a step back from our method, it appears counter-intuitive not to set the pose at the beginning of the new frame (*ie* $\mathbf{T}_b$) to the pose at the end of the previous frame (*ie* $\mathbf{T}_e^{prev}$). In this case, only 6 degrees of freedom would be necessary for registration. Indeed, the trajectory is continuous, so these two should be equal if no wrong insertions were made.

This is however by design. Through experiments, we found that when if we fix the initial pose in this manner, progressively as frames are registered, translational errors introduced get overcompensated with the optimized pose of the beginning of the frame. This leads to trajectory divergence, with poses oscillating above and below the ground. Instead, we found that allowing the trajectory some discontinuity between frames allowed the algorithm to correct errors introduced, and globally led to more robustness.

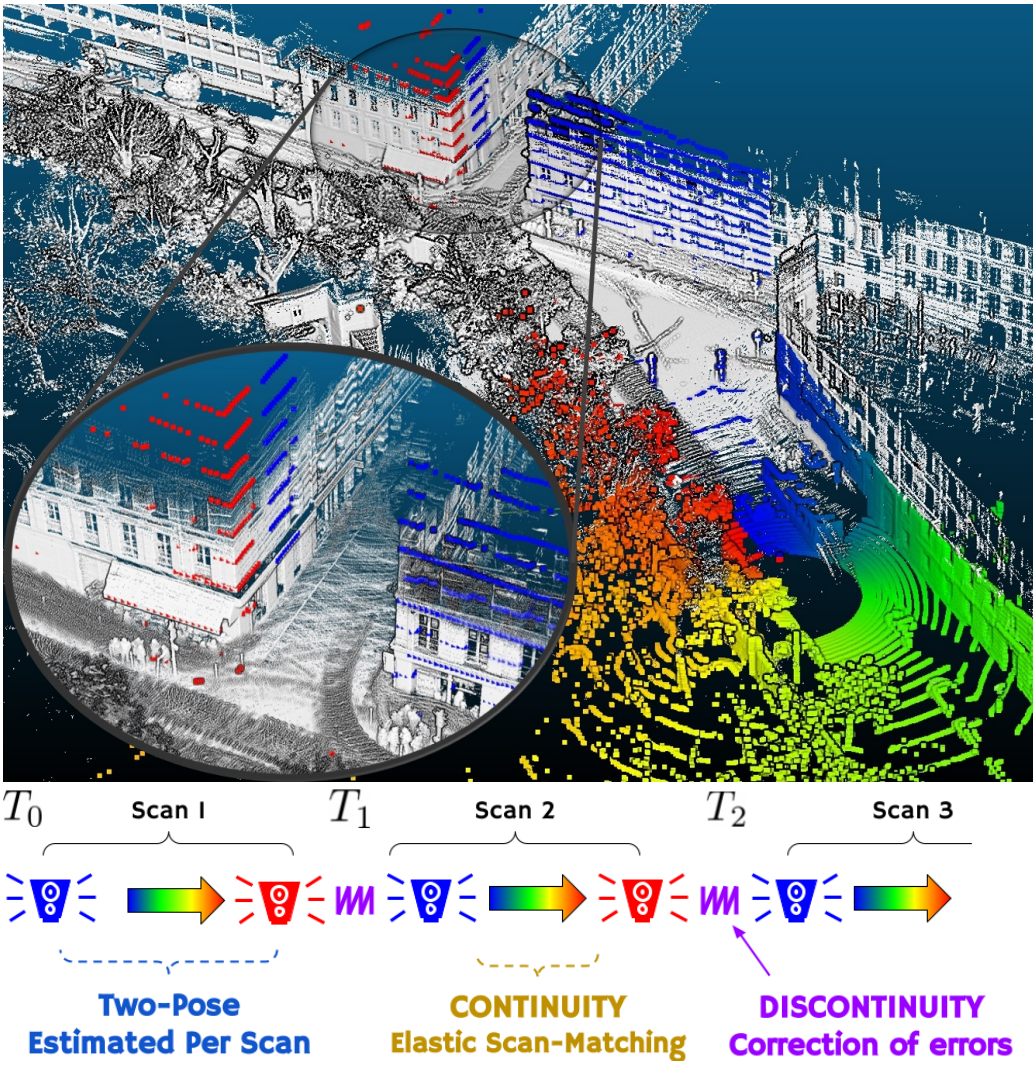


Figure 3.23: Image describing the main contribution of our work [25]: the elastic registration procedure consisting of a continuous scan matching, with allowed discontinuity between scans allowing for some error corrections.

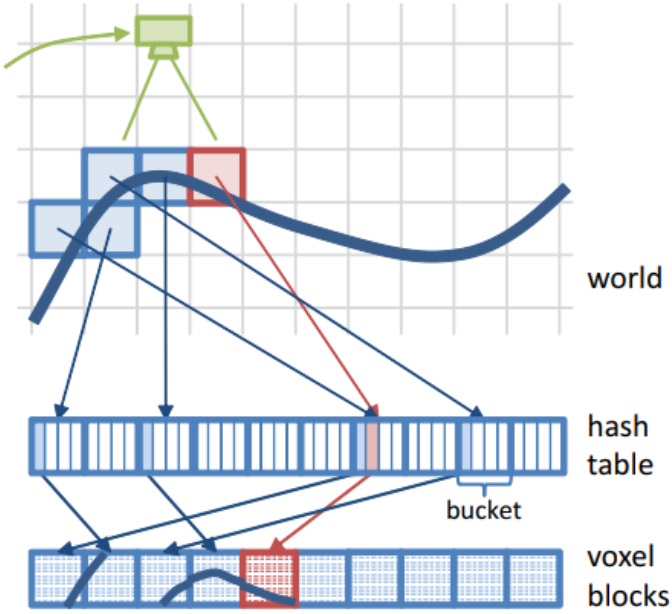


Figure 3.24: Voxel hashing scheme description. Points of the world's surface are assigned to a voxel block using a hash table.

3.4.3 Neighborhood Construction & Map Management

As mentioned above, the principal bottleneck of the approach proposed in [26] and [24] is the map management selected. Similarly to these approaches, our map is a dense point cloud augmented with a spatial query acceleration structure. We use a voxel hashing scheme [78] for this. The point cloud is stored in a voxel hash-map $\mathcal{M}$. For each voxel (*ie* the key in the map), we store a voxel block as a value, which is simply an array of the inserted points falling into the voxel. The voxel hash map is parametrized by the voxel size σ.

Insertion Given a point cloud $\mathcal{PC}^W$ to insert in the map, for each point $\mathbf{p}_i = (x, y, z) \in \mathcal{PC}^W$, we compute its voxel coordinates as follows:

$$\mathbf{v}_i = (v_x, v_y, v_z) = (\lfloor x/\sigma \rfloor, \lfloor y/\sigma \rfloor, \lfloor z/\sigma \rfloor) \in \mathbb{Z}^3 \tag{3.49}$$

The point is then inserted in the array in the hash map for the key $\mathbf{v}_i$, which is allocated if it does not yet exist. To control the memory, we limit the number of points in the map to N_{vox}, and to avoid inserting duplicate information we only insert points if their minimum distance with another point in a voxel is at least d_{min}.

Neighborhood Construction At each ICP iteration, and for every key point, we compute its nearest neighbor in the map, and estimate the associated normal. To do this, we construct a local neighborhood for each key point by searching through the $N_r = (2r+1)^3$ voxels neighboring the key point. Here r is the voxel radius and is typically set to 1 to explore the direct neighbors of the key point's voxel, so we explore the $N_r = 27$ neighbors.

We test each entry in the hash map, using the following hash function on voxels to quickly test existence in the hash maps:

$$\text{hash}(v_x, v_y, v_z) = p_1 * v_x + p_2 * v_y + p_3 * v_z \tag{3.50}$$

Where $p_1, p_2, p3$ are the following prime numbers which guarantee good hash function with few intersections:

$$p_1 := 73856093 \tag{3.51}$$
$$p_2 := 19349669 \tag{3.52}$$
$$p_3 := 83492791 \tag{3.53}$$

So exploring the N_r entries in the map amounts to testing the N_r neighboring voxels hashes in the map:

$$\forall (\delta x, \delta y, \delta z) \in [-r, r]^3, \text{ test if hash}(v_x + \delta x, v_y + \delta y, v_z + \delta z) \in \mathcal{M}$$

Going through the N_r voxels, we keep the set of the N_n closest points to the key point from which we extract the closest point and the normal from the covariance similarly to 3.34. This neighborhood construction is fast, and what's more, it does not depend on the size of the map. Indeed, the construction has essentially a complexity in $O(1)$ (or $O(N_n \times N_r)$ at a finer scale), as the complexity of accessing an entry in a hash map is a $O(1)$ operation.

Voxel removal The interest of this data structure, in addition to the speed of queries, is the simplicity of management, *ie* of insertion and deletion operations. To remove a voxel, we simply erase the corresponding entry in the hash map. In this work, we simply remove voxels far from the current sensor location at each insertion. Like all parameters of the hash map, the appropriate distance depends on the scale of the problem. We explore the impact of the parameter values for the different configurations in more detail in section 3.5.

3.4.4 Initialization, Preprocessing and Sampling

Lastly, we present the remaining aspects of our SLAM which differentiates from [26] and [24]: the initialization of the new pose estimate before registration (initial distortion, motion model, etc...), and our sampling procedure.

Initialization The initialization of the motion in our algorithm requires some care, due to our distortion compensation. Initially, we do not assume any motion prior, thus we have no way of applying a correct distortion to the frame before inserting it into the frame. Thus if a vehicle or robot is already in motion, this necessarily introduces some errors in the map as we insert a distorted frame in the map.

The second frame is then registered against this map, *ie* the initial frame set in the absence of prior knowledge at a pose equal to the identity. For this second frame, we use a rigid registration with only 6 degrees of freedom (instead of the 12 of CT-ICP's continuous time registration), because if the vehicle is moving, the frame distortion between the two first frames should be similar (under constant velocity assumption). The distortion of this frame, in the constant velocity model, should match that of the initial frame, which would render the rigid registration valid. Then we apply the distortion to the new frame before inserting it into the map and removing the initial frame.

At this point, our map has a valid corrected frame, and we can initialize the pose parameters of the new frame using the constant velocity model, as follows. Noting $\mathbf{X}^k = (\mathbf{T}_b^k, \mathbf{T}_e^k)$ the pose parameters of the k^{th} frame registered, we thus have the following initialization procedure for the constant velocity model:

$$\mathbf{X}^0_{init} = (\mathbf{Id}_{SE(3)}, \mathbf{Id}_{SE(3)}) \tag{3.54}$$
$$\mathbf{X}^1_{init} = (\mathbf{Id}_{SE(3)}, \mathbf{Id}_{SE(3)}) \tag{3.55}$$
$$\Delta\mathbf{T}^k = \left(\mathbf{T}^{k-1}_b\right)^{-1} * \mathbf{T}^k_b \tag{3.56}$$
$$\mathbf{X}^{k+1}_{init} = (\mathbf{T}^k_b * \Delta\mathbf{T}^k, \mathbf{T}^k_e * \Delta\mathbf{T}^k) \tag{3.57}$$

Note that we do not use the frame distortion's relative transformation to initialize the relative motion (*ie*, $\left(\mathbf{T}^k_b\right)^{-1} * \mathbf{T}^k_e$), but instead use the relative transformation between two consecutive begin poses (see equation 3.56). This helps not to propagate distortion errors during the initialization phase. As a preprocessing step for the k^{th}, we initialize the distortion of the new frame using the estimated poses $(\mathbf{T}^k_{init,b}, \mathbf{T}^k_{init,e})$. These estimates are then refined by the continuous time registration described in section 3.4.2.

Sampling In this work, we adopt a two-step grid-sampling approach. First, each new frame is sampled with a grid sample at a resolution of c_{global}. The purpose of this sample is to filter out finer information that would not be relevant to the scale of the problem when inserted in the map.

After this sampling, a new sampling is performed at a coarser resolution c_{kpts} before the registration. This sampling selects the key points to register, and its main purpose is to reduce the computation load during the registration. Indeed, we can express the processing time of the registration τ_{ICP} as:

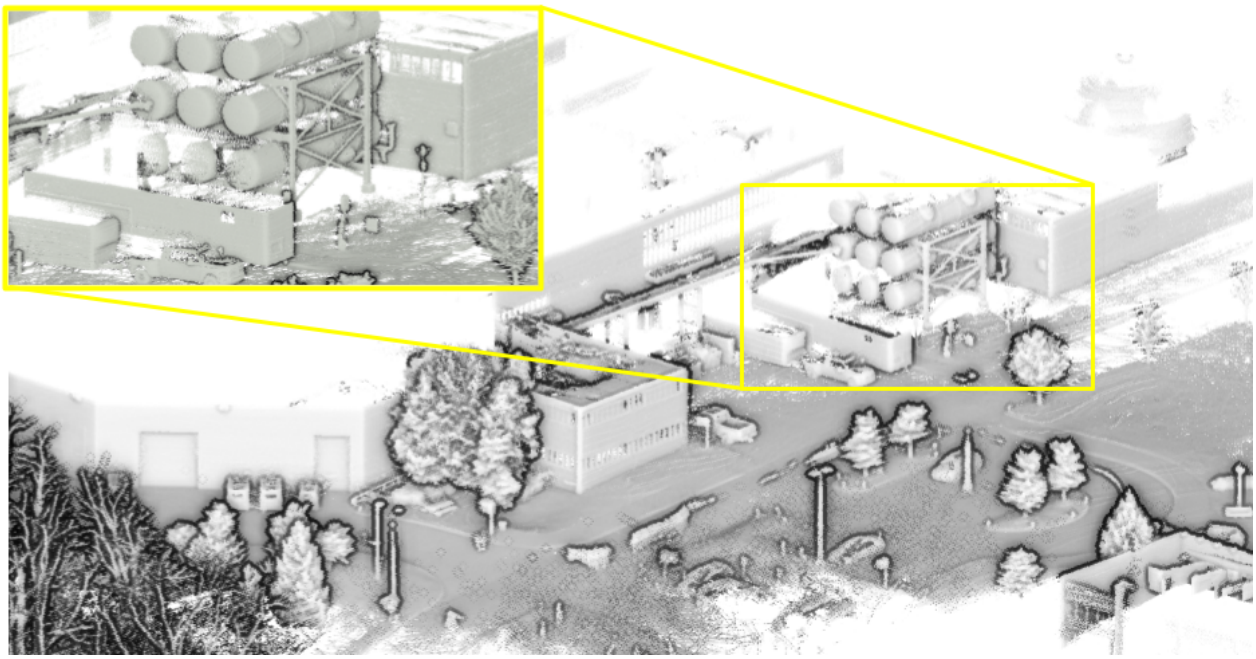
$$\tau_{\text{ICP}} \approx K_{\text{iters}} \cdot (N_{kpts} \cdot \tau_{neigh} + \tau_{LS}) \tag{3.58}$$
$$K_{\text{iters}} := \text{Number of ICP iterations} \tag{3.59}$$
$$N_{kpts} := \text{ Number of key points selected} \tag{3.60}$$
$$\tau_{LS} := \text{ Time of solving the least-square problem} \tag{3.61}$$
$$\tau_{neigh} := \text{ Time of constructing the neighborhood of a point} \tag{3.62}$$

Equation 3.58 clearly shows the relevance of limiting the number of key points via a coarse sampling to accelerate the processing time of a frame. Note that sampling leads to loss of information. As mentioned before, this leads to a tradeoff between the quantity of information preserved and the accuracy of the algorithm. We investigate in section 3.5 this tradeoff for different values of c_{global} and c_{kpts}.

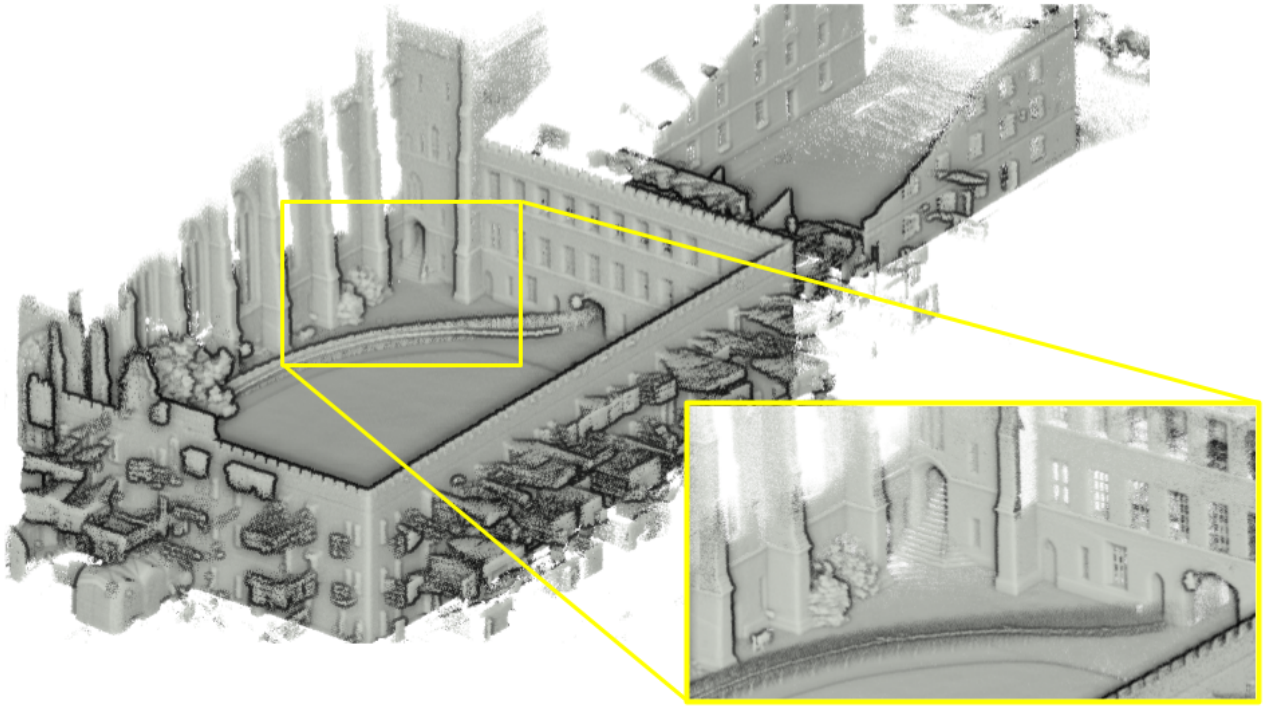
3.4.5 Conclusion

In this section, we presented the different components of *CT-ICP*, our real-time LiDAR odometry. In figures 3.25,3.26 and 3.27, we show point cloud aggregated using CT-ICP's odmetry. This illustrates qualitatively, the quality of dense 3D reconstruction which is possible using 3D LiDAR. Even more impressive, is the speed at which this reconstruction is obtained, typically faster than real time on a cpu. By comparison, photogrammetry methods, *ie* the methods comparing light intensities between multiple frames to build the geometry, can reconstruct dense geometry from images, however, these operations are extremely computation-expensive, and are very slow (multiple hours with GPUs for reconstructions of that scale).

In the next section, we prove through thorough experiments, across multiple datasets, the claims we made above, concerning both methods presented.



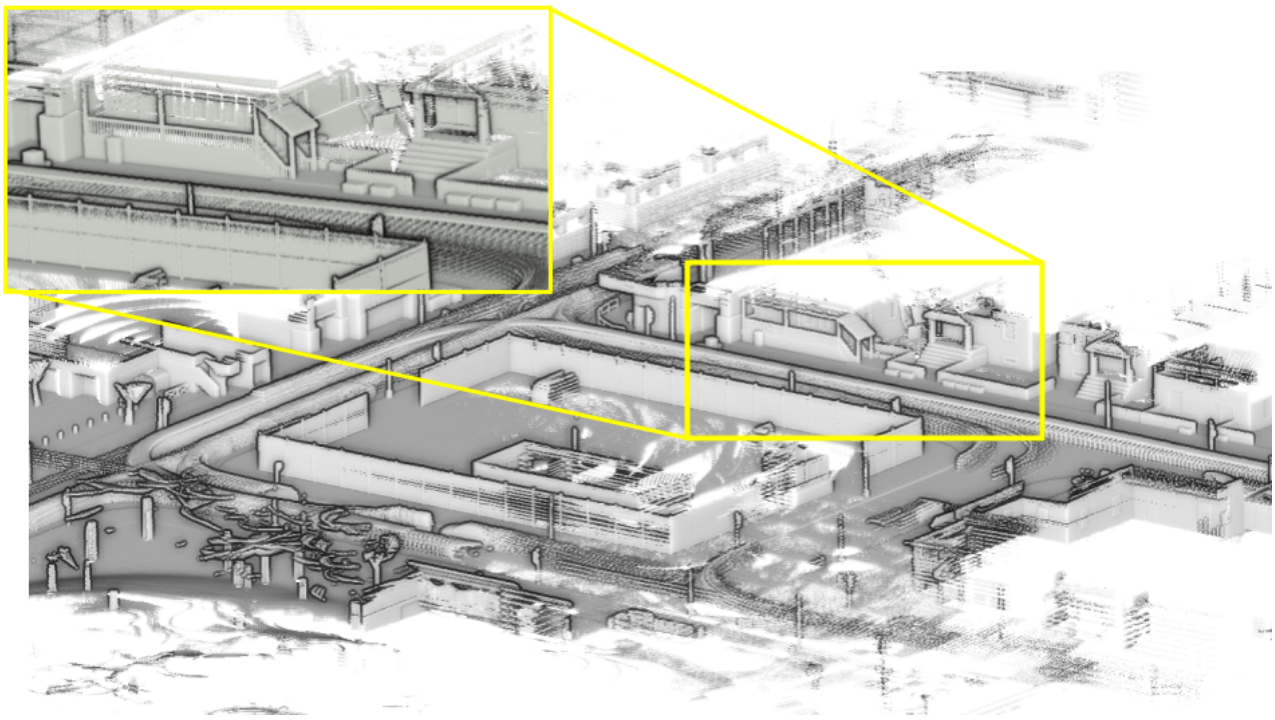
(a) Aggregated point cloud frames, using CT-ICP's trajectory on sequence `2012-01-08` of the **NCLT** dataset.



(b) Aggregated point cloud frames, using CT-ICP's trajectory on sequence `exp01` of the **NCD** dataset.

Figure 3.25: Point cloud assembled using CT-ICP's odometry on mobile robotics (Top) and handheld (Bottom) datasets. Illustrates the quality of the point cloud obtained by CT-ICP.

(a) Aggregated point cloud frames, using CT-ICP's trajectory on sequence `00` of the **KITTI** dataset.



(b) Aggregated point cloud frames, using CT-ICP's trajectory on sequence `Town02` of the **KITTI-CARLA** dataset.

Figure 3.26: Point cloud assembled using CT-ICP's odometry on driving datasets. Illustrates the quality of the point cloud obtained by CT-ICP.

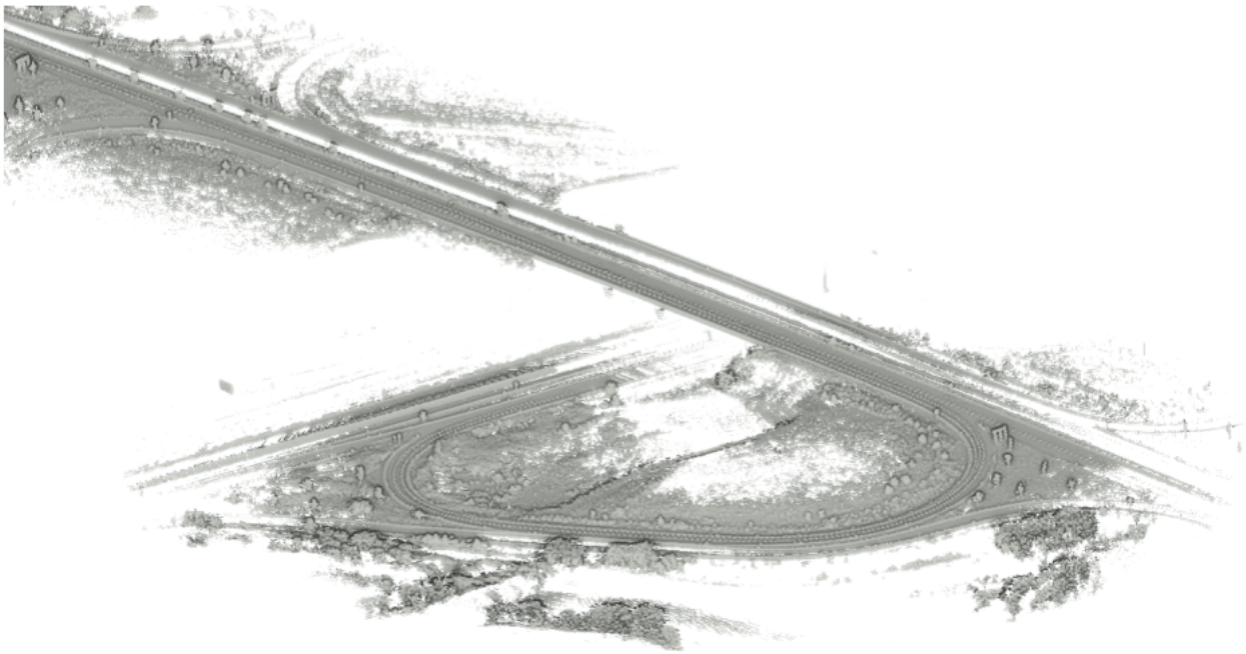


Figure 3.27: Aggregated point cloud frames, using CT-ICP's trajectory on the highway sequence (`01`) of the **KITTI** dataset.

3.5 Experiments and Benchmark

In this section, we finally prove the claims we made above, concerning our two LiDAR Odometries. These claims were:

- `pyLiDAR-SLAM`'s simple LiDAR Odometry pipeline is near-state-of-the-art on the KITTI dataset [40].
- *CT-ICP* sets a new state-of-the-art on KITTI and multiple other datasets.
- *CT-ICP* has very good performances on a wide range of scenarios.
- *CT-ICP* obtained these precise results faster than real-time.
- *CT-ICP*'s distortion handling helps improves the accuracy of our method.

Additionally to these claims, we study different components of CT-ICP using abblation studies, to give even more insight into the performance of the algorithm.

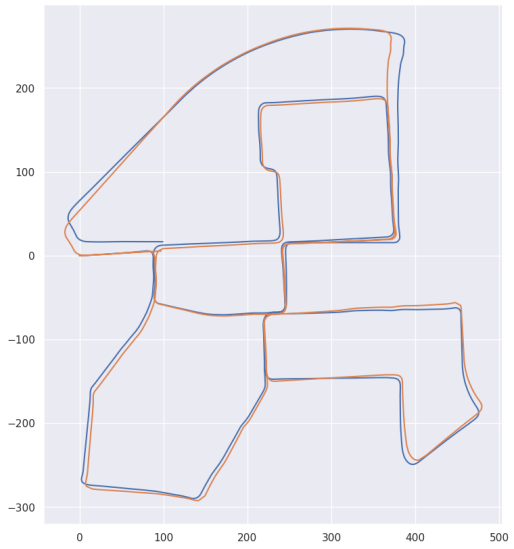
The rest of the section is organized as follows: first, in section 3.5.1, we present the main results table of the CT-ICP algorithm presented in section 3.4, and compare it to some other state-of-the-art methods, and show the CT-ICP did set a new state-of-the-art in terms of precision. Then in section 3.5.2 we take a deeper dive and leverage all our datasets to showcase the strengths and weaknesses of our algorithm. This is done through multiple ablation studies, and by isolating some components of the odometry to study its impact on the overall performance. Finally, in section 3.5.3 we offer a detailed performance and robustness analysis of the odometry, and show its potential use for embedded platforms.

3.5.1 `pyLiDAR-SLAM` (near) state-of-the-art, CT-ICP state-of-the-art

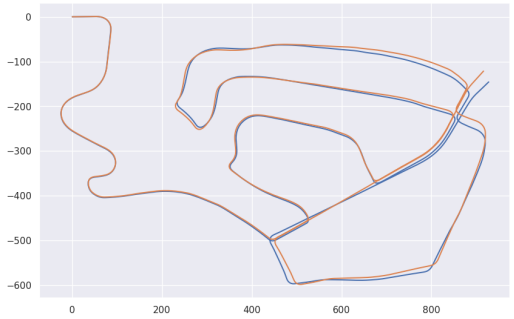
To benchmark our odometries, we evaluate both our methods on the set of datasets presented in chapter 2. Both methods, as we will explain in more details at the end of this chapter were published in 2021 (`pyLIDAR-SLAM`) and 2022 (CT-ICP). Since then, we continuously refined *CT-ICP* (using the Cauchy instead of a truncated loss, cleaning the initialization procedure, and with smaller implementation details which improved drastically the performance). So in the experiments below, we show two results: *CT-ICP(paper)*, corresponding to the results at the time of publication, while *CT-ICP(now)* describes the version presented above. In contrast, `pyLiDAR-SLAM` did not change at all, so we have only one entry for it.

As described in chapter 2, we identified in the datasets we presented three categories: **Driving**, **Mobile Robotics**, and **Handheld**. For each category, we associated a relevant scale to measure the **RPE** (see table 2.1). *CT-ICP* only needs, however, two sets of parameters, one for driving **Dr**, and one for mobile robotics and handheld **MB&H** datasets.

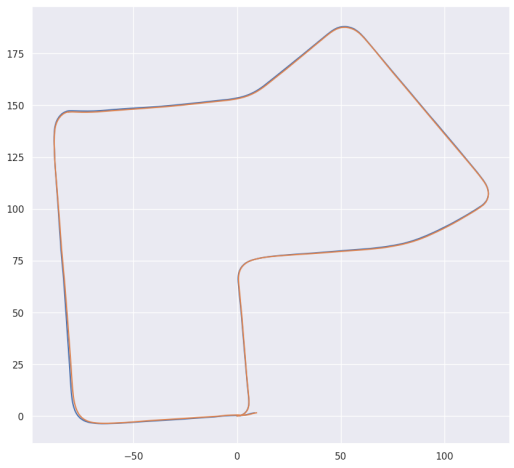
We present in section 3.5.1.1 the choice of parameters, for both CT-ICP and `pyLiDAR-SLAM`. Then we present and analyze the results in section 3.5.1.2, and show that they demonstrate some of our claims. The rest of them will be shown in section 3.5.2.



(a) KITTI-00



(b) KITTI-02



(c) KITTI-07

Figure 3.28: Slam trajectories (blue) for a selection of datasets, compared with ground truth (orange) (1/3).

Table 3.1: Parameters of `pyLiDAR-SLAM`, for the best results on the KITTI dataset.

Parameter	Value	Description
Preprocessing:		
Grid Sampling size	0.3m	The voxel size of the grid sampling to reduce the dimensionality of the point cloud.
CV Distortion	✓	Applies Constant Velocity distortion as a preprocessing step.
Initialization:		
Motion Model	CV	Constant Velocity: motion model used to initialize the motion. **CV** means Constant Velocity.
Map Managemenet		
Local window size	20	Number of point clouds in the local window
Registration Type	**Kd**	The type of neighbor association (either Projective **Pr** or Kd-Tree based **Kd**).
Registration Parameters		
Robust Loss	Cauchy	The robust loss function scheme used
Sigma(σ)	0.3m	The robust loss function parameter controlling the resistance to outliers.

3.5.1.1 Parameters and configurations

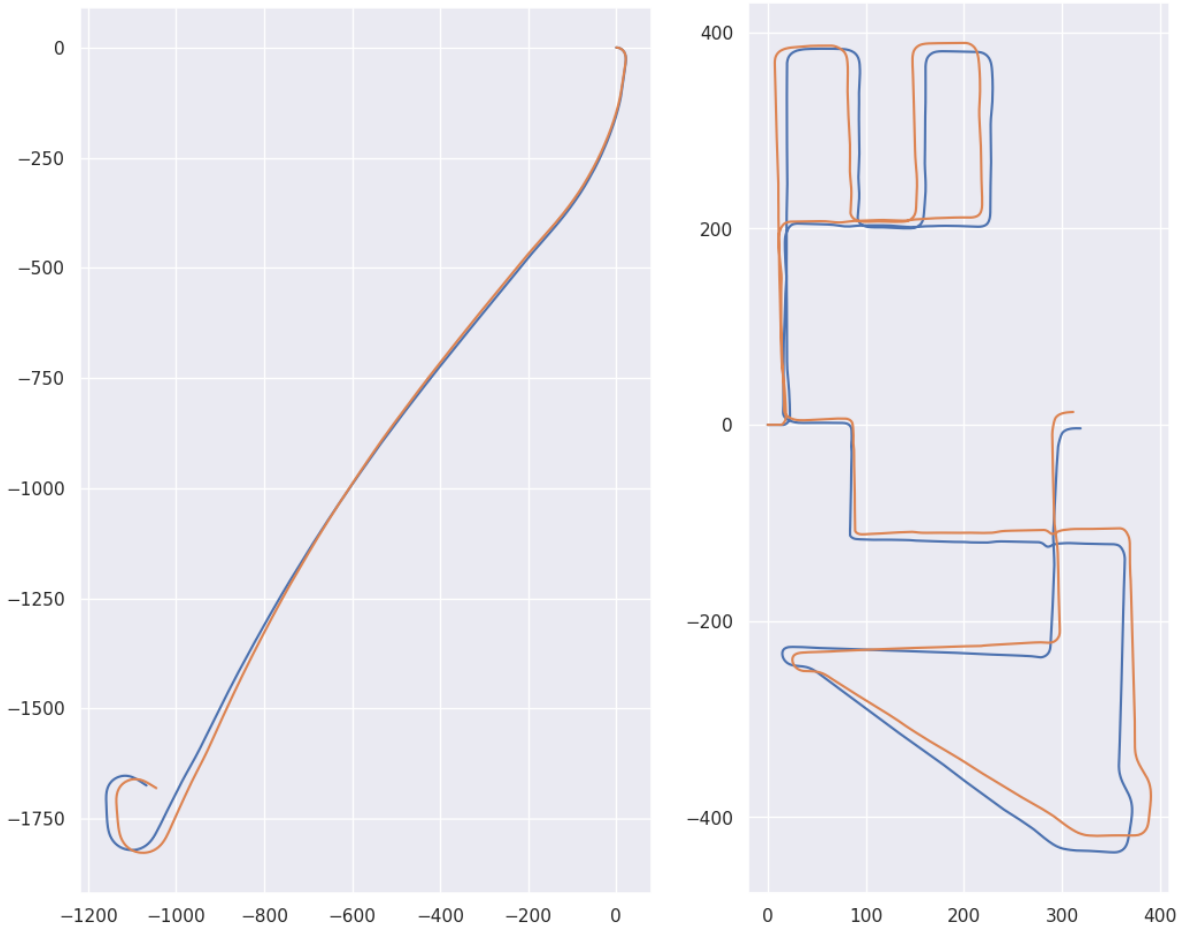
Our algorithms, like most LiDAR SLAMs, does not have a single set of parameters fitting all scenarios and use cases. As mentioned in chapter 2, each scenario comes with multiple inherent scales. And like most others, our algorithms are not designed to adapt to these different scales online, and we need to set some fixed parameters adapted for each desired scenario.

In the case of CT-ICP, and to clarify the role of parameters, we do not optimize the parameters of the SLAM for each sequence of each dataset. Instead, we propose two distinct parameter profiles, for the three scenarios presented above, which we now detail.

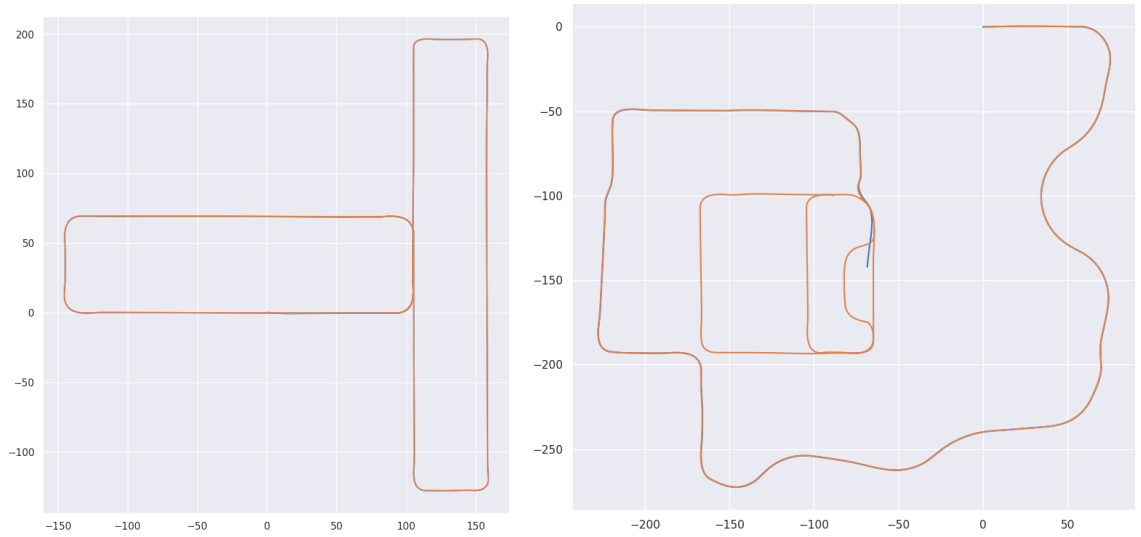
Parameters of `pyLiDAR-SLAM` We present the parameters for the Kd-Tree based LiDAR Odometry only, for clarity, as it is the method which obtains the best results on KITTI [40] but also for the widest range of datasets (while the Projective version has important drops in performance less outside of driving datasets). The parameters are summarized in table 3.1. We selected this set of parameter to obtain the best performance on KITTI, without sacrificing too much other datasets. Some of these parameters control the runtime performance of the algorithm, such as the **Local window size**, **Grid Sampling**. The others (σ, **Motion Model**) control the precision and robustness.

Parameters of CT-ICP The principal parameters of our algorithm CT-ICP are summarized in table 3.2. The first group of parameters is described as **Geometric parameters**. This includes the voxel sizes for the initial preprocessing grid sampling, the key points selection and for the resolution of the voxel map. For the driving profile (**Dr**), we have for all of them a larger value (resp. 0.5m, 1.5m and 1.0m) than for **Mb&H** (resp. 0.3m, 0.8m, 0.8m). The rationale behind this choice is related to the scales both of the environment and the relative motion between consecutive frames. In driving scenarios, the environment is typically larger than for the two other scenarios, thus a larger sampling size will not lose too much information about the environment in contrast with mobile robotics or handheld environments. So the relevant level of details for the registration is typically larger than for other scenarios. And thus, a larger sampling size helps enforce a coarser level of detail than a smaller sampling size.

This is counterintuitive. Indeed, at a high level, the sampling size should be seen as a tradeoff between precision and performance: the larger the sampling the fewer points need to be processed (and thus the faster the SLAM), but the more information is lost. Zooming

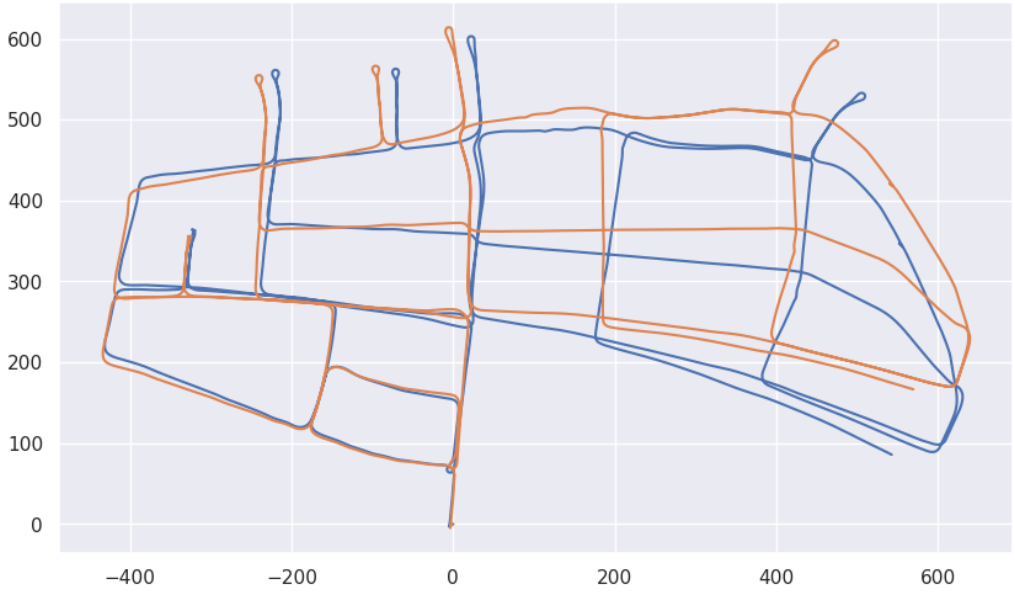


(a) `KITTI-01` (b) `KITTI-08`

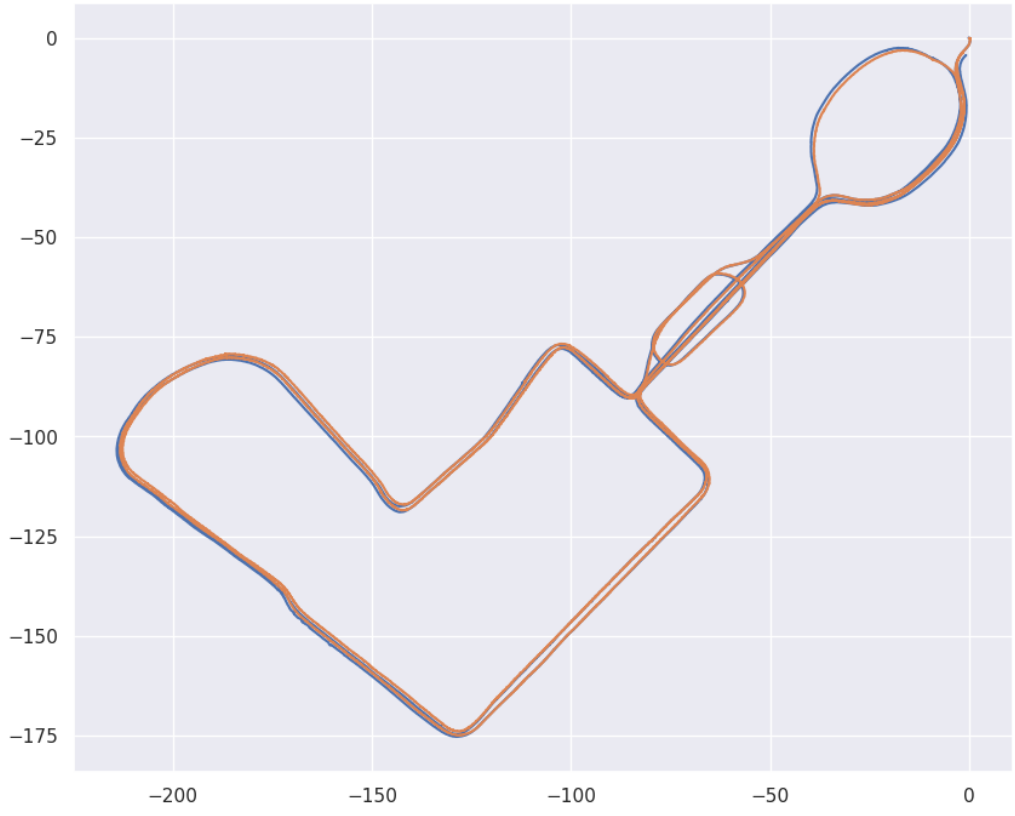


(c) `KITTI CARLA-Town01` (d) `KITTI CARLA-Town07`

Figure 3.29: Slam trajectories (blue) for a selection of datasets, compared with ground truth (orange) (2/3).



(a) KITTI-360-00



(b) NCD-01

Figure 3.30: Slam trajectories (blue) for a selection of datasets, compared with ground truth (orange) (3/3).

Table 3.2: CT-ICP Parameters description and values for the two different parameter profiles: **Dr** (Driving) and **Mb&H** (Mobile robotics and Handheld).

Parameter	**Dr**	**Mb&Hd**	**Description**
Sampling parameters			
Sample Size	0.5	0.3	Voxel size for the initial grid sampling preprocessing step.
Keypoints Sample Size	1.5	0.8	Voxel size for the grid sampling for the key points selection.
Map parameters			
Size voxel map	1.0	0.8	Voxel size of the voxel hash map, where the dense point cloud is stored.
Min distance points	0.15	0.1	Min distance between two points in the map.
Max points per voxel	30	30	The maximum number of points per voxel.
Motion Model Parameters			
Motion model	CV	None	The motion model to initialize the motion before registration.
Optimization parameters			
ICP iters	10	20	The maximum number of iteration of the ICP.
Tr threshold	0.01	0.01	Stop criterion of translation difference (in meter) for the ICP iterations.
Rot threshold	0.1	0.1	Stop criterion of rotation difference (in degrees) for the ICP iterations.
Sigma(σ)	0.1	0.05	The parameter of the Cauchy outlier rejection scheme.

in, however, the sampling size also has a regularization feature managing the noise (due to the sensor or registration errors) and brushes off smaller irrelevant details. And thus larger sampling size in the context of driving scenarios does improve the precision of our SLAM, while this is not the case for other scenarios. These detail these relationships through ablation studies in section 3.5.2. This rationale also explains the minimum distance between two points stored within the map.

The choice of the motion model is pretty straightforward, as a car typically adheres very well to the constant velocity model, while handheld devices typically do not and present very large accelerations. Mobile scenarios are typically in-between the two, but using the constant-velocity model was satisfying for all our experiments.

The second group of parameters regroups the **Optimization parameters**. They control the effort made within the ICP to obtain the most precise results possible. At each iteration, an optimal step is computed, if the ICP is successful, this step should converge to smaller values at each iteration. CT-ICP uses a stop criterion to qualify the convergence, using a threshold for the translation component (1cm) and of the rotation component (0.1°) of the optimal step. Additionally, we limit the maximum number of iterations in cases where the algorithm does not fully converge. Though we have the same stop criterion for all profiles, we set a smaller maximum amount of iterations for the ICP for the driving scenario. Driving scenarios are typically much easier, and the motion model is typically much more valid than the other scenarios.

The parameters we presented in table 3.2 are the principal parameters of the CT-ICP algorithm. However, in section 3.5.2, we present alternative configurations which will allow us to play not only on parameters, but also design choices of the algorithm, through relevant ablation studies.

Table 3.3: KITTI's Relative Pose Error (**RPE**), which is the relative pose error averaged for all segments of length (from 100m to 800m) in the trajectory. We compare the results of multiple state-of-the-art methods on multiple driving sequences and demonstrate the precision of our method.

DRIVING PROFILE													
KITTI-corrected*	00	01	02	X	04	05	06	07	08	09	10	**AVG**	ΔT
IMLS-SLAM [26]	0.50	0.82	0.53	0.68	**0.33**	0.32	0.33	0.33	0.80	0.55	0.53	**0.55**	1250 ms
MULLS [84]	0.56	**0.64**	0.55	0.71	0.41	0.30	0.30	0.38	**0.78**	0.48	0.59	**0.55**	80 ms
LOAM [135]	0.78	1.43	0.92	0.86	0.71	0.57	0.65	0.63	1.12	0.77	0.79	-	-
pyLiDAR F2M [24]	0.51	0.79	**0.51**	**0.64**	0.36	0.29	0.29	0.32	**0.78**	**0.46**	0.57	**0.53**	175 ms
CT-ICP (paper)	**0.49**	0.76	0.52	0.72	0.39	**0.25**	**0.27**	**0.31**	0.81	0.49	**0.48**	**0.53**	60 ms
KITTI-raw	00	01	02	X	04	05	06	07	08	09	10	**AVG**	ΔT
IMLS-SLAM [26]	0.79	0.86	0.76		0.51	0.48	0.62	0.67	0.97	0.70	0.75	**0.71**	1070 ms
MULLS [84]	1.43	3.12	1.01		0.57	1.93	1.60	0.69	1.28	1.49	0.71	**1.41**	80 ms
pyLiDAR F2M [24]	2.20	0.98	1.55		0.45	1.46	0.69	1.72	1.60	1.28	1.18	**1.61**	530 ms
CT-ICP (paper)	**0.51**	**0.81**	**0.55**		**0.43**	**0.27**	**0.28**	**0.35**	**0.80**	**0.47**	**0.49**	**0.55**	65 ms
CT-ICP (now)	0.52	0.68	0.51		0.37	0.26	0.30	0.34	0.81	0.47	0.52	**0.52**	59 ms
KITTI-360	00	02	03	04	05	06	07	09	10			**AVG**	ΔT
IMLS-SLAM [26]	0.65	0.63	0.64	0.89	0.63	0.70	0.54	0.67	0.79			**0.68**	1060 ms
MULLS [84]	1.60	1.29	0.87	1.64	1.27	1.48	6.25	1.28	0.88			**1.55**	90 ms
pyLiDAR F2M [24]	1.79	1.25	0.90	1.60	1.26	1.38	0.69	1.72	1.39			**1.46**	475 ms
CT-ICP (paper)	**0.41**	**0.38**	**0.34**	**0.65**	**0.39**	**0.42**	**0.34**	**0.45**	**0.69**			**0.45**	70 ms
KITTI-CARLA	Town01	Town02	Town03	Town04	Town05	Town06	Town07					**AVG**	ΔT
IMLS-SLAM [26]	**0.03**	0.05	0.16	0.20	0.06	4.90	0.25					**0.81**	780 ms
MULLS [84]	1.39	0.77	0.69	1.24	1.13	0.98	0.97					**1.04**	70 ms
pyLiDAR F2M [24]	16.25	7.92	37.16	10.06	9.11	73.29	2.69					**23.84**	530 ms
CT-ICP (paper)	0.03	0.04	0.03	**0.03**	0.02	0.04	0.41					**0.09**	65 ms
CT-ICP (now)	**0.011**	**0.012**	**0.013**	0.049	**0.010**	**0.017**	**0.084**						40 ms

Table 3.4: Relative Pose Error **RPE** averaged over all segments of 100m for each trajectory. This table shows the high performance of CT-ICP over concurrent methods.

MOBILE ROBOTICS PROFILE					
NCLT	2012-01-08	2012-04-29	2012-01-15	2012-05-11	ΔT
pyLiDAR F2M [24]	1.89	1.78	-	-	460ms
CT-ICP (paper)	1.17	1.09	-	-	180ms
CT-ICP (now)	1.12	1.07	1.16	1.34	180 ms

Hardware Setup We ran all experiments on a laptop with an Intel Core i7-9750 CPU @2.60GHz with 12 threads, and 64Gb of RAM. We present in the experiments below the average runtime for each run, but we present in section 3.5.3 a deeper analysis of the performance, and hardware requirements.

3.5.1.2 Results and Analysis

We now present a first pass of results for the CT-ICP algorithm on the dataset we presented above. These results demonstrate our claim that CT-ICP is a state-of-the-art real-time LiDAR odometry, which is a contribution presented in our work.

In table 3.3 we show the results using KITTI's **RPE** for the driving datasets. While the KITTI metric is less intuitive than **RPE** presented in chapter 2, it can be used to compare our method to the previous body of work. In tables 3.4 and 3.5 we describe the **RPE** at the relevant scales (resp. 100m and 20m) for the mobile robotics and handheld sensors.

We also show the trajectory of **CT-ICP** for a selection of sequences in figures 3.28, 3.29

Table 3.5: Relative Pose Error **RPE** averaged over segments of 20m for each trajectory of each handheld dataset with ground truth.

HANDHELD PROFILE				
Newer College Dataset	exp01	exp02	exp05	ΔT
pyLiDAR F2M [24]	1.94	3.72	✗	437 ms
CT-ICP (ours)	**1.13**	**1.21**	✗	33ms

and 3.30.

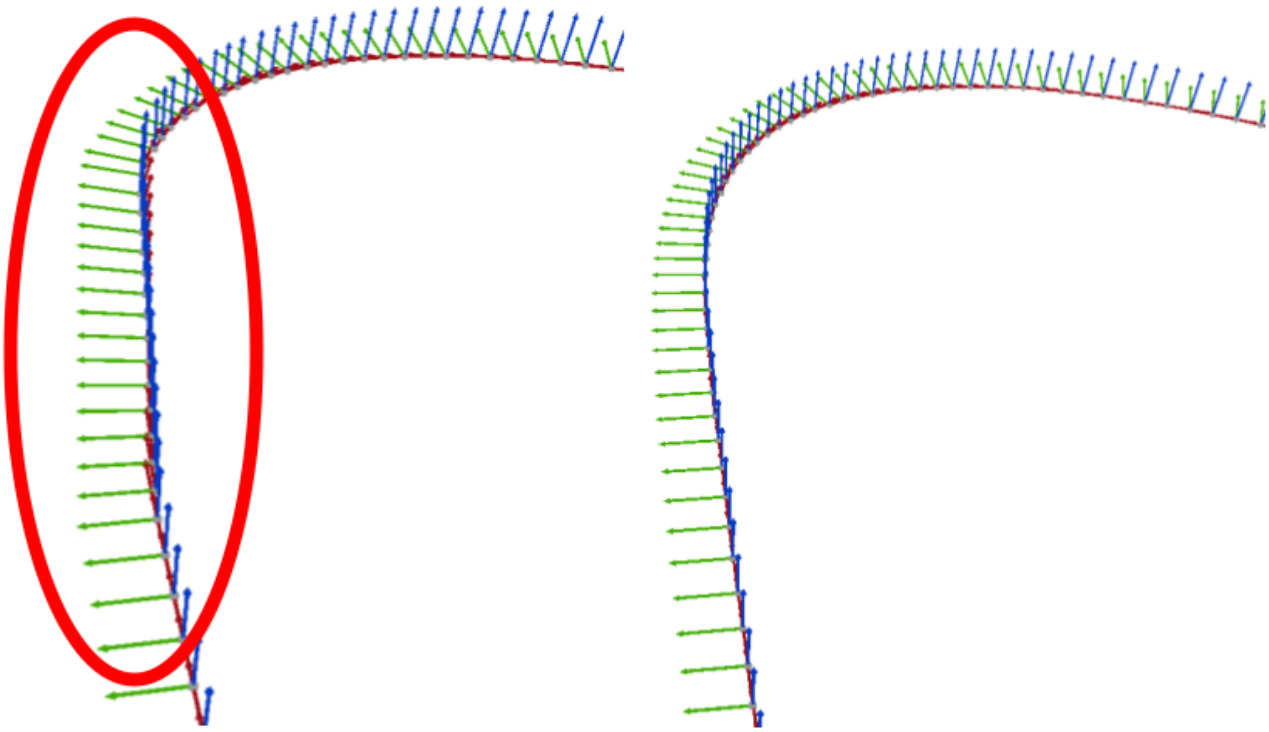
A first look at the table, and we see globally that two of our important claims were demonstrated: `pyLiDAR-SLAM` is a near-state-of-the-art LiDAR Odometry on KITTI [40], and CT-ICP is a state-of-the-art LiDAR Odometry operating in real time. Indeed, `pyLiDAR-SLAM` functions even better than MULLS [84] (previous state-of-the art on KITTI) and LOAM [135], the baseline method for LiDAR SLAM. We see however, that to obtain this performance, `pyLiDAR-SLAM` must operate just below 200ms, or twice the speed of acquisition of the sensor. By contrast, CT-ICP runs at a higher-frequency than the 10Hz of the sensor. Which makes it compatible for online use. It can run even faster if we select the right parameters, as we show in section 3.5.3.1.

We will not discuss further the performance of `pyLiDAR-SLAM`. Instead, and it was designed for this purpose, we use it as a basis of comparison with other methods. In this section, we compare it essentially with CT-ICP, but in chapter 4, we use it to compare also deep and hybrid LiDAR odometries.

Below, we take a deeper dive into the performance of our principal contribution of this thesis: *CT-ICP*.

Driving datasets performance There has been a tremendous amount of effort invested LiDAR SLAM for driving scenarios from the LiDAR SLAM community. The space has been saturated with new methods since the release of the KITTI odometry benchmark. Yet we see that our method still performs better than other state-of-the-art methods on KITTI as demonstrated by table 2. One of the explications is that we tested, unlike the KITTI-odometry benchmark, on sequences where timestamp information is available. Our method, which has a fined distortion handling than other non-inertial methods, proved superior over other state-of-the-art methods of the time. And this will appear more clearly in section 3.5.2 when we isolate this contribution.

One thing limits the interpretability of the results on the KITTI and KITTI-360 dataset, that is the limit due to the nature of ground truth. Figure 3.31 shows the ground truth trajectory and the estimated slam poses for a section of KITTI's first sequence (`00`), and illustrates that the loss of GPS signal is compensated by a simple interpolation which adds significant noise on the results. This has an important impact on the score as the plot in figure 3.31 illustrates: every sharp peak of the graph corresponds to a ground truth problem, which results in a large error contribution, and there is a large gap between the mean error (0.44%) and the median (0.36%). Still, the relative result between methods can indicate better performance, though one must be careful about methods overfitting their parameter set on each sequence (this is for example the case of MULLS which proposes one set of parameters per sequence of the KITTI dataset).



(a) Ground truth trajectory for the `KITTI-00` sequence.

(b) Slam trajectory, for the same section. It is locally much faithful to the motion of the sensor.

Figure 3.31: An illustration of KITTI's ground truth problems. The ground truth (left) has errors of interpolation, probably due to the loss of GPS signal. This leads to artificial error terms in the evaluation of LiDAR odometries.

Mobile and Handheld datasets performance The mobile and handheld datasets presented are particularly challenging, due to high-frequency rotations, rotation around the z axis (for the NCLT dataset), etc... Yet CT-ICP does perform well in these challenging scenarios.

In the different tables, we presented the results for the method presented in our paper and the latest available version. We considerably improved the runtime of our method for the mobile and handheld scenarios, as well as the precision. Though the approach remains the same, a better implementation did lead to these significant improvements. And since the article, our method has gained robustness and efficiency, as demonstrated by the tables 3.4 and 3.5.

Why is our method this precise ? As shown above, our method reached at the time of its publication a state-of-the-art status. Yet, it is by essence a very simple method, based on a point-to-plane ICP algorithm, so the interesting question to ask candidly is *What makes it state-of-the-art ?* While we investigate in detail many components through ablation studies in section 3.5.2, we can provide here a high-level answer.

Our LiDAR Odometry is the most precise because we only make very precise point cloud insertions in the map. Two aspects allow this to happen. First, and this is our main contribution, is our elastic distortion method which is very precise, and ensures that the scan deforms itself very precisely onto the map. Second, is the structure of our map. Indeed, because we limit the number of points in each voxel, each voxel keeps in memory the oldest points observed. This means that the points of newer scans falling within a voxel are associated with the oldest points possible, which ensures long-term consistency.

The two combined allow each insertion of points to be very precise and thus prevent the addition of noise and pollution to the map. As we will see in section 3.5.3, herein lies also the

vulnerability of our method. If noise and pollution are indeed added to the map, then neighbor associations will be corrupted, which will deteriorate future registrations.

3.5.2 Understanding CT-ICP through ablation studies.

In the previous section, we proved our claim that we set a new state-of-the-art of LiDAR odometry. Here, we demonstrate other claims, through ablation studies which allow us to study components in isolation, and better understand the importance of parameters and design choices in the performance of the algorithm. First, we focus on our core contribution, namely the distortion handling, and study the performance of different alternatives in section 3.5.2.1. Then, in section 3.5.2.2 we study the impact of the principal parameters.

3.5.2.1 Frame distortion

In our related work, we presented 4 motion compensation algorithms, which, adding our novel method results in five methods to test. We summarize them in table 3.6. Note that we study only the frame distortion, so we keep each of the other parameters as defined in table 3.2. We selected one sequence of each dataset to perform this experiment and present the results in table 3.7.

Table 3.6: Different motion compensation strategies implemented in the ablation study.

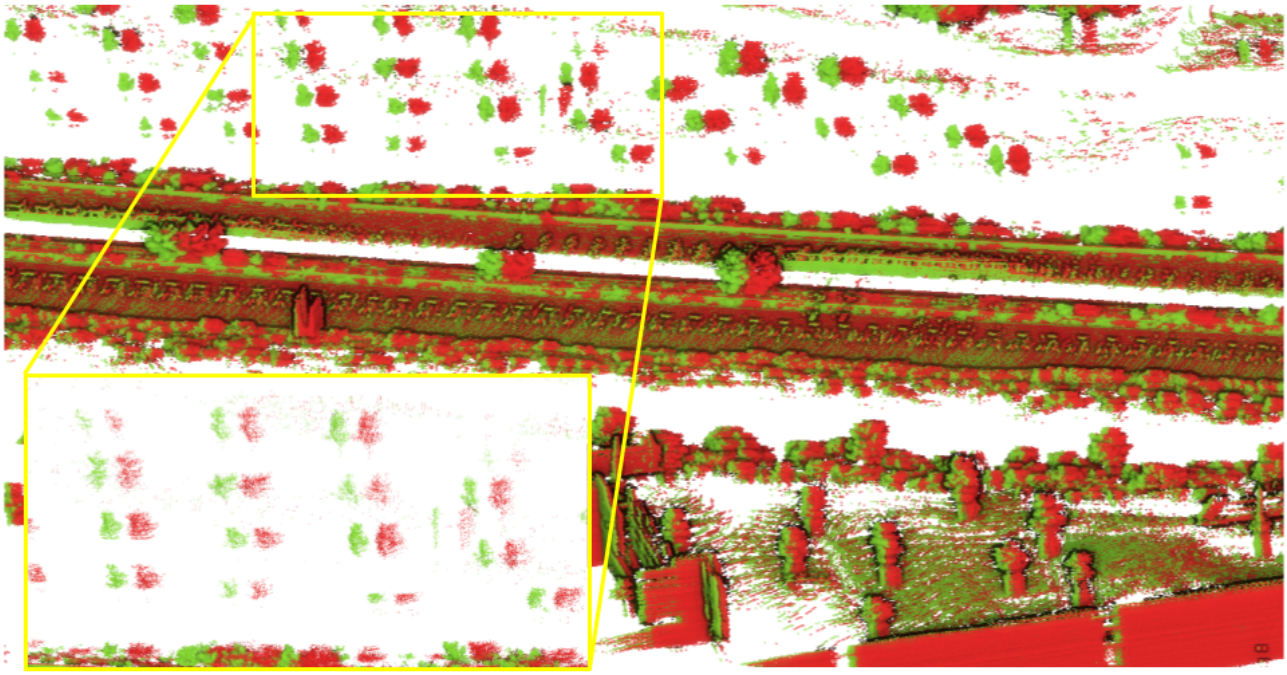
Strategy	Description
NONE	No distortion is applied to the point cloud before registration.
CV	The point cloud is distorted once before registration using the previous relative pose.
TWO STEPS	Similar to **CV** but the distortion is refined after registration.
ITERATIVE	Before each iteration of the ICP, the distortion is corrected with the best estimate.
CONTINUOUS	The ICP computes at each iteration the 12 degrees of freedom of an elastic registration.

First, these results demonstrate the importance of distortion handling for the performance of our LiDAR Odometry. Indeed, we see the large gap of the **RPE** between the strategy **NONE** and every other strategy, for each sequence. Without the correction of the distortion, points are added to the map at the wrong position, which adds noise and leads to less precise registrations.

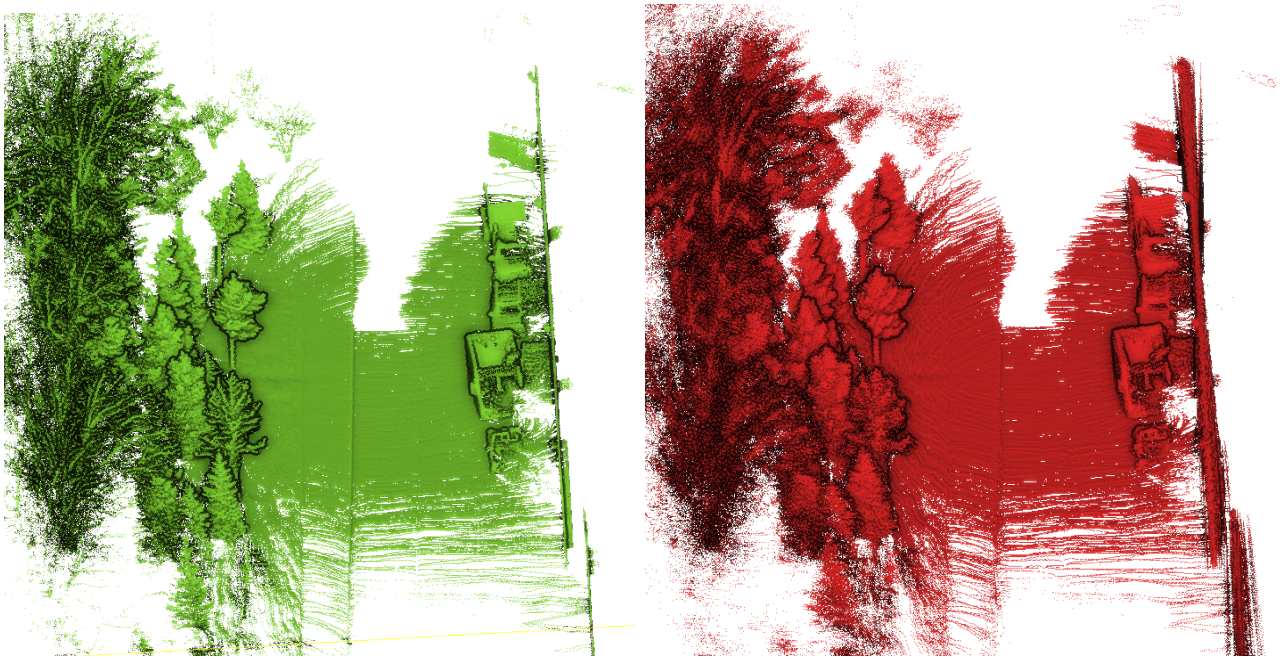
Secondly, we see that our novel **CONTINUOUS** method obtains consistently the best results, for each sequence. This proves the relevance of this new method, which is our principal contribution.

Comparing the different strategies, we can separate our analysis between driving scenarios on the one hand and mobile robotics with handheld scenarios on the other hand. For driving scenarios, the motion model is very consistent with the scenario, and the constant velocity model is a good predictor of motion. Thus, for driving sequences, the difference in performance between the different methods is less important than with the no-distortion scenario. Still, we can see that more complex methods like the **ITERATIVE** and **CONTINUOUS** methods show improvements over the **CV** of **TWO STEPS** methods.

Outside of driving scenarios, the gap between the **CONTINUOUS** approach and the others is even greater. Our mobile robotics and handheld datasets have irregular motions, with fast rotation (fast compared to the sensor acquisition frequency) which are particularly challenging to handle. In those cases, the **CONSTANT VELOCITY** model is not very well suited, and the errors in the undistortion of the scan lead to map pollution with noise, which in turn leads to unprecise registration. Figure 3.33 shows the map obtained for the **ITERATIVE** distortion strategy vs the **CONTINUOUS** strategy.



(a) Without correction (Red), the aggregated point cloud has a shift of 1-2m from the true point cloud (Green). (Dataset `KITTI-01`)



(b) Point cloud aggregated without correction (Red), and with correction (Green). If the distortion is not compensated, the aggregated point cloud is noisy. (Dataset `NCLT-2012-01-08`)

Figure 3.32: Illustrations of the impact of frame distortion on the quality of the aggregated point cloud. The green point cloud is the point cloud corrected for distortion and the red is without correction.

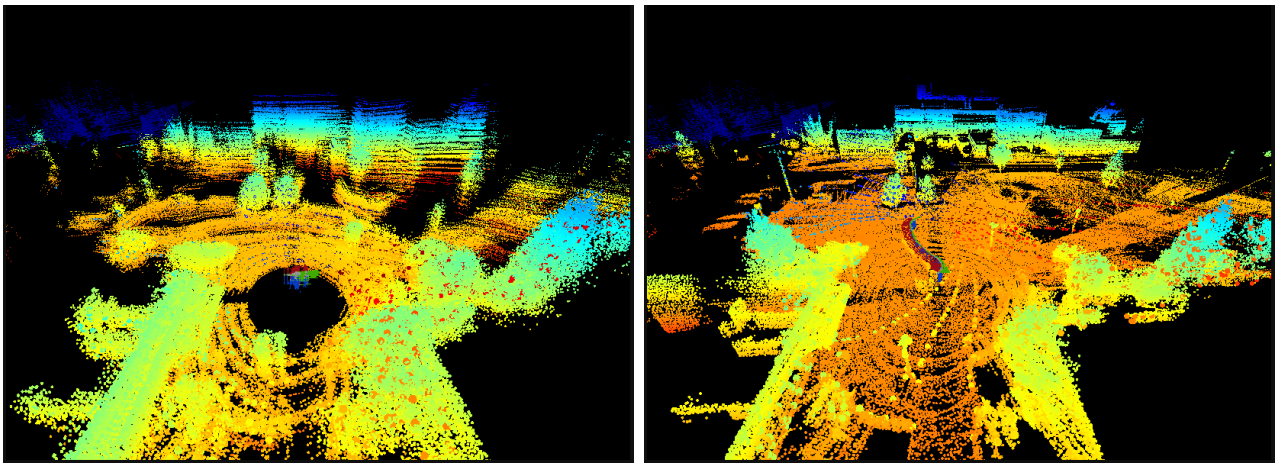


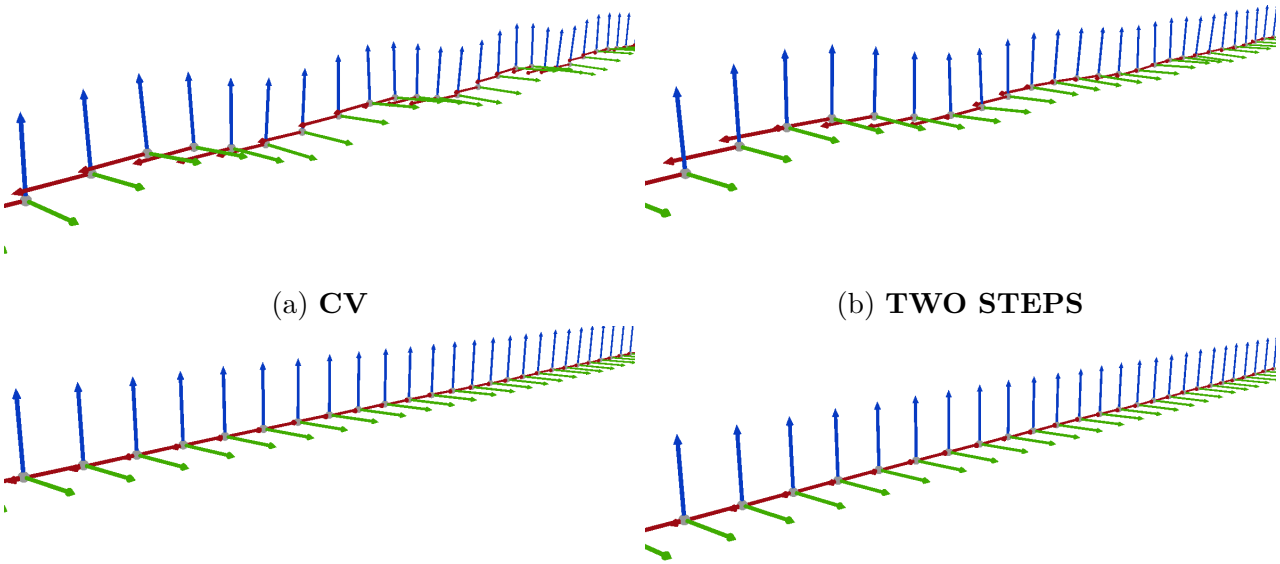
Figure 3.33: Map for the sequence `NCLT-2012-01-08` for the **ITERATIVE** (left) vs the **CONTINUOUS** (right) strategy. The incorrect handling of the distortion leads to poor insertions of scans in the map, which leads to the map pollution observed on the right. On the right, each frame inserted is properly distorted and keeps the map clean.

One question remains to be addressed: *Why does our method handle the distortion better than the others*? In figure 3.34, we present a segment of the trajectory for each distortion method. We see that for the **CV** and **TWO STEPS** the trajectories present an oscillating pattern. This pattern illustrates the compensation which typically occurs when errors in the distortion are not corrected: If an error is introduced that leads the ground to be elevated, the registration will compensate by predicting a pose that is lower than it should be. In certain conditions, this phenomenon, though it leads to imprecise maps, is not fatal. However, when the motion is too hard or violent, and the environment featureless this can degenerate, and lead to catastrophic SLAM failures (see `NCLT-2012-01-08` for the **TWO STEPS** method), or high levels of noise as shown in figure 3.33.

Table 3.7: **RPE** on a sequence (for multiple datasets) of the different motion distortion strategies. A ✗ symbol means that the sequence failed. The **RPE** is given at the relevant scale for each sequence.

Sequence	NONE	CV	TWO STEPS	ITERATIVE	CONTINUOUS
Driving:		*Consistent motion model*			
KITTI-raw - `00`	1.0398	0.59	0.65	0.44	**0.41**
KITTI-360 - `00`	0.86	0.70	0.91	0.64	**0.63**
KITTI-CARLA - `Town01`	0.38	0.036	0.035	0.039	**0.034**
Mobile Robotics and Handheld:		*Inconsistent or irregular motion model*			
NCLT - `2012-01-08`	4.2	2.3	✗	2.22	**1.5**
NCD - `exp_01`	3.4	2.01	2.1	1.87	**1.21**

In this section, we presented the experiments which demonstrated our principal contribution and clearly showed the relevance of our distortion-handling method. We need, however, to nuance the analyses presented in this section. While the high-level message: "*Proper Handling Distortion is important for precision*" remains true outside of this particular algorithm, the extent and the performance of each distortion strategy does depend on LiDAR Odometry method. In particular, our method while highly precise, because we keep a dense point cloud representation of the map, isn't robust to noise and map pollution. Other methods, notably the sparser method or probabilistic method can typically be more robust to map pollution.



(a) **CV** (b) **TWO STEPS**

(c) **ITERATIVE** (d) **CONTINUOUS**

Figure 3.34: Segment of the trajectory for the sequence `KITTI-00` obtained for the different motion compensation methods: **CV** (top left), **TWO STEPS** (top right), **ITERATIVE** (bottom left) and **CONTINUOUS** (bottom right).

3.5.2.2 Study of parameters impact

We now study the impact of the parameters on the performance of the algorithm. We selected the principal parameters, and though we could not be completely exhaustive, this study helps to give a clear picture to explain the performance of the algorithm.

Motion model First, we study the importance of the motion model for the different configurations proposed. As we treated the impact of the distortion handling in section 3.5.2.1, here we only focus on the initialization or the so-called motion prediction. We present the **RPE** for the different motion strategies in table 3.8.

The results show the importance of the motion model for the performance of the LiDAR Odometry, especially when considering driving scenarios. Indeed, if we look at the first three sequences, we see a clear degradation of the **RPE** between the **NI** (No Initialization) model and the **CV** (Constant Velocity) model (see figure 3.8). The mechanism for this degradation is very similar to the one presented in the previous section: bad registrations lead to map pollution which leads to the degradation of the precision of the SLAM.

The origin of this degradation however is different. When considering distortion errors, the registration algorithm does converge to the optimal solution of the problem and finds the best alignment between the distorted point cloud and the map. Here, the problem lies with the fact that the registration algorithm sometimes falls into a local minimum, because the initial pose was too far from the global minimum.

Note that the choice of the motion model for the Mobile Robotics and Handheld scenarios has less impact on the performance of our LiDAR Odometry. Their main reason is that the scale of the relative translation between two frames is much smaller for both scenarios than for the driving scenario. Thus, from this point of view, there are fewer differences between the two models, in contrast with road vehicles.

The problem with the constant velocity model, for such sequences, is that when a registration error occurs, it tends to propagate much quicker in the constant velocity model. The

NI however, in such cases is more tolerant to registration approximations, thanks to the error correction mechanism of our elastic registration.

Table 3.8: **RPE** on a selection of sequences (for multiple datasets) of multiple motion models. **NI** stands for no initialization and **CV** for constant velocity.

Sequence	Type	NI	CV
Driving:	*Consistent motion model*		
KITTI - 00	Driving	4.39	**0.41**
KITTI-360 - 00	Driving	1.73	**0.63**
KITTI-CARLA - Town01	Driving	0.063	**0.034**
Mobile Robotics and Handheld:	*Inconsistent or irregular motion model*		
NCLT - 2012-01-08	Mobile Robotics	**1.47**	**1.48**
NCD - exp_01	Handheld	**1.34**	**1.35**

Voxel Map resolution Another important aspect explaining the performance of our SLAM is our map. The sparse voxel map data structure allows for fast neighborhood computation and is perfectly adapted for map management tasks. As we set a limit to the maximum amount of points allowed per voxel to 20, the voxel resolution of the map also sets its sparsity, as figure 3.35 illustrates. Furthermore, the voxel size also defines the maximum radius to construct neighborhoods. Indeed, for efficiency, we limit the search radius to the 27 neighboring voxels, and a larger neighborhood considerably increases the runtime.

Thus, we now investigate how impactful is this parameter on the performance of our SLAM. We run our SLAM on a selection of sequences for different voxel sizes and present the results in table 3.9. As expected, this table shows that globally, the smaller the voxels the more precise the LiDAR Odometry.

There is, however a limit to this. As we can see, for a voxel size of 0.25m and when visiting only the 27 nearest voxels, most sequences have catastrophic failures. These failures are due to the radius of the neighborhood which is too short and prevents the scan from sliding as efficiently. When considering the 125 nearest neighbors, however, we find the highest level of precision for many sequences though there is a great cost for the runtime. This is a limit of our voxel hash map data structure. While we can also control the density of the point cloud by controlling the maximum number of points per voxel, given a voxel size, augmenting the size of the neighborhood leads to an exponentially increasing number of voxels to visit.

While we study in more detail the performance and bottlenecks of our method in section 3.5.3, it is interesting to note that the density of the point cloud of the map is not directly correlated with the performance of the algorithm.

Table 3.9: **RPE**(%) / Average Runtime (ms) of **CT-ICP** on a selection of datasets, for different voxel sizes of the map. For the **0.25m*** column, the neighborhood construction visits the $5^3 = 125$ neighboring voxels instead of the $3^3 = 27$ nearest. The mark ✗ means that a catastrophic failure happened.

Sequence / Voxel Size	0.25m	0.25m*	0.5m	0.8m	1.2m	1.6m
KITTI - 00	✗	**0.42**%/82ms	**0.44**%/41ms	0.46%/38ms	0.51%/34ms	0.61%/40ms
KITTI-360 - 00	✗	0.81%/88ms	0.65%/49ms	**0.64**%/36ms	0.62%/41ms	0.79%/42ms
KITTI-CARLA - Town01	✗	**0.013**%/50ms	0.027%/39ms	0.044%/29ms	0.09%/65ms	0.09%/35ms

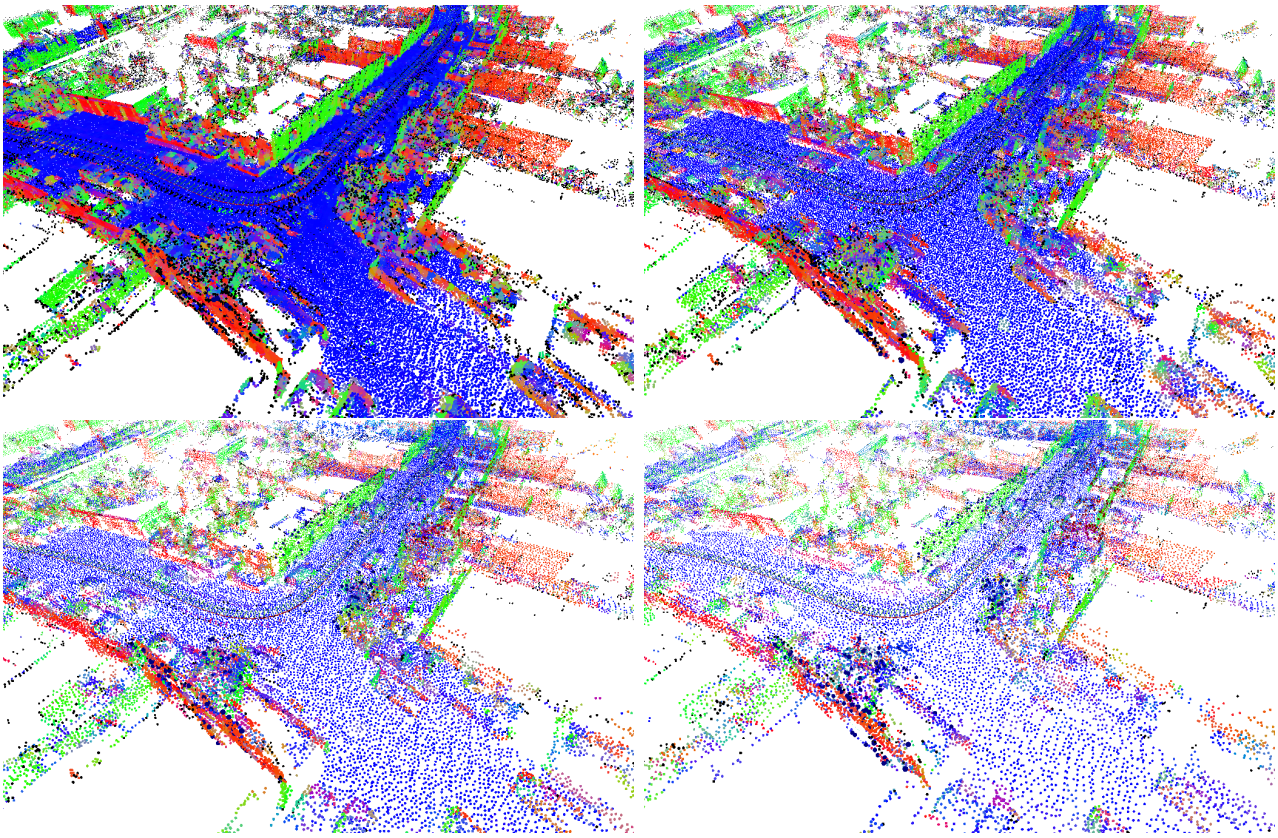


Figure 3.35: Point cloud of the map of **CT-ICP** running on the sequence `KITTI-00`, for different voxel resolutions: 0.5m (top left), 0.8m (top right), 1.2m (bottom left) and 1.6m (bottom right). The point cloud is colored by the direction of the normal at each point.

Sampling strategy Finally, we analyze the performance of the algorithm for different sampling strategies. We retain four strategies which we present in table 3.10. They are the principal strategies used in different odometries, and we presented some of them in the related work of section 3.2. Our method uses a double sampling approach, one as an initial step to reduce the computation burden, and another one for the key points selection step. We present in the table 3.11 the **RPE** as well as the runtime for different pairs of sampling strategies for a selection of sequences from our datasets.

Table 3.10: Description of the different sampling strategies used in this ablation study.

Sampling Strategy		**Abbrev**	Description
No Sampling		**NS**	No initial sampling is performed, all the points are added to the map.
Random:	N_{max}	**R**(N_{max})	Random selection of N_{max} points from the new frame.
Grid:	V_{grid}	**G**(V_{grid})	Grid selection with a voxel size of V_{grid} meters.
Regular Sample:	N_{max}	**RS**(N_{max})	Regular sampling of N_{max} points by sampling regularly each laser independently.

Looking at the results of table 3.11, the first thing to note is the importance of the sampling strategy. If no sampling is applied to the key points, the runtime is an order of magnitude (559ms instead of 51ms) longer than without this sampling. The impact on the performance of the frame sampling is much smaller, as the only operations applied to this point cloud are the transformation of the points once the optimal pose is computed and then the insertion in the map. It is furthermore mitigated by the fact that the map does introduce a "sampling" of its own, as it restricts the number of points inserted per voxel. Yet it is still noticeable if we

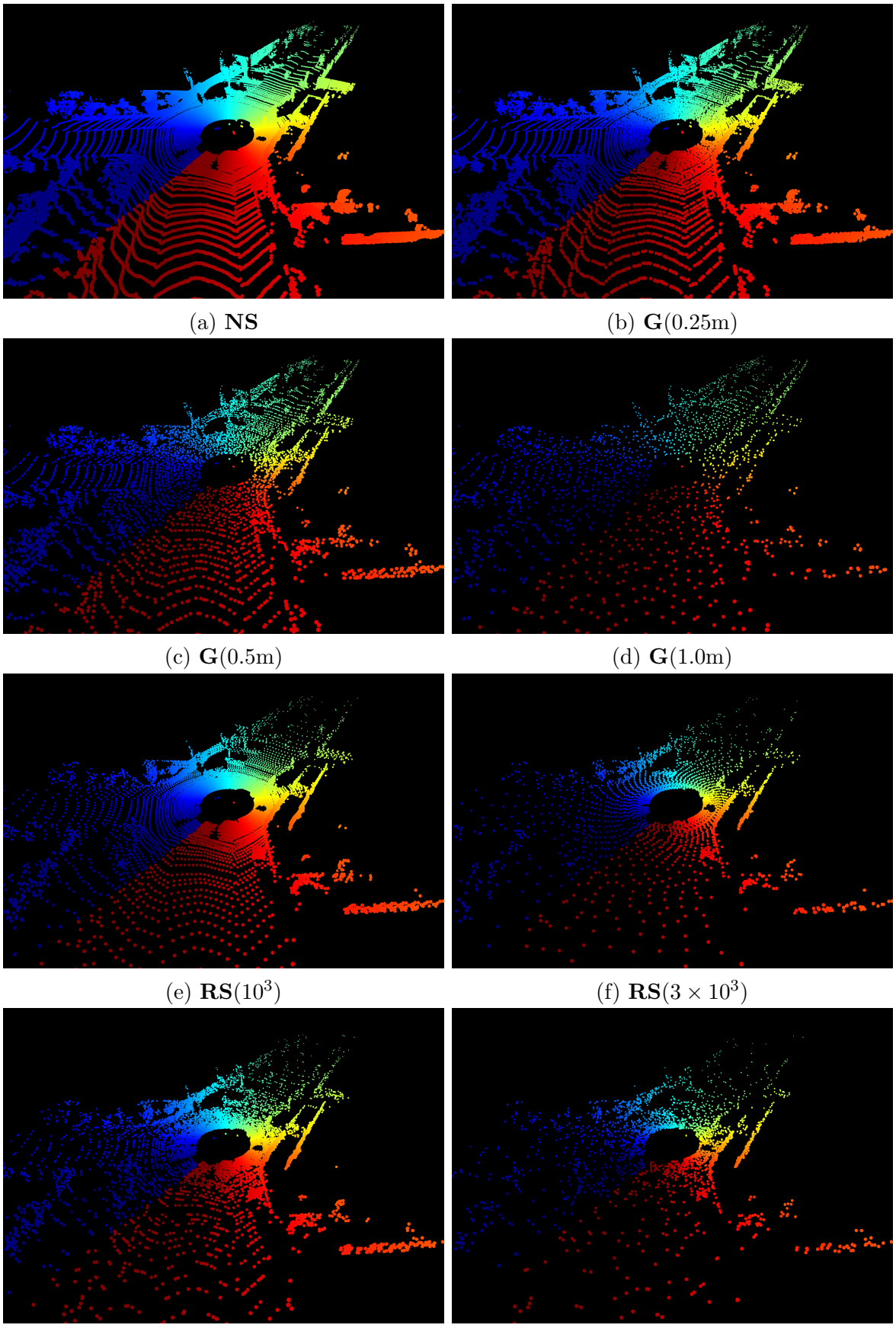
compare it with the grid sampling proposed in the default configuration of CT-ICP.

Table 3.11: Table presenting the mean **RPE**, runtime and the average number of points and key points sampled for multiple sampling strategies for both the initial preprocessing step and the key points selection. **FS** describes the frame sampling strategy, **KS** the key points strategy, **ΔT** the average runtime, and #**Points** /#**Key Points** respectively the average number of points and key points processed by the algorithm. We present the result on a selection of sequences from our datasets.

Sequence	FS	KS	RPE (%)/ΔT(ms)	#Points /#Key Points
No Sampling				
KITTI - 00	**NS**	**NS**	0.56%/559ms	$12.0 \times 10^4/12.0 \times 10^4$
KITTI - 00	**NS**	**G**(1.m)	0.56%/51ms	$12.0 \times 10^4/2.4 \times 10^3$
Default Grid Sampling				
KITTI - 00	**G**(0.5m)	**G**(1.m)	0.52%/37ms	$7.0 \times 10^3/2.4. \times 10^3$
KITTI-360 - 00	**G**(0.5m)	**G**(1.m)	0.79%/40ms	$8.0 \times 10^3/2.9 \times 10^3$
KITTI-CARLA - Town01	**G**(0.5m)	**G**(1.m)	0.032%/31ms	$9.1 \times 10^3/3.5 \times 10^3$
Finer Grid Sampling				
KITTI - 00	**G**(0.25m)	**G**(0.5m)	0.51%/52ms	$1.81 \times 10^4/7. \times 10^3$
KITTI-360 - 00	**G**(0.25m)	**G**(0.5m)	0.80%/55ms	$2.0 \times 10^4/7.9 \times 10^3$
KITTI-CARLA - Town01	**G**(0.25m)	**G**(0.5m)	0.032%/52ms	$2.0 \times 10^4/9.1 \times 10^3$
Coarser Grid Sampling				
KITTI - 00	**G**(1.m)	**G**(1.5m)	0.55%/27ms	$4.4 \times 10^3/2.0 \times 10^3$
KITTI-360 - 00	**G**(1.m)	**G**(1.5m)	0.83%/30ms	$2.8 \times 10^3/1.3 \times 10^3$
KITTI-CARLA - Town01	**G**(1.m)	**G**(1.5m)	0.08%/21ms	$3.5 \times 10^3/1.8 \times 10^3$
Regular Sampling				
KITTI - 00	**RS**(2.10^4)	**RS**(3.10^3)	0.63%/30ms	$2.10^4/3.10^3$
KITTI-360 - 00	**RS**(2.10^4)	**RS**(3.10^3)	0.99%/36ms	$2.10^4/3.10^3$
KITTI-CARLA - Town01	**RS**(2.10^4)	**RS**(3.10^3)	0.09%/29ms	$2.10^4/3.10^3$
Random Sampling				
KITTI - 00	**R**(2.10^4)	**RS**(2.10^4)	0.62%/33ms	$2.10^4/3.10^3$
KITTI-360 - 00	**R**(2.10^4)	**RS**(3.10^3)	1.0%/33ms	$2.10^4/3.10^3$
KITTI-CARLA - Town01	**R**(2.10^4)	**RS**(3.10^3)	0.09%/33ms	$2.10^4/3.10^3$

If we now compare the **Default Grid Sampling** group with the **Finer Grid Sampling**, we notice very few differences in terms of **RPE**. While the **Finer Grid Sampling** requires more effort (as evidenced by the increased runtime), it offers few improvements. Comparing the two to the **Coarser Grid Sampling**, we see here the expected drop in precision (and the gain in runtime). Given the number of points sampled, however, it is surprising that the drop is so low, especially when compared with the other sampling methods which all keep much more points.

Indeed, when focusing on the **Regular Sampling** or **Random Sampling**, it is clear that despite several points kept comparable to the **Finer Sampling**, the **RPE** shows a significant decrease compared to the different **Grid Sampling** methods. In figure 3.36 we show a point cloud sampled for different sampling strategies. One thing that appears clearly, is that the spatial repartition of points greatly differs from grid sampling methods on the one hand, and the two others on the other hand. More precisely, the **Random Sampling** and **Regular Sampling** keep much more points near the sensor compared to the different grid sampling which have systematically a homogeneous spatial repartition of points. This illustrates that the precision of our SLAM depends strongly on the association of points on the map with points distant from the sensor.



(a) **NS** (b) **G**(0.25m)

(c) **G**(0.5m) (d) **G**(1.0m)

(e) **RS**(10^3) (f) **RS**(3×10^3)

(g) **R**(10^3) (h) **R**(3×10^3)

Figure 3.36: Frame sampled with different strategies (see table 3.10).

3.5.3 Performance and Failure Analysis

So far, we only analyzed our algorithm through the lens of the desired precision of the trajectory produced by the SLAM. However, many design choices made in its design were made to minimize the runtime of our algorithm. In this final section of this chapter, we provide some elements to analyze the performance of CT-ICP.

First, in section 3.5.3.1, we present a precise runtime analysis of the algorithm. This will give us some idea of the bottleneck of the algorithm. We then study the performance/precision tradeoff available by changing the parameters in our algorithm CT-ICP. Finally, we focus on failure cases for our algorithm and discuss how to detect or avoid them.

3.5.3.1 Runtime Analysis

Many robotics projects use LiDAR SLAMs as a building block within a larger system. They require the SLAM not only to be performant but also to minimize its consumption of the platform resource. Our method, as we saw, depending on the parameters chosen can reach an average runtime as low as 33ms. However, we saw that for other parameter sets, the runtime can increase to 50ms or more. So in this section, we present in more detail what are the contributions of the different stages of our algorithm to the runtime, and what are the bottlenecks of our method.

Our LiDAR Odometry is separated into three main stages:

- The initialization, which accounts for the sampling, the motion initialization and the initial transformation of key points.
- The iterative registration algorithm.
- The update of the map. It is obtained by inserting new points in the point cloud.

We present in table 3.12 the average runtime for these three stages on two typical sequences, for the multithreaded and non-multithreaded version. For these experiments, we used the default parameters presented in table 3.2. Looking at the results, the first thing to notice is that the multithreaded version is capable in both cases to run online *ie* at a frequency below the LiDAR acquisition frequency of 10Hz. Furthermore, it could also perfectly run online if the sensor is set at a frequency of 20Hz. This is important, as for SLAM application, unless power consumption is an issue, setting the sensor at a frequency of 20Hz greatly reduces the problem of the distortion problem.

Table 3.12: Runtime (in ms) detailed by stage of the SLAM, averaged over the trajectory, for a selection of sequences of the dataset.

Sequence		**#Threads**	ΔT_{init}	ΔT_{icp}	ΔT_{map}	ΔT_{total}
KITTI-360	`00`	8	9ms	18ms	6ms	34ms
KITTI-360	`00`	1	9ms	66ms	5ms	80ms
NCLT	`2012-01-08`	8	6ms	32ms	5ms	44ms
NCLT	`2012-01-08`	1	5ms	124ms	6ms	133ms

Secondly, we can see that with the single-threaded version, the driving scenario (`KITTI-360`) still has a runtime below 100ms, while this is not the case for the mobile robotics one (`NCLT`). The difference can be explained by the different choices of parameters. Indeed, looking at the table 3.2, we see that the voxel size for the key points sampling is 0.8m. This means that per frame, the average number of key points (which is 2991 for this sequence) is much greater than for driving scenarios with a voxel size of 1.5m which is, on average, 1619 points. Thus every iteration of the ICP is slower on average than for driving scenarios.

Another important parameter is the maximum number of iterations. It is set to 20 for mobile robotics and 10 for driving scenarios. Yet, if we look at the average number of ICP iterations, it is 5.53 for the `KITTI-360` sequence and 4.14 for this `NCLT` sequence. This is because our convergence criterion often leads to early terminations of the registration algorithm. But lowering this number, especially for the `NCLT` sequence is dangerous, as it prevents proper handling of the edge cases where the motion model is a bad predictor of the actual motion. In those rare cases, the registration needs to make more effort.

Looking more closely at the contributions of each stage, we can see that the runtime is dominated by the registration algorithm. However, as we can see, the speedup for this stage is important, and we can leverage multithreading to accelerate greatly the runtime of this stage. For multithreaded applications, especially for the driving scenario, the contribution of the sampling and the map update is not negligible. Concerning the sampling, this is notable because our grid sampling is relatively costly, and necessitates the costly creation of a hash map of voxels filled with the points of the original frame. There are however alternative sets of parameters that can improve the runtime at the cost of lower precision. And we study this tradeoff in the next section.

3.5.3.2 Tradeoff between Runtime and Precision

There are many ways to optimize the runtime of our algorithm if we allow ourselves concessions in terms of precisions. We saw already in table 3.11 that there is a relation between the runtime and the precision of our SLAM. The relation is however deeply nontrivial, and in this section, we explore this relationship a bit.

Looking at the table 3.11, we saw that one of the lever we can use is the sampling of the point cloud, which removes the number of points and key points to process, to accelerate the SLAM. However, there are other levers to impact the runtime:

- *Sampling policy*: Keypoints sampling, Frame sampling
- *Map parameters*: Number of points per voxel, voxel size.
- *Convergence criterion*: Threshold on translation and rotation increments.

In table 3.13 we propose for a few sequences a choice of parameters to optimize the runtime of these sequences. First, we show for a selection of sequences the variation of both runtime and precision by varying the different levers independently. Then we provide for the selected sequences the optimal results in terms of runtime on the one hand, and precision on the other.

The methodology to establish this table was to search for the parameters which minimized the runtime without presenting any catastrophic failures. These choices of parameters are much less robust than the default parameters presented in table 3.2, which guarantees the lack of catastrophic failures on every one of the sequences tested. However, they provide some good intuitions on how to try and improve the runtime of these algorithms.

The first thing to notice is the large variety of potential runtime for the different sets of parameters. When wanting to optimize runtime instead of precision, our SLAM can run at $\sim$100Hz for driving scenarios, and 50Hz for mobile robotics, without critical failures. There is, in those cases a non-negligible drop in performance, but for many scenarios, notably if integrated with loop closure or GPS measurements, this could well be sufficient. Furthermore, this illustrates the versatility of our SLAM, which can not only reach a very high level of precision but also, by modifying the parameters, very high performances.

Secondly, by far the most important tool to reduce the runtime is the reduction of the convergence criterion. This criterion impacts the average number of iterations the registration has to perform. So for a criterion set to $0.005m$ and 0.05ř, the ICP performs an average of 9.5

Table 3.13: Table presenting **RPE** (%) and runtime (ms) for the different parameters used as a lever to influence the precision/performance tradeoff. We select three levers: the sampling (frame sampling **FS** and keypoints sampling **KS**), the map parameters (voxel size S^{vox}, number of points per voxel N_{pts}^{vox} and the min distance between points d_{pts}^{min}). For each set of parameters, we present the runtime ΔT and the Relative Pose Error **RPE** at the scale corresponding to each sequence.

Sequence		**FS/KS**	$\tau_{\mathbf{tr}}(\mathbf{m})/\tau_{\mathbf{rot}}(\check{\mathrm{r}})$	$S^{vox}(m)/N_{pts}^{vox}/d_{pts}^{min}$	**RPE** (%)/ $\mathbf{\Delta T}$(ms)
Using Convergence Criterion as leverage					
KITTI	00	**G**(0.5)/**G**(1.5)	0.005m/0.05°	1m/20/0.15m	0.4543%/41ms
KITTI	00	**G**(0.5)/**G**(1.5)	0.01m/0.1°	1m/20/0.15m	0.4586%/32ms
KITTI	00	**G**(0.5)/**G**(1.5)	0.03m/0.3°	1m/20/0.15m	**0.4520**%/21ms
KITTI	00	**G**(0.5)/**G**(1.5)	0.05m/0.5°	1m/20/0.15m	0.4640%/**19ms**
Using Sampling as leverage					
NCLT	2012-01-08	**G**(0.3)/**G**(0.8)	0.01m/0.1°	0.8m/30/0.15m	1.19%/55ms
NCLT	2012-01-08	**G**(0.4)/**G**(1.2)	0.01m/0.1°	0.8m/30/0.15m	**1.02**%/49ms
NCLT	2012-01-08	**G**(0.5)/**G**(1.5)	0.01m/0.1°	0.8m/30/0.15m	1.19%/44ms
NCLT	2012-01-08	**RS**(10k)/**RS**(2k)	0.01m/0.1°	0.8m/30/0.15m	1.17%/**38ms**
NCLT	2012-01-08	**RS**(20k)/**RS**(3k)	0.01m/0.1°	0.8m/30/0.15m	**1.02**%/48ms
NCLT	2012-01-08	**RS**(40k)/**RS**(4k)	0.01m/0.1°	0.8m/30/0.15m	**1.02**%/76ms
Using map parameters as leverage					
NCLT	2012-01-08	**G**(0.4)/**G**(1.2)	0.01m/0.1°	0.8m/40/0.10m	**1.02**%/56ms
NCLT	2012-01-08	**G**(0.4)/**G**(1.2)	0.01/0.1	1m/20/0.15m	1.19%/45ms
NCLT	2012-01-08	**G**(0.4)/**G**(1.2)	0.01/0.1	1.5m/20/0.20m	1.18%/43ms
Parameters for fastest execution without critical failures					
KITTI	00	**RS**(10k)/**RS**(1.5k)	0.03m/0.3°	1.5m/20/0.15m	0.75%/**10ms**
KITTI-360	00	**RS**(8k)/**RS**(1.5k)	0.03/0.3	1.0m/20/0.15m	0.91%/**11ms**
NCLT	2012-01-08	**RS**(8k)/**RS**(1.5k)	0.03/0.3	1.m/20/0.15m	1.17%/**18ms**
Parameters optimizing for precision					
KITTI	00	**G**(0.4)/**G**(1.2)	0.01m/0.1°	0.8m/40/0.10m	**0.449**%/56ms
NCLT	2012-01-08	**G**(0.4)/**G**(1.2)	0.01/0.1	0.8m/20/0.15m	**1.02**%/56ms

iterations against 1.6 when setting it to $0.05m$ and $0.5°$. Another way to control the average number of iterations is by thresholding the maximum number of iterations. In this work, we set a limit rather high as it allows us to better handle edge cases, and control the runtime with this convergence criterion.

The second most powerful lever to control the runtime is sampling. The grid sampling **G** is the most costly sampling method. A sampling **G**(0.3m)/**G**(0.8m) adds in average 9ms per frame due to the sampling, against 1.2ms for a sampling **RS**(20k)/**RS**(3k). The sampling also influences the time to update the map, as a finer frame sampling leads to more points to insert into the map.

The final lever is less impactful, though it becomes important when looking to find extremely efficient performances. To reduce the time of each iteration of the ICP, we can limit the number of points in each voxel to visit, and we leveraged this to reach 100Hz performance in driving scenarios. The two ways to control them are by setting the threshold on the number of points per voxel, or by modifying the minimum distance between two points in each voxel. The risk, however, is to lose too much information, which can lead to imprecise neighborhoods and increase registration errors.

On a concluding note, one can notice again that the relationship between performance and the choice of parameters is not only nontrivial but also very much nonlinear. One thing that is also hidden from the table, is that a set of parameters can lead to catastrophic failures in a specific sequence and not in others. So to clarify this, in the next section, we focus on failures and the limitations of our algorithm.

3.5.3.3 Examples of failure

In this chapter, we mentioned on multiple occasions the mechanisms for our SLAM to fail. We focus on this aspect in this section and present different failure cases of our SLAM algorithm. There are three principal reasons for our SLAM to fail: registration errors, map pollution and complicated environments. The three are of course intrinsically linked, as registration errors lead to map pollution, complicated environments to map pollution, etc...

In figure 3.37 we present multiple cases of registration failures, with explanations for these failures. We use the following weak criteria to catch some failures:

- *The relative translation greater than 3m.*
- *The relative rotation greater than 3°.*
- *The number of key points smaller than 100.*

These criteria allow us to catch many failures. In figure 3.37 we show the map before a failure is detected. These illustrate multiple scenarios where the SLAM fails. The first one is a long corridor which is a perfect example of a tunnel effect: for the default parameters, our SLAM fails due to the ill-constrained problem. For our SLAM to reconstruct this sequence, we need to use a set of parameters that allows it to catch smaller discriminative details. This means augmenting the resolution of the map and augmenting the number of key points by lowering the grid size.

The second is typical of a featureless flat environment. The geometry is even more ill-constrained than the corridor. Our SLAM has a critical failure on this sequence.

The third case is a typical case of map pollution. Here, the trail of points captures the observer carrying the handheld platform (from the HILTI dataset). These points add noise, which adds noise to the map. This noise, along with the difficulty to navigate in small confined spaces (See the third picture) lead to critical failures of the SLAM for this sequence.

Luckily, in all the cases above the error manifest itself by an ill-constrained problem that diverges significantly and thus can be caught with our criteria. However, very often failures are not detected which is very problematic, as it requires manual supervision to assess the quality of a SLAM when no ground truth information is present. We give in figure 3.38 examples of failures that were not detected immediately. As we can see, bad insertions of even a single frame can create double-wall effects. The following frames then are often registered onto this wrongly inserted point cloud creating a cascade of bad insertions which can be seen on the right image. This lack of error detection is a clear shortcoming of our method.

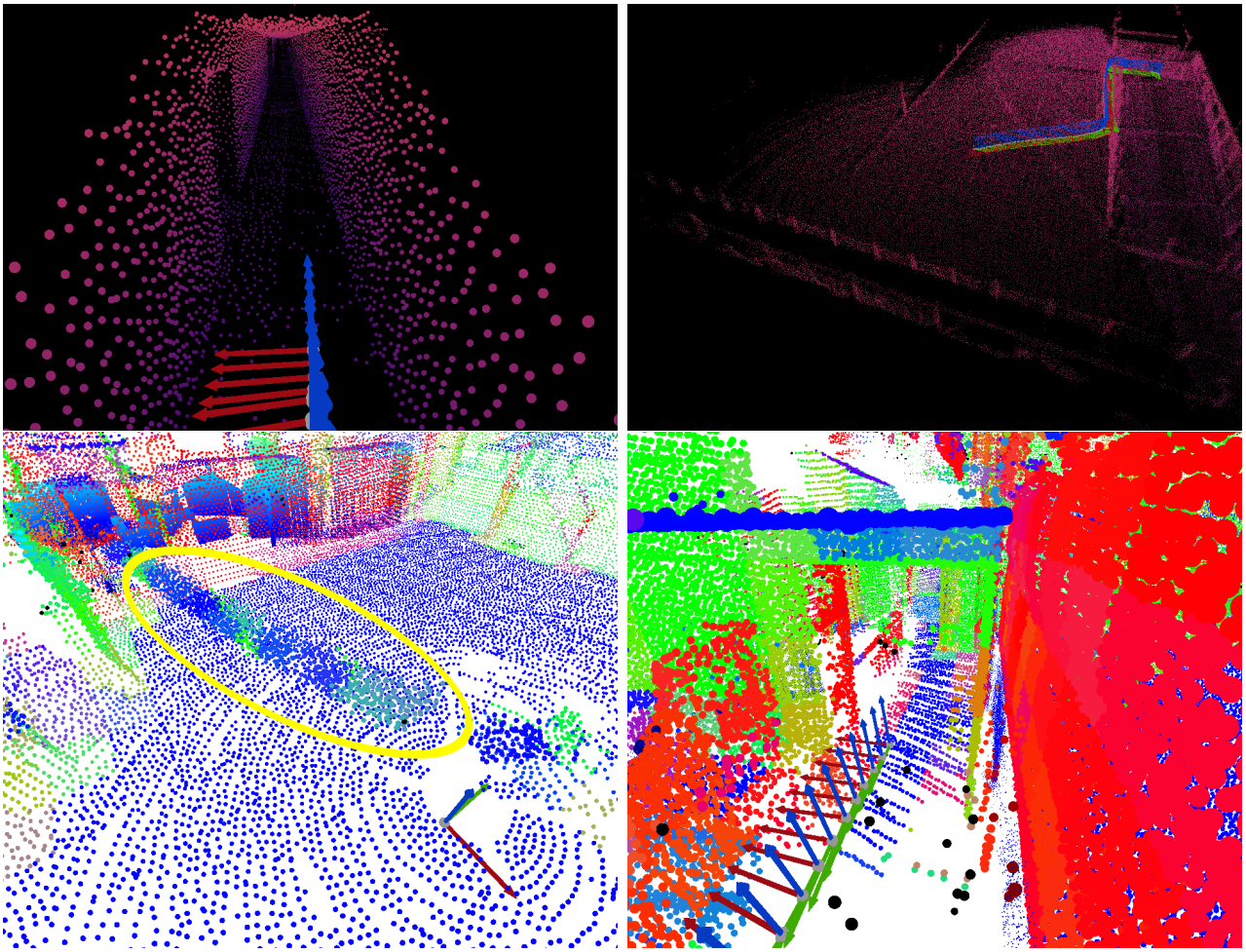
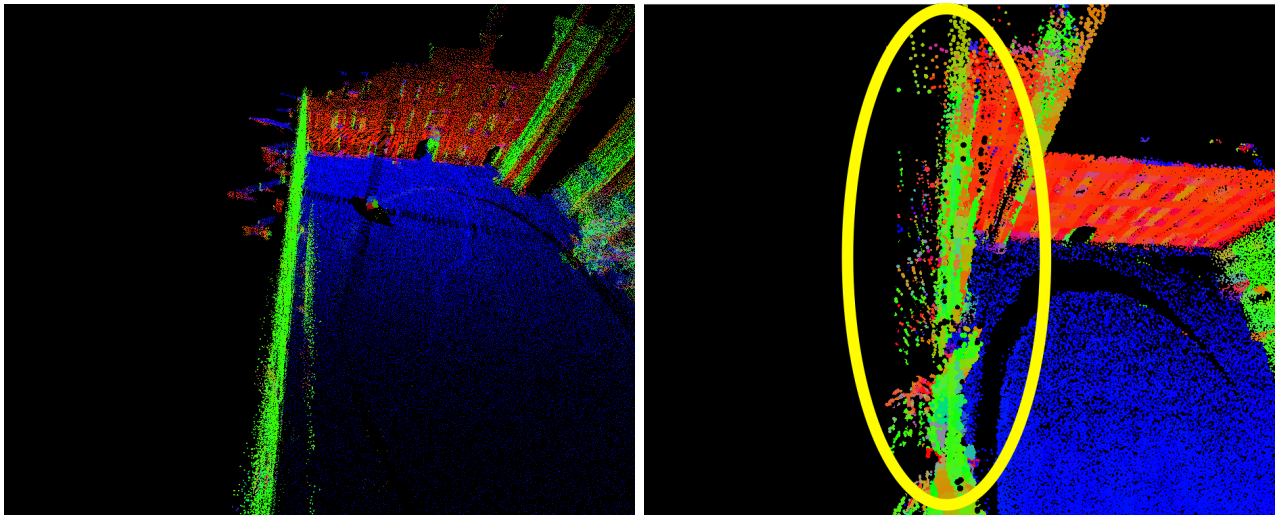
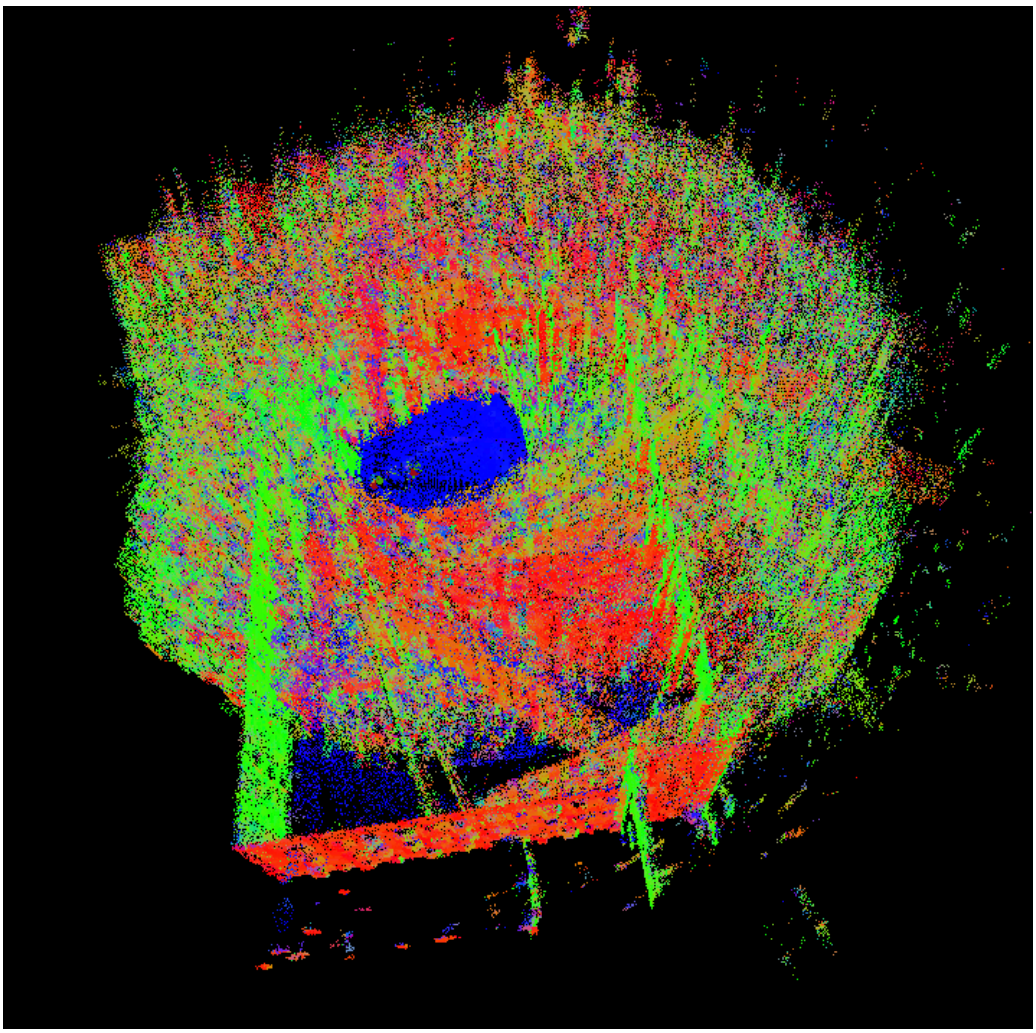


Figure 3.37: Situations leading to catastrophic failures caught by our criterion. From top to bottom, left to right: **1:** Failure due to the tunnel effect of a corridor. **2:** Error due to a featureless environment (empty parking lot). **3:** Error caused by map pollution (trail of the point cloud from the handheld sensor). **4:** Error caused due to very small interior spaces.



(a) Double wall effect (left) caused by a single bad insertion, can lead to a cumulation of bad insertions (right).



(b) This can lead to such catastrophic failures, which are not currently detected.

Figure 3.38: Example of failures caused by bad insertions, resulting in a catastrophic failure.

3.6 Conclusion

In this chapter, we presented the principal contribution of our thesis: the state-of-the-art real-time LiDAR odometry method **CT-ICP**. After a deep analysis of the related work, and the description of both the methods **pyLiDAR-SLAM** and **CT-ICP**, we proposed a detailed experimental analysis of CT-ICP which help illustrate its strengths and performance.

As mentioned above, the work `pyLiDAR-SLAM` is part of the publication *What's In My LiDAR Odometry Toolbox ?* [24], accepted at the international conference IROS 2021. CT-ICP has also been published and accepted for the conference ICRA 2022, where it received a nomination for the *Oustanding Paper Award*. In this chapter, however, we went deeper in our analysis, and evaluated our method on a wider range of datasets. This work will be part of a journal publication in preparation.

We also presented in the last section 3.5.3.3 challenges common to all LiDAR Odometries. When starting the thesis, a field started to emerge, promising to address these challenges with Deep Learning approaches. Indeed, as we saw, it is difficult to establish the failure or success of LiDAR Odometries. These methods arrive with a original perspective: if it is difficult, why not learn how to do it? We investigate these approaches in more details in chapter 4.

Chapter 4

Deep and Hybrid LiDAR odometries

Contents

Résumé

En parallèle des progrès réalisés par les odométries LiDAR classiques ces dernières années (que nous avons présentés dans le chapitre 3), des méthodes de Deep Learning essayant de résoudre plusieurs types de problèmes géométriques sont apparues. En effet, le Deep Learning a été proposé comme outil pour résoudre un large éventail de tâches telles que la localisation, avec des réseaux de type PoseNet [54], des méthodes non-supervisées de prédictions d'images de profondeur (à partir de flux d'images monoculaires avec SfM-Learner[140]), pour de l'appariement de points d'intérêts[104], etc…De plus, de nombreuses représentations Deep pour nuages de points sont apparues avec des architectures telles que PointNet [89], KPConv [116] ou utilisant des structures de voxel éparses efficaces [22].

Ces méthodes illustrent bien qu'il existe un potentiel d'utilisation du Deep Learning pour résoudre, ou aider à résoudre le problème d'odométrie LiDAR. Cherchant à exploiter ce potentiel, diverses méthodes de type "boîte noire" sont notamment apparues, prédisant directement la trajectoire du LiDAR. Ces méthodes cherchent à résoudre la complexité du SLAM LiDAR en "apprenant" à prédire le mouvement du capteur, en comparant deux nuages de points successifs fournis par le LiDAR. Elles visent donc à répondre à plusieurs problèmes que nous avons observé dans le précédent chapitre: le coté arbitraire du choix des paramètres des algorithmes de SLAM classiques, l'adaptabilité aux différentes échelles d'environnement et de mouvement, etc…Dans ce chapitre, nous évaluons ces méthodes de type boîte noire plus en détail, et nous regardons notamment si elles peuvent permettre d'améliorer les odométries classiques. Pour le faire, nous produisons une analyse comparative approfondie entre plusieurs classes d'odométries LiDAR classiques, hybrides et pure Deep Learning.

Le reste de ce chapitre est organisé de la manière suivante: tout d'abord nous présentons l'état de l'art dans la section 4.2, puis dans la section 4.3 nous présentons notre analyse comparative étendue.

Les contributions présentées dans ce chapitre sont les suivantes:

- Une analyse comparative de méthodes classiques, deep et hybrides d'odométrie LiDAR.
- Une implémentation publique de méthodes deep et hybrides d'odométries LiDAR, intégrées au projet `pyLiDAR-SLAM`.

4.1 Introduction

Along with the progress in classical LiDAR odometry which we presented in chapter 3, recent years have seen the emergence of deep learning methods attempting to solve geometric problems. Deep learning methods have been proposed for a large variety of such tasks such as localization with PoseNet [54], unsupervised depth prediction from monocular images with SfM-Learner [140], feature matching [104], etc ... In parallel, there has been rapid progress in recent years in deep learning representations for point cloud with neural architectures such as PointNet [89], KPConv [116] or efficient sparse voxel grid convolutions [22].

Thus there is a lot of potential for integrating Deep Learning methods to solve, or at least, help solve LiDAR odometries. Notably, end-to-end deep learning methods have appeared, which directly predict the trajectory of the sensor. These methods promise to answer multiple problems observed in the previous chapter, notably the arbitrary choice of parameters of classical SLAM pipelines, their capacity to adapt to different environments or scenarios, etc…The goal of this chapter is to evaluate the relevance of these methods, against or combined with classical LiDAR odometry methods.

The rest of this chapter is organized as follows: first, we present the related work in section 4.2, then in section 4.3, we present our extended comparative analysis of end-to-end methods.

4.2 Related Work

The use of deep learning to solve geometric tasks has been growing for years. The intrinsic relevance of learning approaches resides in their capacities to learn from example, from real data collected. As we saw in chapter 3, classical SLAM(s) are sophisticated machines often with a large number of parameters, carefully tuned and non-trivially adaptable to different environments. Deep Learning promises to remove this problem by providing algorithms that can be trained in an environment. This removes the complexity of hand-tuning parameters for specific usage but offers different challenges.

First, deep learning methods need to be trained on a large volume of data. Thus, in contrast with classical methods, they typically require an initial effort of collecting data and then training on it, before a model can be used for inference. For supervised methods, the data collected also needs to be labeled. So if a task requires a large volume of data to obtain the desired levels of precision, this is typically a very costly operation. In the context of SLAM the collection of labeled data, though less painful than semantic labeling, requires some additional effort. Benchmarks such as [99] compute accurate ground truths by registering scans in ground truth maps obtained from scanning with high-precision survey-grade LiDAR sensors. The benchmark KITTI [40] uses GPS measurements to provide the ground truth. The first method is costly in time and equipment, and the second cannot work in GPS-denied environments and requires costly material to obtain comparable levels of precision.

The second challenge is the capacity of deep learning methods to generalize on data outside of the training dataset. The real world is extremely diverse, and to work properly the inference data should match as closely as possible the training data. Note that for SLAM applications to work properly, this can be challenging, as one of the goals of SLAM is to explore new and unseen environments. So to hope of generalizing to a wide range of environments, massive amounts of data are typically required.

Finally, in the context of LiDAR odometry, an additional problem is the nature of the data. In contrast with images, and up until recently, point cloud representation was lacking. Methods such as PointNet [89] unlocked many learning applications on point cloud data, and later convolutional operators such as Kpconv [116] or Sparse Tensor Convolutions [22] provided the point cloud's equivalent of images' convolutional operator which was lacking until now. Still,

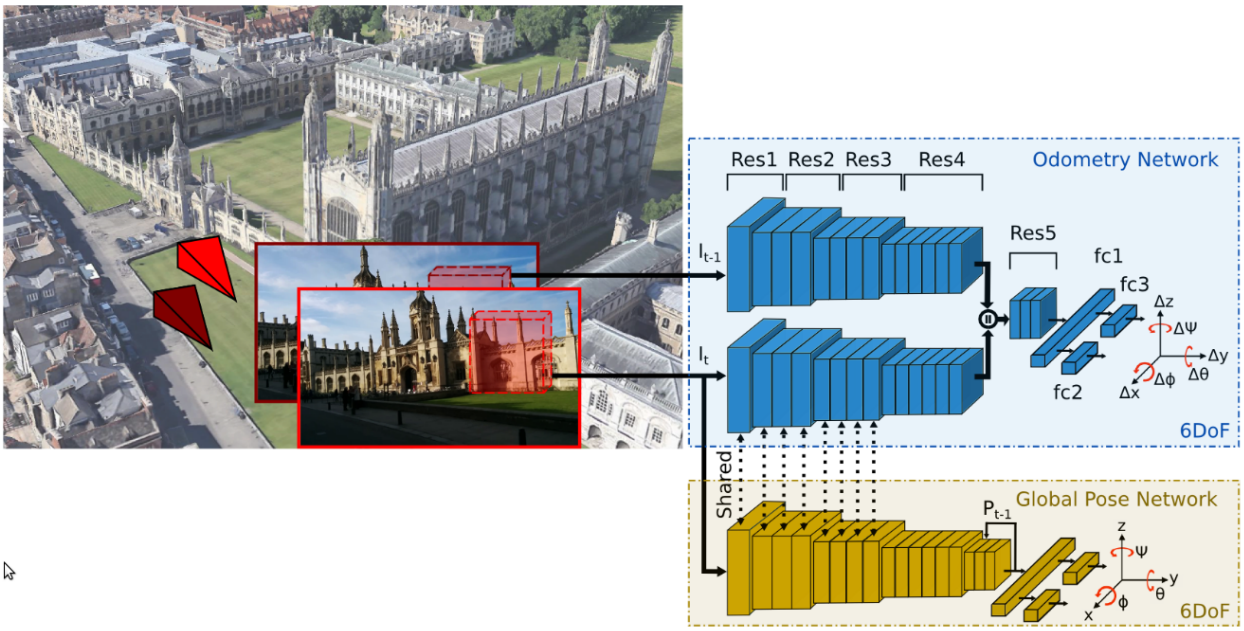


Figure 4.1: Architecture and description of PoseNet for image localization. In this approach from [119], which builds on the original PoseNet [54], uses one network to predict relative poses between two frames, and another one to predict absolute poses for image localization.

even with high-end GPUs many of such representations are too slow and costly for many real-time applications. So few LiDAR odometry methods have incorporated such representations in their pipeline so far.

Despite these challenges, different approaches to addressing LiDAR odometry with deep learning have been proposed, and we present the most relevant in this section. The remainder of the section is organized as follows: in section 4.2.1 we present end-to-end LiDAR odometries proposed, then in section 4.2.2 hybrid methods which integrate deep learning within a classical pipeline.

4.2.1 PoseNet-based and other End-to-End LiDAR odometries

A first class of method proposes end-to-end deep LiDAR odometries, *ie* they propose a neural network architecture which takes as input LiDAR frames and directly regresses as outputs pose parameters. All these methods typically use a similar prototypical neural architecture but typically differ both in architectural details and most importantly in their training methods. This prototypical architecture was initially proposed by the image-based localization method PoseNet [54] and is composed of a feature extraction module (either constructed using a series of convolutional layers, or deep point cloud feature extraction modules) that transforms a tensor into a feature vector, and a Multi-Layer Perceptron (MLP) head which regresses pose parameters from the feature vector as presented in figure 4.1. In the remainder of this work, we also name PoseNet this prototypical neural architecture and thus consider all the neural architectures derived from this prototype as flavors of the original PoseNet.

In the original paper, PoseNet [54], the neural network was trained to output absolute poses based on specific images. The goal is to train the network on a given site, with a sequence of images of these sites labeled with ground truth poses, and to be able to estimate for new images of this same site the pose of each image. Thus the task solved here is Absolute Pose Regression (APR). This method has generated a lot of interest, and many work have followed

its tracks: [119, 18, 77, 129, 1]. We show in figure 4.1 a description of the "PoseNet" for image localization from [119].

For odometry and SLAM, however, the goal is to navigate in new and unknown environments. To this end, the same architectures are trained to predict relative poses, *ie* to solve the task of Relative Pose Regression (RPR). The typical approach to this end is to train the same network architecture to predict for pairs of consecutive frames (images or LiDAR frames projected with a spherical projection) the RPR.

The first LiDAR method to use such an end-to-end approach is Lo-Net [65], which proposes a hybrid method (see section 4.2.2) with notably a PoseNet module predicting RPR at a high frequency. This approach is supervised and thus requires ground truth for the relative poses of each consecutive frame of the dataset. The method was only trained on the sequences of the KITTI odometry benchmark [40] containing the ground truth.

DeepLO [19] was proposed the following year, with for the first time an unsupervised learning method to train PoseNet for RPR. In their method, presented in figure 4.2, the parameters of the PoseNet network are trained by differentiating a point-to-plane loss, which is constructed using the predicted pose. This loss (which is essentially the energy minimized by LiDAR odometry) is small near the optimum and thus provides them with a supervision signal which allowed them to reach precision levels on KITTI, with notably much better generalization than Lo-Net. Following DeepLO, [20] proposed an additional term to the loss to improve the precision of the odometry, proposing an additional plane-to-plane loss to the loss proposed by DeepLO.

In this work, as we implement a version very much similar to DeepLO [19], we present in much more detail the method in section 4.3.1.

End-to-end methods released since the publication of our work Since the publication of our work, new end-to-end approaches have been proposed. **UnDeepLIO** [118] have proposed a LiDAR-Inertial end-to-end odometry, which fuses DeepLO's PoseNet with the initial predictions of poses from IMU signal constructed using LSTM [46] recurrent neural networks. This work is trained with a similar unsupervised to DeepLO. Selfvoxelo [128] uses 3D convolutions, and thus does not rely on the spherical image projection of DeepLO. They propose a complex loss construction, which notably uses the standard ICP as a more precise pose supervision signal. Another work [79] uses PointNet++[90] as a siamese network as an alternate feature extraction module, while keeping the MLP head. Finally, **PWCLO-Net** [122], uses an iterative three stages warp-refinement approach greatly inspired by 2D flow networks. This work, which has the more complex architecture is supervised with the ground truth poses.

These works have iterated over the previous end-to-end approaches which are considered in this chapter for our analysis. Using more sophisticated feature extraction, more adapted for point clouds, some do present improvements both in terms of generalization [128, 122] and capacity to learn, over the simpler DeepLO. However, as will be detailed in section 4.4, they do not change the conclusions of our analysis.

4.2.2 Hybrid Methods

As we will see in section 4.3.2, by themselves, and though promising, end-to-end approaches, (*ie* the approaches that directly regress poses presented in the previous section) compare poorly to state-of-the-art classical odometry methods. However, they can be integrated as components of hybrid LiDAR odometry pipelines. This is the first type of hybrid method and those which are the focus of this work.

More precisely, we will use in this work the methods [65, 80] as a basis for comparing our approach. For both of these methods, a hybrid LiDAR odometry is constructed by using PoseNet end-to-end odometry (presented in the previous section) to initialize a more costly mapping procedure.

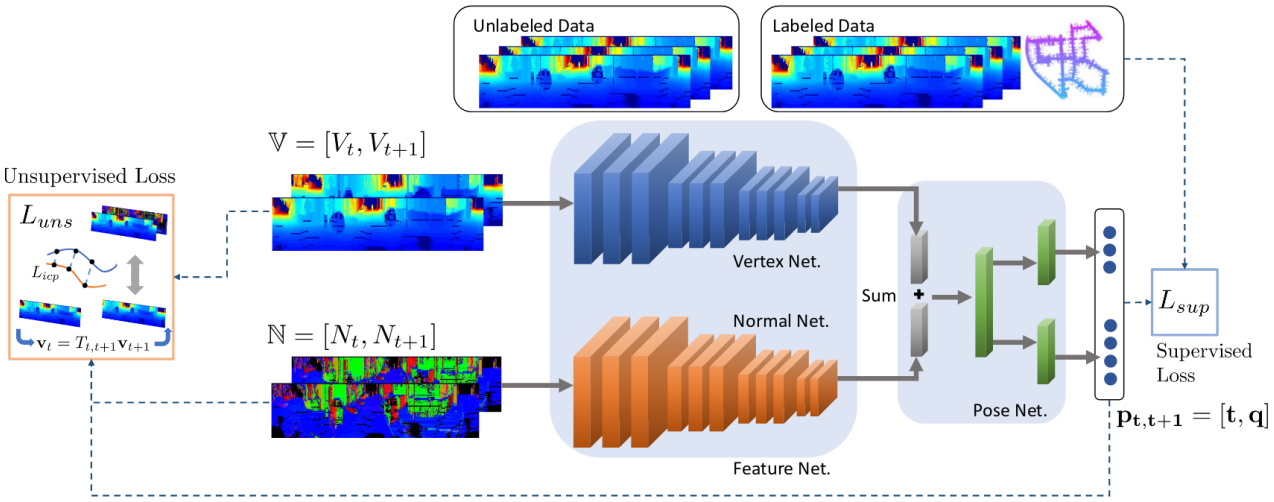


Figure 4.2: DeepLO method as described in the original article [19].

Detailing the first category a bit further, Lo-Net [65] (see figure 4.3) proposes its mapping procedure, which is a classical point-to-plane frame-to-model registration algorithm. Additionally, it uses the normals computed for each frame on the frame's spherical image as described in section 3.3.3. On the other hand, [80] directly uses LOAM [135]'s mapping procedure.

Essentially, both of these approaches replace the frame-to-frame registration of an algorithm like LOAM [135], with an estimate provided by a PoseNet architecture. Both methods evaluate the precision of their end-to-end odometries by comparing them to frame-to-frame ICP-based alignment. And appear to have better performance. We mitigate these conclusions in section 4.3. Broadly, we consider that to properly evaluate these methods, they need to be treated as initialization strategies (constant velocity model, elevation images and frame-to-frame registration) and see if they provide some real benefits within SLAM pipelines. These different strategies were presented in the section 3.2.2.1 above.

We evaluate these different deep and hybrid LiDAR odometry methods in detail in section 4.3.2.

There exists a second category of Hybrid methods, which operate on pairs of frames (like end-to-end approaches presented in the previous section). However, these methods do not regress pose parameters in an end-to-end fashion. Instead, they use DeepLearning to construct an intermediary step, which is fused with a frame-to-frame classical geometric method. approaches but does share the pitfalls observed with these methods which we will describe below. Some, for example, learn how to perform point cloud association [67, 117], or learn feature description, and then use an SVD to predict the motion. We show in figure 4.4 a description of the hybrid method DMLO[67]. Differently, LodoNet [137] uses an image-based feature matching technique on spherical images, before regressing the pose using an MLP.
These approaches are beyond the scope of this work, where we focus on the first category of method.

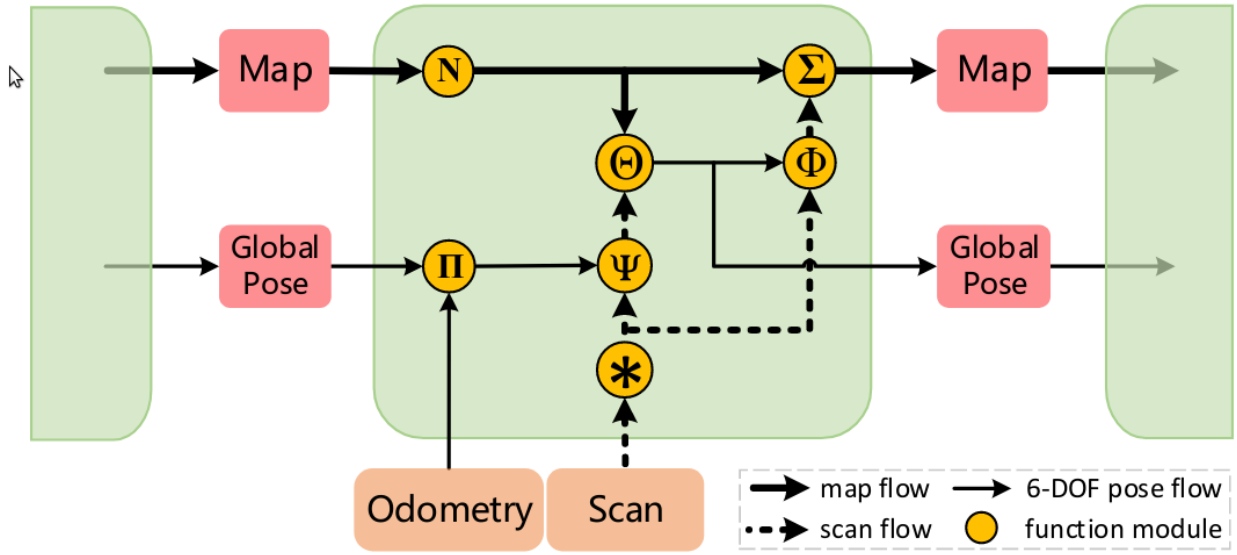


Figure 4.3: Description of LoNet [66] method, a Hybrid method which combines an end-to-end deep LiDAR odometry, with a classical frame-to-model mapping approach. The operation $\mathbf{\Pi}$ fuses the previous global pose with the Deep Odometry for initialization. Then, the operation $\mathbf{\Theta}$ iteratively refines the pose using a standard point-to-plane based ICP (see chapter 3).

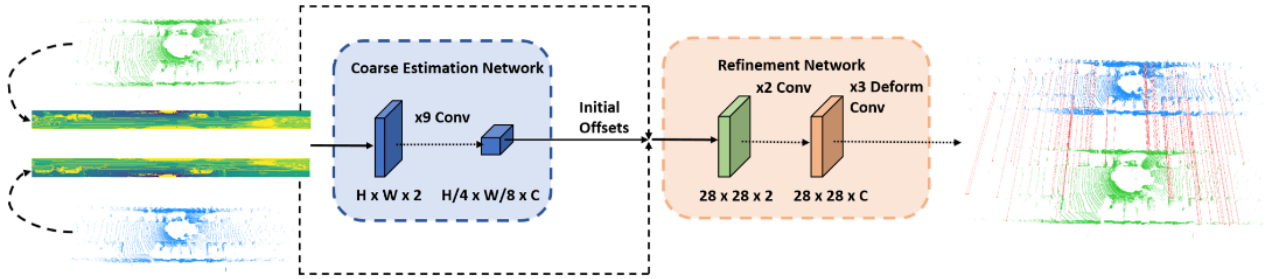


Figure 4.4: Description of DMLO[67]'s method. DMLO has a hybrid approach, where it learns to do point association between two consecutive frames. The resulting matching is combined with a differentiable SVD to predict poses, which allows to backpropagate the poses to the parameters of the network. This work uses a supervised loss comparing the prediction to ground truth poses.

4.3 *What's In My LiDAR Odometry Toolbox*

While studying in detail deep and hybrid LiDAR odometries, we found strong limits to the potential these methods seem to have. First, we found that some of the advantages presented in these approaches were, either small compared to classical equivalents, or not sufficiently proven.

This motivated the work we now present, in which we analyze and compare the previous deep and hybrid methods presented in section 4.2.2, to their classical counterparts. In it, our goal was not to propose a new methodological contribution, nor did we aim to obtain state-of-the-art accuracy on KITTI [40]. Instead, the contribution of this work lies in the analysis itself, and the conclusion drawn, which helps to better understand the limits of these end-to-end and hybrid approaches compared to classical algorithms.

This work is integrated within the python package `pyLiDAR-SLAM`, which contains publicly available implementations for deep, hybrid and classical odometries. We presented in section 3.3 the classical pipeline which is part of this work, and the minor contributions which came along. In this section, we first present the deep and hybrid methods proposed and implemented in detail in 4.3.1 and 4.3.1.3, and finally our analysis in section 4.3.2. We present in table 4.1 for clarity common abbreviations used in this chapter.

Table 4.1: Abbreviations used in this chapter. These abbreviations can be combined, so for example **P-F2F** denotes a classical frame-to-frame registration algorithm using a projective data association, **CV-Kd-F2M**, an odometry using the constant velocity model, a Kd-Tree for data association between a frame and a model, etc...

Abbreviation	Name	Description
Registration types		
F2F	Frame-to-Frame	Registration algorithm/odometry using only the previous frame to estimate the relative transform of a new scan. (*ie* without a model or a map).
F2M	Frame-to-Model	Describes registration/odometry algorithm registering new frames on a model constructed from multiple consecutive frames (also called a map in standard SLAM terminology).
Initialization types		
NI	None	No initialization, *ie* the pose is initialized with the previously inserted pose.
EI	Elevation Images	Uses a 2D alignment on elevation images to estimate the 2D motion, as described in section 3.2.2.1.
CV	Constant Velocity	An initialization which assumes that the model has constant velocity, *ie* the frame is initialized by applying the previous relative transform.
Neighbor Association types		
P	Projective	Projective data associations: neighbors computed with pixel neighborhoods on spherical images.
Kd	Kd-Tree	Kd-Tree data associations: nearest neighbors are computed using a Kd-Tree constructed on the map point cloud.
Error metrics		
RPE	Relative Pose Error	KITTI's[40] trajectory error as described in section 2.4.1.1.
Pose Regression		
RPR	Relative Pose Regression	Deep Learning objective: interprets the output of a network as the parameters of a relative pose of $SE(3)$ between two frames
APR	Absolute Pose Regression	Deep Learning objective: interprets the output of a network as the parameters of an absolute pose of $SE(3)$ within a global frame.

Contributions The principal contributions of this work are the following:

- Comparative analysis of deep, hybrid and classical LiDAR odometries
- Publicly available implementation of deep and hybrid LiDAR odometries within the `pyLiDAR-SLAM` python package

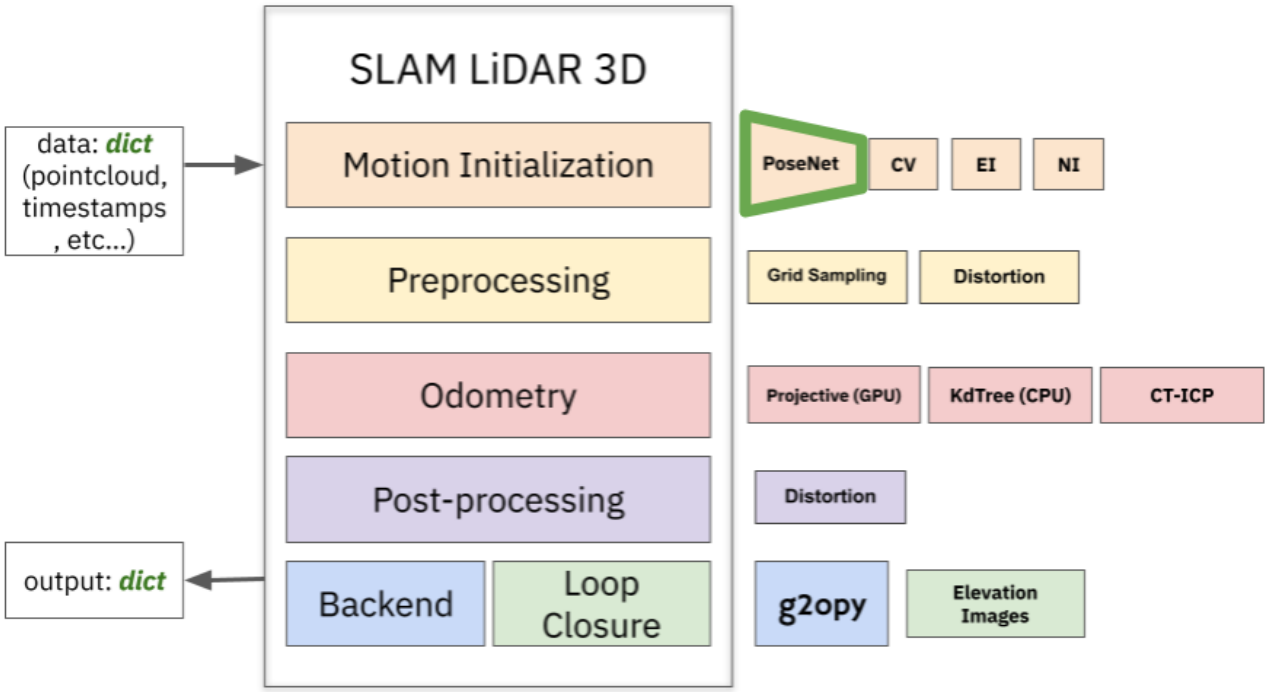


Figure 4.5: Description of the modules of `pyLiDAR-SLAM` (already shown in section 3.3). We highlighted the Deep Odometry module presented in this section.

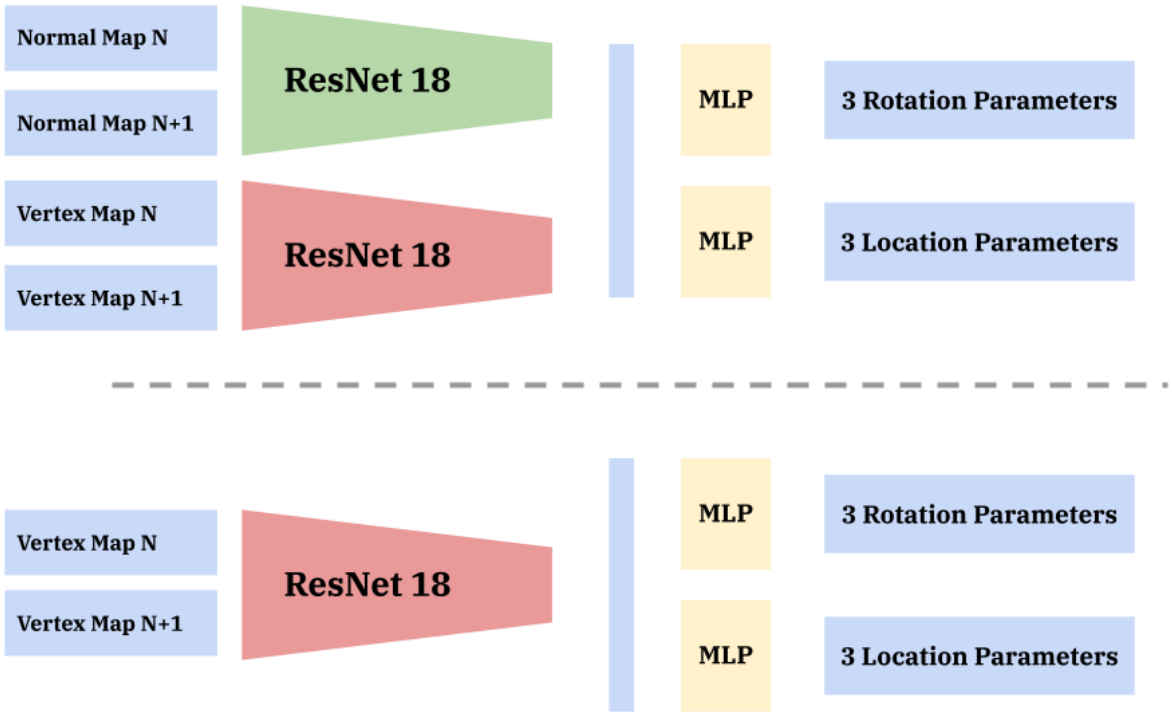


Figure 4.6: Neural Architecture for the original DeepLO [21] (above), and our simplified implementation (below)

4.3.1 Deep & Hybrid LiDAR Odometry

The end-to-end deep LiDAR odometry in this article is essentially the same as DeepLO [21], though some implementation details lead to differences in the loss formulation and training procedure, they are minor enough not to be considered real contributions in our view. So in this section, while we present this work more carefully, we will mention the stages where our method differs from the original paper.

As can be seen in figure 4.2, the PoseNet of this method expects two consecutive Vertex Maps ($\mathcal{VM}$), which are obtained by projecting a LiDAR frame in a spherical image as described in section 3.3.3. These two vertex maps are treated as 2D tensors of dimension $H \times W \times C$ (for Height $\times$ Width $\times$ Channel) and concatenated along the axis of the channel.

4.3.1.1 Network Architecture

The network architecture used in this work is a simplified version of DeepLO [21], and we present the differences below. Figure 4.6 illustrates how our architecture is constructed. Our architecture uses a ResNet18 backbone [43] to learn the feature representations from a pair of consecutive frames, concatenated along the channel dimension. The backbone outputs a feature vector of size 1024, which is passed to two distinct linear layers to predict respectively the 3 rotation parameters and 3 translation parameters of the relative pose. Each MLP head is a single linear layer with 512 parameters.

Differences with DeepLO: As figure 4.6 illustrates, our architecture differs from DeepLO, which uses two separate ResNet18 backbones for a pair of consecutive normal maps on the one hand and a pair of vertex maps on the other hand. The two resulting vectors are then added before passing through the MLP heads. In our experiments, we found that using these two backbones did not make any noticeable difference compared to using only one on the vertex

maps. We tested with the two backbones, one for normal maps and one for vertex maps, and found very similar results. In section 4.3.2, our study provides insights into the learning power of these networks, which makes it clearer why this is the case.

A final difference is the rotation representation used. In DeepLO's original approach, they proposed a normalized quaternion representation, *ie* they interpret the 3 rotation parameters as q_x, q_y, q_z. The unit quaternion is then computed as $(q_x, q_y, q_z, 1/(1 - \|(q_x, q_y, q_z)\|_2))$ from which they compute the rotation. On the other hand, we use the Euler representation and interpret the parameters as $\theta_{yaw}, \theta_{pitch}, \theta_{roll}$. Despite claims made in the original method, and through experimenting with different rotation representations (we tested the same regression of quaternions, as well as the regression of the angle-axis parametrization), we found no significant difference in performance or training capacity of using either representation.

4.3.1.2 Training & Losses

Similarly to DeepLO [21] we implemented two distinct training strategies: a *supervised* strategy and an *unsupervised* strategy. A supervised strategy requires ground truth poses, which, as explained above is often a costly operation. On the other hand, unsupervised training presents the promise of learning directly from the data, by only providing consistency constraints which is much more interesting and relevant for many applications. In our SLAM context, an unsupervised strategy would allow estimating poses for an unknown set of frames during training, providing the added benefit (additionally to the inference power of the network) of an offline mapping algorithm. There are however limitations to this view, which we present in section 4.3.2, but we first detail the two training strategies.

Supervised The PoseNet network outputs separately rotation parameter $\boldsymbol{\omega} \in \mathbb{R}^3$ and translation parameters $\mathbf{t} \in \mathbb{R}^3$. Our supervised loss L_{sup} compares these parameters to the parameters of the real pose $(\boldsymbol{\omega}^{GT}, \mathbf{t}^{GT}) \in \mathbb{R}^6$. Doing a naive loss scheme such as $L_{sup} = \|\boldsymbol{\omega} - \boldsymbol{\omega}^{GT}\|_2^2 + \|\mathbf{t} - \mathbf{t}^{GT}\|_2^2$ results in unstable training. This is because relative rotations and translations do not have the same inherent scale. In our implementation, the rotation output of the network is expressed in radians and the translation in meters. So typically the relative translation output from the network is orders of magnitude greater (in norm) than the relative rotation.

While other strategies would propose a fixed parameter rescaling in the loss component, we follow [19]'s strategy of using a learnable rescaling, and define the supervised loss as follows:

$$L_{\mathbf{t}} := \|\mathbf{t} - \mathbf{t}^{GT}\|_1 \tag{4.1}$$

$$L_{\boldsymbol{\omega}} := \|\boldsymbol{\omega} - \boldsymbol{\omega}^{GT}\|_1 \tag{4.2}$$

$$L_{sup}(\boldsymbol{\omega}, \mathbf{t}) := L_{\mathbf{t}} \exp(-s_{\mathbf{t}}) + L_{\boldsymbol{\omega}} \exp(-s_{\boldsymbol{\omega}}) + s_{\boldsymbol{\omega}} + s_{\mathbf{t}} \tag{4.3}$$

This introduces two additional scale parameters $s_{\mathbf{t}}$, and $s_{\boldsymbol{\omega}}$, which are trained and can evolve during training. This approach to learning the scaling of parameters was first introduced to balance rotations and translation loss components for training PoseNet networks in the context of image-based localization in [119].

Unsupervised More interesting in our opinion, is the unsupervised training method. Without ground truth poses, the network is trained by constructing a loss $L_{unsup}(\boldsymbol{\omega}, \mathbf{t})$ encoding geometric constraints that the estimated pose must verify. In our context, the natural loss is a registration loss (or ICP Loss as named in [21]), which is similar to the objective minimized by a classical ICP-based registration procedure (see section 3.3.2 for more details).

Given two consecutive vertec maps $\mathcal{VM}^n, \mathcal{VM}^{n+1}$, the forward propagation produces the pose $\mathbf{T}^{n,n+1}$. The points of $\mathcal{VM}^{n+1}$ are transformed using this pose and reprojected into the spherical image $\mathcal{VM}^{n,n+1}$. Then, as described in section 4.3.2, the points of $\mathcal{VM}^n$ and $\mathcal{VM}^{n,n+1}$ are associated by pixel association. Indexing by $\mathcal{I}$ all the pixels for which this association is valid, and given the normal map $\mathcal{NM}^n$, the loss is expressed as:

$$L_{unsup}(\mathbf{T}^{n,n+1}) = \sum_{\mathbf{p}\in\mathcal{I}} w(\mathbf{p}) \cdot \|(\mathcal{VM}^n[\mathbf{p}] - \mathcal{VM}^{n,n+1}[\mathbf{p}]) \cdot \mathcal{NM}^n[\mathbf{p}]\|_1 \tag{4.4}$$

$$= \sum_{\mathbf{p}\in\mathcal{I}} w(\mathbf{p}) \cdot \|(\mathcal{VM}^n[\mathbf{p}] - \mathbf{T}^{n,n+1} * \mathcal{VM}^{n+1}[\mathbf{p}]) \cdot \mathcal{NM}^n[\mathbf{p}]\|_1 \tag{4.5}$$

$$w(\mathbf{p}) = \exp(-(\|\mathcal{VM}^n[\mathbf{p}] - \mathcal{VM}^{n,n+1}[\mathbf{p}]\|_2^2)/\sigma^2) \tag{4.6}$$

Note again that this is the standard point-to-plane error minimized by classical lidar-odometry (see chapter 3). We add only a single weight for each residual $w(\mathbf{p})$, which, and this is important, is detached from the gradient backpropagation.

In DeepLO's original paper, an additional loss term $\mathcal{L}_{fov}$ was incorporated, to prevent divergence during training. This divergence can easily occur, which leads the network to predict large and wrong pose parameters. In this case, very few points are projected back into the spherical image, which leads to singular loss functions. However, our weighting scheme was enough to prevent divergence, by providing a more robust frame-to-frame registration objective, compared with the one proposed by DeepLO.

4.3.1.3 Hybrid LiDAR Odometry

Our hybrid approach adopts a strategy similar to Lo-Net [66] and [80], *ie* we use a PoseNet as an initialization strategy. More precisely, the PoseNet model is trained as a primary offline step and exclusively with the unsupervised training strategy presented above. For each new frame, our hybrid SLAM algorithm first initializes the pose with the PoseNet network, by predicting the relative pose between the previously inserted frame to the new one. Then, the new pose estimate is used within our frame-to-model registration procedure presented in section 3.3.

Note, that we can combine this initialization approach with the two registration procedures implemented within `pyLiDAR-SLAM`, the Kd-Tree based frame-to-model approach (*Kd-F2M*) and the projective frame-to-model approach (*P-F2M*). Essentially resulting in two distinct hybrid algorithms based on PoseNet.

4.3.2 Experiments

We conducted a large campaign of experiments of the different modules implemented within `pyLiDAR-SLAM`, which we present in this section. These experiments will serve as a basis of our comparative analysis in section 4.3.3. First, we present the datasets and metrics used in this experiment in 4.3.2.1, which differ slightly from the one used in the LiDAR-Odometry chapter 3. Then we present our main results in section 4.3.2.2, and show that we reach near state-of-the-art performance, which legitimates our analysis in the next section.

4.3.2.1 Datasets & Metrics

In this work, we use principally the **KITTI** dataset [40], which we already presented in section 2.4.1.1. As mentioned above, this driving dataset is the most popular public benchmark to evaluate LiDAR odometries, and all the existing hybrid and deep LiDAR odometries [66, 19, 80] evaluate and compare themselves on this dataset. For this reason, we mainly focus on this

dataset for the core of our analysis. From this dataset, we only retain the first 11 sequences for which the ground truth poses are provided and evaluate all the methods proposed on this set of sequences.

To train PoseNet for our hybrid and deep odometries, we split these sequences between a training and test set. Though we experimented with multiple test sets as we explain below, our principal split use sequences `00-08` as training sequences and `09,10` as test sequences, similarly to [21].

Ford Campus [85] is another driving, which consists of 9996 frames acquired on the Ford Campus of Dearborn Michigan, with another HDL-64 LiDAR rotating at 10Hz, and similarly to KITTI, the LiDAR is installed in autonomous driving configuration (*ie* with the rotation axis of the sensor perpendicular with the ground-plane).

The motion of the car in this dataset is much simpler than KITTI's (far fewer turns, long stops at red lights, and long straight lines). However, it offers new environments such as parking lots, and typically American small buildings, with some additional challenges (many mobile objects).

Thus this dataset is a great addition for studying the generalization capacities of PoseNet between similar datasets, and we use it mostly to this effect in our analysis. The dataset, which in terms of size is roughly half the size of KITTI, contains 2 sequences (`short` and `long`). For training, we use the `long` sequence as the training sequence, and the `short` sequence as the test sequence.

NCLT [13] is used lastly, to test the capabilities of the network outside of the context of autonomous driving, and with different sensor configurations. As presented in section 2.4.2.1, this dataset was acquired with a Velodyne HDL-32 mounted on a segway, traveling through Michigan University's North Campus. This dataset contains much more sequences, much longer than KITTI's, so we used sequences `2012-01-08` for testing and 10 other sequences for training (sequences `2012-01-22` to `2012-03-31`).

Metric For each method and each dataset, we use KITTI's Relative Pose Error (**RPE**) as a sole metric, which allows us to compare efficiently our method with other state-of-the-art methods evaluated on KITTI. Already mentioned in section 2.2, this metric averages the translation errors over all segments of a given trajectory over multiple lengths (100m to 800m).

4.3.2.2 Results

The deep and hybrid odometries presented in section 4.3.1 were implemented in Python using the PyTorch library. The networks were trained using the Adam optimizer with default parameters, for 100 epochs, with an initial learning rate of 10^{-4} divided by 2 every 20 epochs.

For the classical LiDAR odometry (presented in section 3.3), the local map is built from the last 30 point clouds registered. As a preprocessing step for the Kd-Tree-based classical frame-to-model LiDAR odometry (**Kd-F2M**) implemented in `pyLiDAR-SLAM`, we perform a grid sample with a voxel size of $0.4m$. For the projective classical frame-to-model odometry (**P-F2M**), the preprocessing step is only the projection in a 64×720 spherical image. This resolution is the same as that of the input vertex maps of the PoseNet network and acts as a sampling procedure that reduces the initial frame resolution (which was 64×1024).

We present the odometry results in terms of the **RPE** metric in table 4.2. It shows that we obtain near state-of-the-art results for our tree-based LiDAR odometry (Kd-F2M) and even better results than the published projective methods with our projective method (P-F2M). And similarly for our PoseNet with the weighted L2 loss and in the hybrid LiDAR odometry. The

Table 4.2: Relative Pose Error (**RPE**) as a percentage of distance traveled for KITTI's Dataset, comparing Published LiDAR Odometries and the Proposed Methods (italic) / EI="Elevation Image", CV="Constant Velocity", F2M="Frame-to-Model", * are training sequences for PoseNet.

	00*	01*	02*	03*	04*	05*	06*	07*	08*	09	10
Classical Tree-Based LiDAR odometries											
IMLS-SLAM [26]	**0.5**	0.82	0.53	0.68	**0.33**	0.32	**0.22**	0.33	0.8	0.55	0.53
LOAM [135]	0.78	1.43	0.92	0.86	0.71	0.57	0.65	0.63	1.12	0.77	0.79
EI + Kd-F2M	0.53	0.79	0.52	0.69	0.45	0.34	0.31	0.36	0.79	0.54	**0.51**
CV + Kd-F2M	0.51	**0.79**	**0.51**	**0.64**	0.36	**0.29**	0.29	**0.32**	**0.78**	**0.46**	0.57
Projective LiDAR Odometries											
SuMA [4]	0.68	1.70	1.20	**0.74**	**0.44**	0.43	0.54	0.74	1.20	0.62	**0.72**
EI + P-F2M	**0.57**	**0.67**	**0.62**	0.83	0.51	**0.37**	**0.36**	**0.32**	**0.93**	**0.60**	1.01
CV + P-F2M	0.57	0.71	0.62	0.82	1.12	0.37	0.37	0.33	0.93	0.61	1,01
Unsupervised Deep LiDAR Odometries											
DeepLO [21]	1.90	37.83	2.05	2.85	1.54	1.72	**0.84**	**0.70**	**1.81**	6.55	7.74
Self-supervised LO [80]	-	-	-	-	-	-	-	-	-	**6.05**	**6.44**
PoseNet + ICP loss	**1.33**	7.11	1.81	3.09	0.93	1.38	1.43	0.84	2.27	6.31	8.99
PoseNet + Weighted ICP (W-ICP) loss	1.36	**1.88**	**1.46**	**2.33**	**0,97**	**1.26**	1.05	1.07	2.05	6.79	7.6
Hybrid LiDAR Odometries											
LO-Net [66]	0.78	1.42	1.01	**0.73**	0.56	0.62	0.55	0.56	1.08	0.77	0.92
PoseNet (W-ICP loss) + P-F2M	0.60	**0.67**	0.68	0.85	0.56	0.41	0.39	0.34	0.96	0.69	1.07
PoseNet (W-ICP loss) + Kd-F2M	**0.55**	0.85	**0.58**	0.74	**0.44**	**0.34**	**0.36**	**0.33**	**0.88**	**0.61**	**0.83**

goal of this work was not to propose state-of-the-art LiDAR odometries (by contrast with our work CT-ICP, presented in section 3.4). Instead, we show here that the performances of our methods justify the analyses we present in the next section.

Yet, we can already make some observations from the results presented in section 4.2. Notably, we see that the tree-based method offers superior accuracy than projective methods. This is natural, as tree-based methods generally offer more precise neighbor associations, at the cost of slower execution. Yet in the context of autonomous driving, the difference is not very important, and we can see for example that our projective frame-to-model obtains better results than LOAM's original results. Additionally, it appears clearly that PoseNet as an initialization strategy and as part of hybrid methods offers only very marginal gains compared to other strategies, and we study this more in detail in section 4.3.3.3. Finally, it is also clear in the table, that there is still a large gap between classical frame-to-model (**F2M**) LiDAR odometries and unsupervised deep learning-based odometries. And one of the goals of this work was to study the potential of deep LiDAR odometries and answer the question: can LiDAR odometries become practically useful and help improve classical LiDAR odometries ? We give some answers to this question in the next section.

4.3.3 Comparative Study

We now present the main contribution of this work, the comparative analysis of deep, hybrid and classical LiDAR odometries. The original idea behind this analysis was to understand the strengths and weaknesses of each method and understand how they could complement each other.

First, we investigate the claim made by several papers that PoseNet outperforms frame-to-frame (**F2F**) ICP-based odometries in section 4.3.3.1, and essentially show that it is relatively easy to build a classical **F2F** odometry contradicting this claim. Then, in section 4.3.3.2, we take a closer look at PoseNet and show the limits of its generalization capacity. Essentially, we show that further research is needed before any practical or industrial use of these methods.

And this leads us to study PoseNet as an initialization strategy, and how it compares to other, simpler initialization strategies; this is the subject of section 4.3.3.3. Finally, we expose in section 4.3.3.4 the weaknesses and failure cases of classical LiDAR odometries which reveal scenarios where PoseNet could theoretically be useful.

4.3.3.1 PoseNet vs Classical Frame-to-Frame Odometry

In other approaches, PoseNet is often compared to the default **F2F** ICP registration [19, 80] (with either a point-to-plane or a point-to-point metric). We, on the other hand, consider the comparison unfair and show that very little effort is required to build a decent frame-to-frame LiDAR odometry, which can obtain comparable, and even slightly better precision on both the training and test sets. We show the results of running a frame-to-frame LiDAR odometry on KITTI for different initialization strategies in table 4.3.

First, it confirms the result of previous approaches, that using a naïve point-to-plane registration with no initialization (**NI**) leads to worse performances than PoseNet (rows 3 and 6 of the table). However, using very simple initialization methods such as the constant velocity model (**CV**), or the elevation image alignment (**EI**), both presented in section 3.3.1, we already obtain results much better than PoseNet on the training set, and significantly better and the test set. In a way, this is normal and even reassuring, as a **P-F2F** alignment is precisely the supervision signal used to train PoseNet. Yet, the mechanism of construction of the pose is very different for PoseNet and an ICP-based method. While an ICP iteratively refines an original estimate until convergence, PoseNet maps features extracted from a pair of consecutive scans to a relative pose, by a series of nonlinear deformations of space parametrized and optimized with the training data. Thus, the precision of the pose, and this mechanism, is limited by the representative power of the network, the capacity to extract accurate and discriminative features and the volume of training data. While the former method's precision simply depends on noise, the richness of the environment, and the ability to make correct neighbor associations. On a side note, these parameters are also important for training PoseNet in the unsupervised setting.

But the table 4.3 still hints at a potential utility for PoseNet: when the motion cannot be properly initialized (either due to abrupt motion, or initializing in rotations), PoseNet can use all the data of the training set as a regularizer and obtain in some cases decent enough predictions, which explains why they outperform the standard frame-to-frame with no initialization **NI-F2F**. Still, in this rather "niche" case, to train a PoseNet supervision is required, but if the strategy considered is the unsupervised setting, the supervision signal is precisely the classical method **P-F2F**. Thus we have the following paradox: when this signal is possible, classical methods typically perform well (as shown with KITTI's example).

4.3.3.2 Understanding the limitations of PoseNet for RPR

Studying PoseNet, we found that the idea that PoseNet as a Relative Pose Regression (**RPR**) estimator learns to find optimal pose between consecutive frames is a false promise. A previous work [105] already showed the limitations of PoseNet for *Absolute* pose regression (in the context of localization), and notably showed that the behavior of such networks was much closer to image retrieval techniques than the desired geometric understanding of a scene. Without this geometric understanding, it is clear why PoseNet has poor generalization capacities. And, in this section, we show that this conclusion is also true for PoseNet for **RPR** and we explore the limits of PoseNet generalization to unknown motion (which corresponds to the label distribution) or unknown environments (corresponding to input distribution).

Table 4.3: Relative Pose Error (**RPE**) on KITTI: compares PoseNet to different Frame-to-Frame (F2F) methods / NI ="No Initialization", EI="Elevation Images", CV="Constant Velocity" / P="Projective" and Kd="Kd-Tree" for the respective data associations. "*" Denotes training sequences

	KITTI Training Sequences	**KITTI Test Sequences**
	{00-08}*	{09-10}
PoseNet (ICP loss)	2.24	7.65
PoseNet (Weighted ICP loss)	1.49	7.19
NI + P-F2F	40.1	30.4
CV + P-F2F	1.46	1.7
EI + P-F2F	1.47	1.9
NI + Kd-F2F	24.18	14.04
CV + Kd-F2F	1.41	1.84
EI + Kd-F2F	1.41	1.87

Generalization to unobserved motion We already saw in table 4.2 and 4.3, the difference in accuracy between training and testing for PoseNet. Additionally, we examine the behavior of PoseNet on motion unobserved during training. To achieve this, we use KITTI's sequence `01` (highway), which is unique among the KITTI dataset which is mainly composed of residential and city environments (sequence {`00,02-10`}). The test split {`11-21`} also contains several road sequences but with a maximum speed of 90km/h.

We present in figure 4.7 the distribution of the motion (using the ground truth) and the PoseNet prediction for this split. We also trained PoseNet using all the sequences but the `01`, resulting in the split {`00,02-21`}(city, suburbs, roads), and present the trajectory errors for both splits in the table 4.4.

We can see that both training splits lead to poor generalization capacity of PoseNet to the unobserved motion of sequence `01`. The scale of the motion predicted by PoseNet does not leave the distribution observed during the training set as is clearly shown in figure 4.7. Interestingly, when faster roads are added to the training set (split `00,02-21`), PoseNet can relate the motion of sequence `01` to the motion of the roads (with speeds up to 80km/h), but cannot interpolate further. This illustrates the same behavior as for absolute pose regression: essentially PoseNet associates a pair of frames with the closest motion observed in the dataset.

Generalization between datasets After showing poor generalization to unknown output distributions, we now show the limitations of PoseNet's capacity to generalize to different but comparable datasets. Essentially, we show that currently PoseNet networks have poor generalization between datasets. To do this, we trained a PoseNet on KITTI and tested it on the Ford Campus dataset (and reciprocally). The results are shown in figure 4.4. For both datasets, we find that PoseNet obtains decent results on the training sequences, however, obtains poor results on the other dataset. This is underwhelming, as both datasets were acquired using the same LiDAR, in the same driving configuration, using the same spherical projection parameters to build the PoseNet inputs. Thus, it generally, shows the limited generalization capacities of PoseNet, even for comparable scenarios. And shows that work is required before PoseNet becomes pertinent as a network used for inference in real conditions.

4.3.3.3 Initialization Strategies

We now study the last application potential of the network: *ie* PoseNet as an initialization procedure. This approach was already introduced by LO-Net [66] as a supervised network

Table 4.4: Relative Pose Error (**RPE**) on sequence `01` of KITTI shows that PoseNet fails to infer unobserved motion during training / CV="Constant Velocity", P-F2F="Projective Frame-to-frame". "*" Denotes training sequences.

	Target KITTI Sequences	
	KITTI {00, 02-10}	**KITTI 01**
PoseNet *(KITTI {00, 02-10}*)*	1.51	56.04
PoseNet *(KITTI {00, 02-21}*)*	1.63	13.46
CV + P-F2F	1.52	2.17

Table 4.5: Relative pose Error (**RPE**) shows Poor Generalization of PoseNet between KITTI and Ford Campus datasets. "*" Denotes training sequences.

	Target Datasets	
	Ford Campus	**KITTI {00-10}**
PoseNet *(Ford Campus)**	3.94	73.42
PoseNet *(KITTI {00-21})**	71.56	1.54
CV + P-F2F	4.32	1.52
CV + P-F2M	2.04	0.69

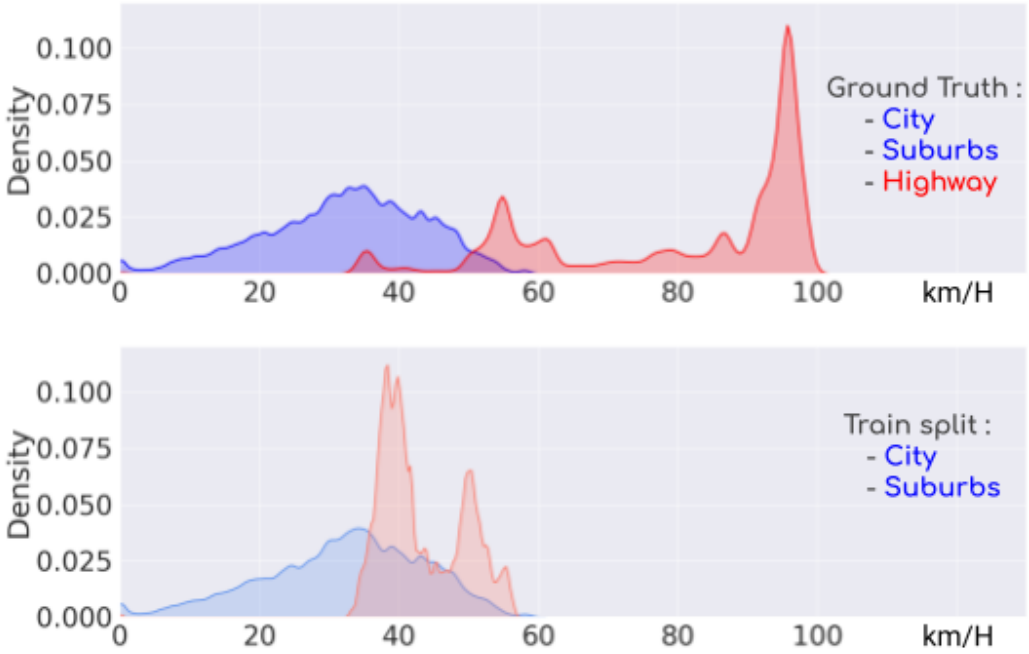


Figure 4.7: Distribution of the speed (scaled in km/h) for KITTI's agglomeration sequences (`{00, 02-10}` in blue) and highway (`01` in red) computed from the ground truth (Top) and from PoseNet's prediction (light blue and light red) for the training split `{00, 02-10}` (Bottom). The PoseNet incapacity to generalize well to motion unobserved during training is demonstrated.

predicting a motion estimate before mapping, and was extended in [80] with unsupervised training for PoseNet.

This approach is meaningful, as initialization is a crucial step for a successful **F2M** odometry, due to the small convergence radius of ICP methods. However, for a deeper complete analysis, it is interesting to compare it to simpler approaches in isolation, as well as in more

challenging settings. This is the topic of this section.

KITTI and Ford Campus: Two training strategies are relevent for PoseNet + **F2M**, yet carry different meanings. The first strategy uses all available data for training, using the regularizing effect of the data to improve the PoseNet prediction. It can be relevant when the LiDAR odometry is used offline, for example as a mapping algorithm. The second strategy uses only data unobserved during training and evaluates the capacity of PoseNet to be used online as an initialization method. To evaluate both scenarios, we train PoseNet on two different splits of KITTI, sequence `00-21` for the mapping scenario and `11-21` for the online scenario. We show the results in table 4.6. We see that PoseNet does improve **P-F2M** slightly, compared to the simpler **EI** and **CV** approaches, when running on data observed during training (see rows 1 and 3 on KITTI). Yet, this is not the case when unobserved data is considered, which leads to poorer performances. We can make the connection with the analysis of section 4.3.3.2: due to PoseNet's bad generalization capacities when the motion or environment at inference differs from the training distributions. Still, the gains obtained using test data are very marginal for the complexity of using PoseNet and are very likely null for more precise LiDAR odometry such as CT-ICP.

NCLT: We now study LiDAR odometries in another context than driving scenarios where both classical and hybrid LiDAR odometry are very good, precisely because the motion is easily predicted, as the vehicle mostly has a straight trajectory with occasional turns. This leads to simple registration and high map qualities. We now test a method with the more challenging NCLT dataset. The LiDAR is sparser (only 32 channels) and suffers from abrupt rotations in the yaw (around its axis) and pitch angles (the acceleration/deceleration motion leads to inclinations of the segway).

As mentioned above, we trained PoseNet on 10 sequences in both supervised and unsupervised settings (for more than 200k frames of the same environment). The results for the sequence `2012-01-08` (test) and `2012-01-22` (train) are presented in table 4.7, and the corresponding trajectories in figure 4.8. They show very poor performances for **CV** as an initialization method compared to **EI**. This illustrates well the importance of proper initialization for classical methods. **EI** provides a better good first estimate of the 2D motion (notably the rotation around the z axis), which allows correct registration and thus keeps the map clean. In contrast, the **CV** here fails to provide a good enough initialization to ensure convergence of the **F2M** and quickly leads to poor map qualities due to inaccurate registrations.

Secondly, we can compare **P-F2M** and **Kd-F2M** in a more challenging registration setting. We see that our **Kd-F2M** behaves better than the **P-F2M** implementation. This is expected as the projective alignment leads to less precise neighbor associations. What's more, the **P-F2M** is even more sensitive to abrupt rotations, which can lead to large pixel distances between neighbors. However, the most important difference here does lie in the initialization strategy. And we see that with the appropriate initialization strategy, even **P-F2M** as correct results, with reasonable drift, as can be seen in figure 4.8.

On a last note, we see that PoseNet behaves poorly on the dataset in supervised and unsupervised settings, for both the training and test sets. This again illustrates the limited representative power of our current PoseNet for more challenging motions. Despite using 10 times more data than for KITTI in a repeating environment, PoseNet is unable to learn a mapping from pair of frames to relative poses at a precision interesting enough to allow correct registrations. Due to the abrupt rotations, the motion of the sensor is located in a much greater volume of the parameters space than for driving motions. Thus, further research is required to allow neural network architectures to address these more challenging settings.

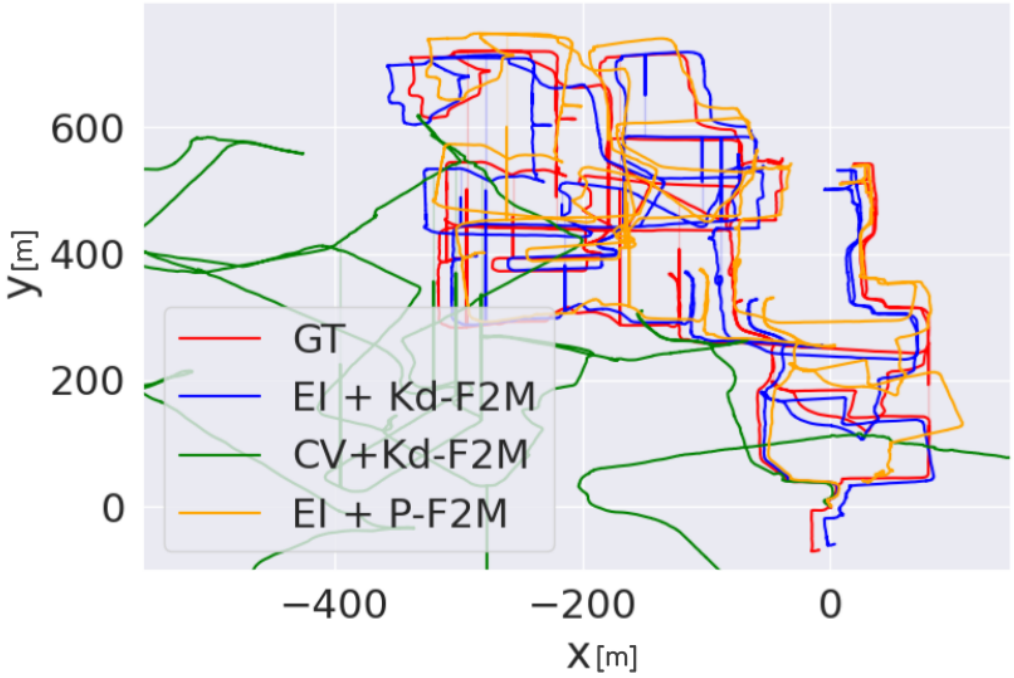


Figure 4.8: Trajectories of sequence `01/08` of the NCLT dataset obtained by the proposed **Kd-F2M** for different initialization strategies. The figure shows that the constant velocity (**CV**) (green) model is less appropriate for this dataset than our 2D registration-based estimator (**EI**). Ground Truth (GT) is in red.

Table 4.6: Relative Pose Error (**RPE**) on KITTI and Ford Campus for different initialization strategies, followed by **P-F2M** / **P-F2M**=Projective Frame-to-Model / CV="Constant Velocity", EI="Elevation Image".

	Target Datasets	
	Ford Campus	**KITTI {00-10}**
PoseNet 00-21* + P-F2M	-	0.66
PoseNet 11-21* + P-F2M	-	0.81
PoseNet Ford Campus* + P-F2M	2.41	-
CV + P-F2M	2.04	0.69
EI + P-F2M	2.11	0.69

Table 4.7: Relative Pose Error (**RPE**) on sequences `2012-01-08` and `2012-01-22` of the NCLT dataset, shows the importance of the selection of the initialization for LiDAR odometries.

	NCLT Target Sequences	
	2012-01-08 (test)	**2012-01-22* (train)**
PoseNet (supervised) + Kd-F2M	35.5	37.5
PoseNet (unsupervised) + Kd-F2M	48.7	46.3
CV + Kd-F2M	11.49	18.7
EI + Kd-F2M	1.84	3.73
EI + P-F2M	4.32	5.21

Table 4.8: Number of failures over driving datasets. "*" Denotes training sequences.

	Target Datasets	
	Ford Campus	**KITTI {00-10}**
PoseNet *(KITTI {00-21})**	-	0
PoseNet *(KITTI {11-21})**	-	0
PoseNet *(Ford Campus)**	261	-
NI + P-F2F	1494	3286
EI + P-F2F	270	257
NI + Kd-F2F	1127	1845
EI + Kd-F2F	261	0
Number of Frames	9996	23201

4.3.3.4 Weaknesses of Geometric LiDAR Odometries

So far, we prove that **F2M** LiDAR odometries obtain superior results when a rich and dense map is constructed and when the new scan to be registered is moved into the small convergence domain of the mapping algorithm with a good initialization. The challenges remaining are then the construction of this precise map and the prediction of the initial motion. This provides hints of weaknesses of the classical methods: elements deteriorating the quality of the map (such as mobile objects, featureless regions, and bad scan registration) as well as rapid and unpredictable motion (see chapter 3).

Another weakness we can mention here is the initial construction of the map. For the first two frames, **F2M** is equivalent to **F2F** without motion initialization. Bad initial map construction can be recovered for methods using a sliding local map (e.g. IMLS-SLAM [26]), which can forget badly registered scans, but the trajectory and map constructed will only be correct locally and not globally. It is even more problematic for methods relying on a global map (such as LOAM [135]).

To illustrate this sensitivity in the driving datasets KITTI and Ford Campus, we count the number of failures to align two consecutive frames (without any motion priors). More precisely, given a predicted pose location $\mathbf{t}_{pred} = (t_x, t_y, t_z)$ and Euler angles $\mathbf{e}_{pred} = (e_x, e_y, e_z)$ and the corresponding ground truth pose $(\mathbf{t}_{pred}, \mathbf{e}_{pred})$, we consider the prediction to be a failure if $\|\mathbf{t}_{pred} - \mathbf{t}_{gt}\|_2 > 1m$ or $\|\mathbf{e}_{pred} - \mathbf{e}_{gt}\|_2 > 3°$. We report in table 4.8, the number of such failures of the **F2F** for KITTI and the Ford Campus dataset. This table shows the known limitation of classical ICP based **F2F** alignments, which have many failures, notably in the projective case. Yet, we see that our **EI** initialization performs well in these outdoor datasets, and has few failure cases.

Finally, we see that for this metric, PoseNet performs well both on training data (rows 1 and 3 of table 4.8), and on unobserved data (row 2). Thus despite its weak generalization properties, and when it can be trained, PoseNet can provide a good enough initial estimate and limits the number of absurd registrations which can occur with ICP-based **F2F**, as PoseNet essentially outputs poses observed during training. In contrast, the ICP outside of its convergence region can move in any direction of the parameter space. So in a sense, PoseNet encodes the motion model of the training dataset.

4.4 Conclusion

In this chapter, we presented a detailed analysis of the strength and weaknesses of deep and hybrid LiDAR Odometries. This work shows the strong limitations that deep and hybrid LiDAR odometries have compared to their classical counterparts. Broadly speaking, they offer no or very coincidental improvements over classical odometries, they require training and tend to generalize poorly to new conditions. So in practice, no advantage of these methods over classical methods has been proven. This work has been published in the article *What's In My LiDAR Odometry Toolbox* [24], which has been accepted at the conference *IROS 2021*. It is made freely available online, and the associated GitHub project, `pyLiDAR-SLAM` (`https://github.com/Kitware/pyLiDAR-SLAM`) has more than 200 stars as of today.

As mentioned in the related work (section 4.2), different methods have been published since ours, proposing new deep [118, 128, 79, 122] or hybrid odometries [67, 117]. They have improved the performance of deep and hybrid odometries, notably (and this is the most important metric) in terms of generalization. One of the determining factors for this is the use of point-cloud-friendly feature encoders, instead of image-based CNNs. Still, the conclusion of this chapter remains mostly unchanged: none have yet proven a real advantage over classical methods.

Thus, in this chapter, we showed clearly that these types of approaches were not yet satisfying enough to help us address the problems of classical LiDAR Odometries identified in chapter 3, despite some early promises. In the next chapter we investigate a different domain, which we will see, is much more pertinent to improve LiDAR-Only odometries, by fusing LiDAR with inertial measurements.

Taking a step back, let us be clear that our work here is not a rejection of Deep Learning for the context of LiDAR SLAM. It is clear that Deep Learning has its place in LiDAR SLAM pipelines and is currently very underused. As we saw, in chapter 3, classical SLAM pipelines have shortcomings that are difficult to address using handwritten rules. Semantic segmentation could be very interesting to remove mobile objects from the points to insert in the map, which would help pollution. Weighting residuals based on classes (Vegetation, potentially mobile objects, etc...), or for straight error detection, could also be very promising approaches. For point-cloud-based loop detection and place recognition, there is also a lot of work to be done. But trying to replace classical LiDAR odometries with end-to-end deep odometry now seems a fool's errand, as it is, in effect, trying to replace the one thing that LiDAR SLAM does work very well already.

Note that it didn't prevent us from trying...

Chapter 5

LIO: LiDAR-Inertial odometries

Contents

Résumé

Dans le chapitre 3, nous avons montré que CT-ICP est une odométrie LiDAR très précise, particulièrement pour des scénarios de conduite automobile. Cependant, nous avons identifié différentes catégories de cas d'échecs. Nous avons exploré dans le chapitre 4 les méthodes d'odométries LiDAR deep et hybrides, et nous avons montré que ces méthodes n'offrent aucun bénéfice sur les méthodes classiques. Donc, dans ce chapitre, nous explorons une nouvelle approche: fusionner des capteurs LiDAR et inertiels.

Les IMU (Inertial Measurement Units) mesurent les dérivées du mouvement de la trajectoire (vélocité angulaire, accélération linéaire), à une fréquence de sortie souvent beaucoup plus importante que le LiDAR. Pour des IMU de type MEMS, leurs fréquences de sorties sont souvent comprises entre 100Hz et 400Hz. Ces types de capteurs ont des informations complémentaires: les LiDARs donnent des informations géométriques sur l'environnement, alors que les IMUs contraignent directement la trajectoire.

Les systèmes LiDAR Inertiel sont donc de parfaits candidats pour résoudre les échecs des odométries LiDAR dus à des mouvements rapides et saccadés. Il existe bien entendu un large corpus de méthodes proposant diverses manières pour fusionner ces deux types de données. Nous en présentons une liste non exhaustive dans la section 5.2.

Dans ce chapitre, nous prenons une approche singulière. Nous partons de notre odométrie LiDAR CT-ICP, et graduellement, nous augmentons ces capacités en proposant des méthodes de plus en plus raffinées.

Nous montrons notamment, qu'en utilisant uniquement les mesures brutes du gyroscope, et sans même recourir à des modèles d'estimation d'état compliqués, nous pouvons améliorer les performances de CT-ICP, et notamment mieux gérer différents cas d'échecs de cette méthode. Dans la section 5.3, nous présentons les différentes étapes de notre approche, et dans la section 5.4, les expérimentations qui nous permettent de démontrer la pertinence de chaque étape.

Contributions: Les contributions présentées dans ce chapitre sont les suivantes:

- Une approche graduelle pour améliorer CT-ICP en utilisant les mesures de gyroscope.
- La démonstration des gains de chaque amélioration à travers une série d'expérimentations.

5.1 Introduction

We saw in chapter 3 that CT-ICP is a very precise LiDAR-Only odometry, especially for driving scenarios. However, we also saw different categories of failure cases. We tried in chapter 4 to explore end-to-end LiDAR Odometries to address them but showed that, currently, such methods offer no benefits over classical methods. Thus, in this chapter, we explore another approach: fusing LiDAR with Inertial Measurements.

Inertial Measurement Units (IMU) measure derivatives of the trajectory's motion (angular velocity, linear acceleration) often at a much higher frequency than the LiDAR (typically for the MEMS-based IMU, at a frequency between 100Hz and 400Hz). Both types of sensors have complementary information: LiDAR provides geometric information about the environment, while IMUs provide trajectory constraints.

Thus LiDAR-Inertial methods are the perfect candidates to address the failures of LiDAR Odometries which are due to abrupt motion. There exists a large body of work proposing novel methods on the topic, of which we present a list non-exhaustive in section 5.2.

In this chapter, we take a singular approach, where we start from our LiDAR-Odometry CT-ICP, and gradually augment its capacities with iteratively more refined approaches. We show, notably, that, leveraging only the raw gyroscope measurements, without state-estimation models, we drastically improve the performance of CT-ICP, and handle many failure cases of the method. We present the different stages of this approach in section 5.3, and present in section 5.4 the experiments which demonstrate the relevance of each stage.

Contributions: The contributions presented in this chapter are the following:

- A gradual approach to improve CT-ICP using gyroscope measurements.
- Experiments demonstrating the improvements for each step.

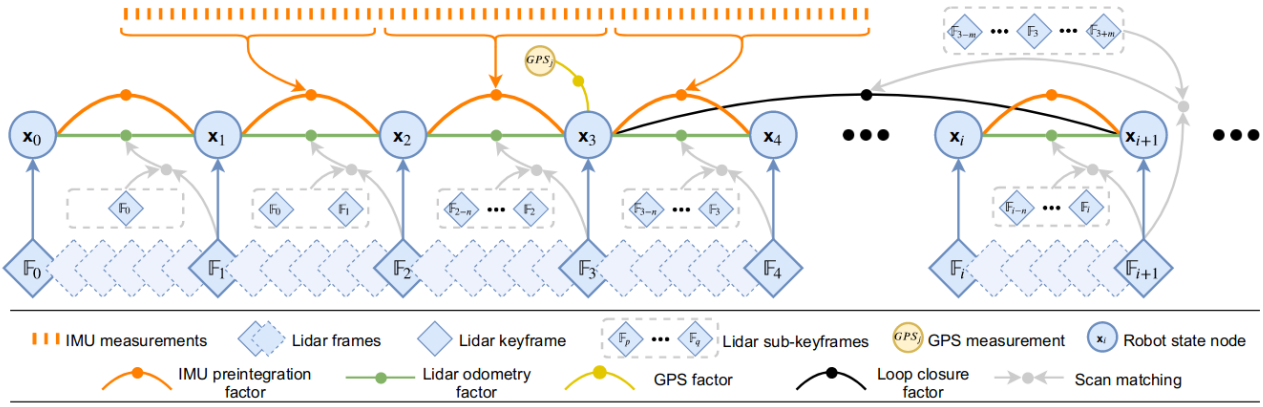


Figure 5.1: Description of LIO-SAM's approach [112] method. LIO-SAM uses a Factor Graph to integrate multiple sources of constraints to estimate the trajectory (GPS, IMU, Loop-Closure). The registration of LIO-SAM is however, loosely coupled, as IMU measurements are only used to distort the point cloud and initialize the motion.

5.2 Related Work

In this section, we present the principal methods which have been proposed to fuse LiDAR and inertial measurements. We presented in section 3.2 the different types of LiDAR odometries, and already presented some *LIO* methods, but through the perspective of registration. Thus, here, we focus only on the mechanisms to fuse IMU and LiDAR measurements.

Similarly to *LO*, there is a large body of work in this field and many different approaches to integrate IMU measurements into LiDAR Odometries. The first approach is to decouple the registration from the integration of IMU measurements. This approach uses a conventional registration procedure, similar to those presented in section 3, but preprocesses the point cloud and initializes the motion using the IMU measurements. We present work of this category in section 5.2.1.

A second type of methods, which is typically called *Tightly Coupled Approaches*, use IMU measurements within the registration procedure, typically as constraints that are designed to improve the performance of the registration procedure. We present this type of method in section 5.2.2.

5.2.1 Loosely-Coupled Approaches

Loosely-Coupled methods separate the fusion of IMU and LiDAR measurements in two different stages. Typically, a probabilistic model of the trajectory predicts accurately the motion when IMU measurements arrive at a high frequency. This allows a very precise initialization of the LiDAR frame. The corrected motion is then used to update and refine the model. The representation of the trajectory is typically modeled as a Pose Graph or a Factor Graph, which allows the integration of multiple trajectory constraints (including loop closure, odometry constraints, GPS constraints, etc ...). Libraries such as GTSAM [23] or G2O [61] have been a great help to the robotics community, providing frameworks to construct and optimize efficiently such models. For a factor graph representation, [37] proposed a practical method to pre-integrate IMU measurements. This method, which is implemented within GTSAM, allows adding IMU measurement constraints within Factor Graphs, at the same frequency as the LiDAR poses, allowing to constrain the trajectory model with IMU measurements. This work has been instrumental to construct Loosely-coupled approaches [112, 83, 99, 115].

The LiDAR-Odomety stage of loosely-coupled approaches is essentially always a standard

LiDAR odometry step, initialized with a precise trajectory estimate. During the accumulation of the LiDAR frame (at a frequency of 10 to 20Hz), the IMU measurements collected allow the trajectory model to predict the pose more accurately than the standard motion models seen in previous sections. LOAM [135] proposed an optional IMU module for their odometry (see section 3.2 for more details on the method). For their approach, they use a simple linear Kalman filter to estimate the relative motion of a scan with the IMU measurements. They then use the estimated relative motion to pre-distort the scan before forwarding it to their main odometry module.

LIO-SAM [112], uses [37] method implemented in GTSAM to build a factor graph allowing them to estimate the IMU biases and predict the relative motion within a frame. This motion is used to distort the frame, using only the gyroscope measurements (thus correcting only the distortion due to the relative rotation within the frame), and initialize the registration. The registration uses the standard LOAM [135] approach, and the optimized pose is then used to refine the factor graph responsible for the IMU preintegration. Additionally, they model their global trajectory with a separate Factor Graph, which receives absolute measurements from GPS, LiDAR odometry and IMU to refine globally the trajectory.

Part of the solution of the CoStar team for the Subterranean challenge relies on a loosely-coupled LIO. They proposed LOCUS [83, 99] which is a standard generalized-ICP [108] based LiDAR odometry, which uses IMU measurements to distort the point cloud frames of their platforms as a preliminary step. The modeling of the trajectory also relies on GTSAM, and is presented in their work [115].

5.2.2 Tightly-Coupled Approaches

Another type of approach fuses directly IMU and LiDAR measurements within the Pose Estimation problem. These *Tightly-Coupled* approaches, can either keep a probabilistic state-estimation approach with filtering or smoothing formulations. Or simply integrate IMU measurements within an optimization objective, either as a prior, or additional residuals to optimize.

Filtering Approaches Filtering approaches maintain only a single state which is updated constantly with new measurements. The seminal approach to probabilistic state estimation is the Kalman filter [53]. In this work, the state estimation is separated between a prediction state (based on the model), and an update step which corrects the model given new measurements, by comparing the predictions to the measurements.

For geometric pose estimation problems, the relationship between the model and measurements is not linear. Thus extensions of the Kalman filter are preferable, such as the extended Kalman filter (EKF) using first-order linearization, Unscented/Sigmapoint Kalman Filter (SKF) which, or iterative approaches which perform the update step in multiple iterations. An in-depth introduction to the subject can be found in [3].

The filtering approach has been widely used in recent years to build LiDAR Inertial odometries [91, 127, 125, 126, 70, 51].

LINS [91] is built on the LOAM framework, but proposes a filtering approach based on the Error State Kalman Filter (ESKF). In this work, each IMU measurement is used in a *predict* step to estimate the trajectory at the frequency of the IMU. Then, the *update* step iteratively minimizes a least-square objective formulated as the standard LOAM objective (expressed with the error state parameters), to which a prior is added, expressing the uncertainty of the prediction. At each step, similarly to most LiDAR-Only odometries, the neighborhood association is updated.

Similarly, Fast-LIO [127, 125] and Faster-LIO [126] also use an iterative Extended Kalman Filter, using a tangent space linearization, and thus represent the uncertainty in the tangent

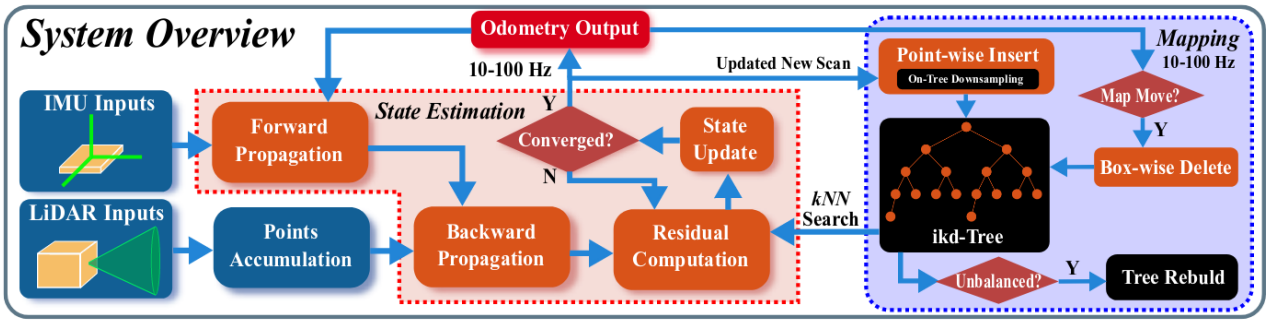


Figure 5.2: FAST-LIO2's [125] system overview .

space of the state estimate. The prediction is similar to [91], and at each iteration of the *update* step, the neighborhood association is updated, then an objective adding the prior residual to the sum of the measurements residuals is optimized to derive the optimal error-state, and propagate the uncertainty. The difference is that these latter methods use a dense point cloud registration procedure, with point-to-plane residuals constructed.

These approaches typically allow for very fast odometries. Note that they are very similar to the LiDAR-Only odometries methods proposed in section 3.2. However, the filtering approach allows them to better model the uncertainty, and thus construct better priors which significantly helps the performance of the algorithm. Thus, filtering approaches are essentially equivalent to LiDAR-Only odometries with good priors and better initializations.

Smoothing / Batch Optimization Approaches Filtering approaches, only optimize one state at a time. Smoothing approaches, by contrast, keep a memory of previous states, and optimizes the new state and the previous states simultaneously. In probabilistic formulation, filtering only estimates current states given past measurements while smoothing estimates each state given all the available measurements.

LIO-Mapping [130] proposes a new fixed-lag smoothing approach that keeps only a sliding window of states optimized simultaneously. The map estimation of the vector of states optimized together is estimated by the least-square minimization of a sum of LiDAR measurement residuals, prior residuals and imu residuals. The LiDAR residuals are constructed by associating features between each frame in the window and a pivot frame. The IMU residuals, similarly to [94], are constructed by pre-integrating IMU measurements between relevant time frames.

In2Laama [62], builds a global factor graph containing both LiDAR and IMU factors together. And optimizes all state variables in a full batch optimization.

Optimization-Constraints Lastly, IMU measurements can also be integrated directly within optimization objectives, as optimization constraints.

One category of method notably uses a continuous trajectory modelization to compare derivatives of the trajectory to IMU measurements. Using notably a representation with B-Splines (cf section 3.2) [97, 75] construct constraints on the derivatives of the trajectory, by comparing them to the raw IMU measurements. This allows them to skip the preintegration step, and thus constrain the set of knots optimized together.

An alternative approach is to use the pre-integrated measurements as constraints. This is notably the case of [15] which uses a pre-integrated gyroscope measurement as a prior for the registration of the scan.

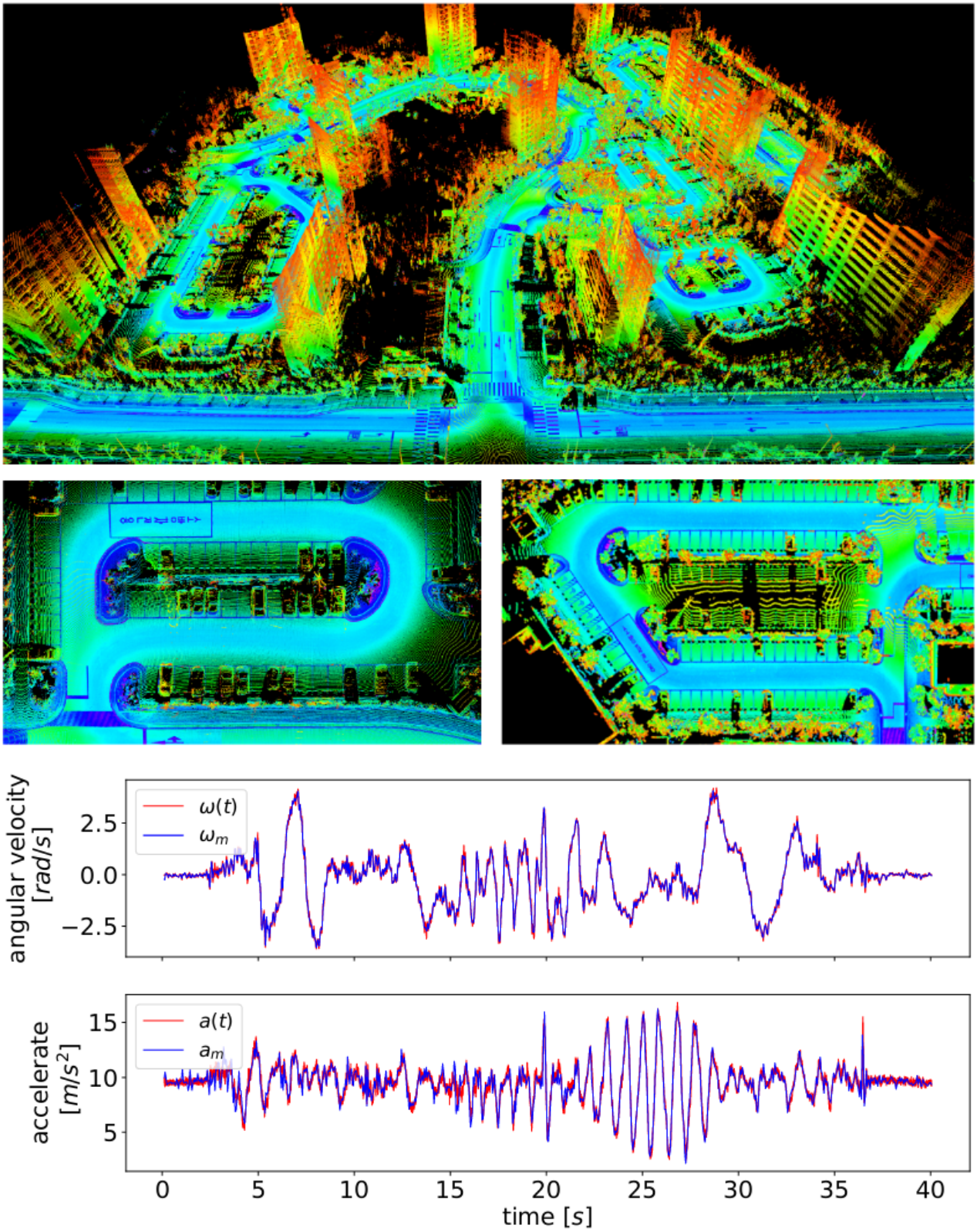


Figure 5.3: Aggregated point cloud reconstructed using CLINS [75] (Top). This LiDAR-Inertial Odometry uses B-Splines to estimate the trajectory. Using both LiDAR and IMU measurements, they can approximate very well the derivatives of the trajectory, as shown in the graph (Bottom), where the blue line corresponds to the model's angular velocity and linear acceleration, and the red line, the corresponding IMU measurements.

5.3 Improving CT-ICP with Inertial Measurements

We saw in the previous section, the different strategies to fuse IMU and LiDAR measurements existing in the related work. In this work, we propose to extend our LiDAR-Only odometry CT-ICP (see section 3.4) with a tightly-coupled, model-less optimization approach. More precisely, we propose two novel loosely-coupled extensions of CT-ICP using inertial measurements and compare them with the lidar-only method.

The section is organized as follows: first, we present the kinematics of the LiDAR-Inertial system 5.3.1. Then in section 5.3.2, we present the motivation for using inertial measurements in the context of LiDAR odometry, by studying failure cases of LiDAR-Only odometries. Finally, we present our proposed methods in section 5.3.3.

Our main contributions in this work are the proposed extensions of CT-ICP, and the demonstration in section 5.4 of their performance and improvement over CT-ICP.

5.3.1 IMU Measurement model / Notations

In this section, we describe the kinematics notations and the measurement model of the IMU sensor. We only work with 6-axis IMU sensors, which provide measurements of the linear acceleration and angular velocity of the sensor in the sensor frame.

To model the motion of the IMU sensor, we introduce the following notations:

$$\mathbf{T}_{imu}(t) := \text{The pose of the IMU sensor at time t in an inertial frame} \tag{5.1}$$
$$\mathbf{T}_{imu}(t) = (\mathbf{R}_{imu}(t), \mathbf{t}_{imu}(t)) \tag{5.2}$$
$$\mathbf{T}(t) := \text{The angular velocity of the IMU sensor at time t in the sensor frame} \tag{5.3}$$
$$\boldsymbol{a}(t) := \text{The linear acceleration of the IMU sensor at time t in the sensor frame} \tag{5.4}$$

The relationship between the motion derivatives and the pose is expressed as follows (see [3] for the derivation of the kinematics robotics applications):

$$\dot{\mathbf{R}}_{imu}(t) = \mathbf{R}_{imu}(t)\mathbf{T}^{\wedge}(t) \tag{5.5}$$
$$\dot{\mathbf{t}}_{imu}(t) = \mathbf{R}_{imu}(t)^T\boldsymbol{a}(t) + \boldsymbol{g} \tag{5.6}$$

IMU sensors provide at high-frequency noisy measurements of the angular velocity $\widetilde{\mathbf{T}}(t_i)$ and linear acceleration $\widetilde{\boldsymbol{a}}(t_i)$ of the sensor. The measurement noise is often modeled as a Gaussian with quantities $\boldsymbol{b}_\omega$ and $\boldsymbol{b}_a$ representing the bias of the sensor:

$$\widetilde{\mathbf{T}}(t_i) = \mathbf{T}(t_i) + \boldsymbol{b}_\omega \sim N(0, \boldsymbol{\Sigma}_\omega) \tag{5.7}$$
$$\widetilde{\boldsymbol{a}}(t_i) = \boldsymbol{a}(t_i) + \boldsymbol{b}_a \sim N(0, \boldsymbol{\Sigma}_a) \tag{5.8}$$

In this work, we only consider gyroscope measurements and ignore accelerometers altogether. Furthermore, we assume that the gyroscope bias is very small and can be neglected. We see the limits of this assumption in the next section, but we design the methods proposed in this chapter based on these assumptions.

We will often integrate gyroscope measurements, and we simply use Euler equations as follows, by assuming an angular velocity constant between t_i and t_{i+1}:

$$\mathbf{R}(t_{i+1}) = \mathbf{R}(t_i)\exp(\mathbf{T}(t_i)(t_{i+1} - t_i)) \tag{5.9}$$

5.3.2 Motivation for using inertial measurements

We already examined in chapter 3 failure cases for our LiDAR Odometry CT-ICP. There are different scenarios of failures, but the one we will detail in this section is due to the limitation of our constant-velocity motion model. CT-ICP's handling of the distortion relies on the constant-velocity motion model assumption. While this assumption is valid for most of the datasets we presented in chapter 2, as demonstrated by the results of section 3.5. However, when this assumption does not hold, badly distorted point clouds inserted in the map will pollute it and lead to catastrophic failures of the LiDAR Odometry.

The HILTI challenge datasets presented in section 2, will help us illustrate this quite clearly. These datasets provide centimetric-precision ground truth for multiple sequences. Using this ground truth, we can reconstruct the aggregated point cloud using different models of continuous trajectory to distort each scan inserted.

First, we use the constant-velocity model (which is the model assumed by CT-ICP), in this model the ground truth trajectory is sampled at the frequency of LiDAR frames; then we use a linear interpolation between the poses at the beginning and end of the frame to distort the point cloud, similar to the one proposed in section 3. We call this method of distortion **GT-CV**[1] :

$$\mathbf{p}^i_{raw} := \text{Point in the sensor frame} \tag{5.10}$$
$$t^i := \text{Timestamp of the point} \tag{5.11}$$
$$\mathbf{p}^i_{world} := \text{Point in the world frame} \tag{5.12}$$
$$(\mathbf{T}_b, \mathbf{T}_e) := \text{The poses at the beginning and end of the frame} \tag{5.13}$$
$$t^i_\alpha := (t^i - t_b)/(t_e - t_b) \tag{5.14}$$
$$\mathbf{R}^{CV}(t_i) = \text{Slerp}(\text{Quat}(\mathbf{T}_b), Quat(\mathbf{T}_e), t^i_\alpha) \tag{5.15}$$
$$\mathbf{t}^{CV}(t_i) = (1. - t^i_\alpha)\mathbf{t}_b + t^i_\alpha \mathbf{t}_e \tag{5.16}$$
$$\mathbf{p}^i_{world} = \mathbf{R}^{CV}(t_i)\mathbf{p}^i_{raw} + \mathbf{t}^{CV}(t_i) \tag{5.17}$$

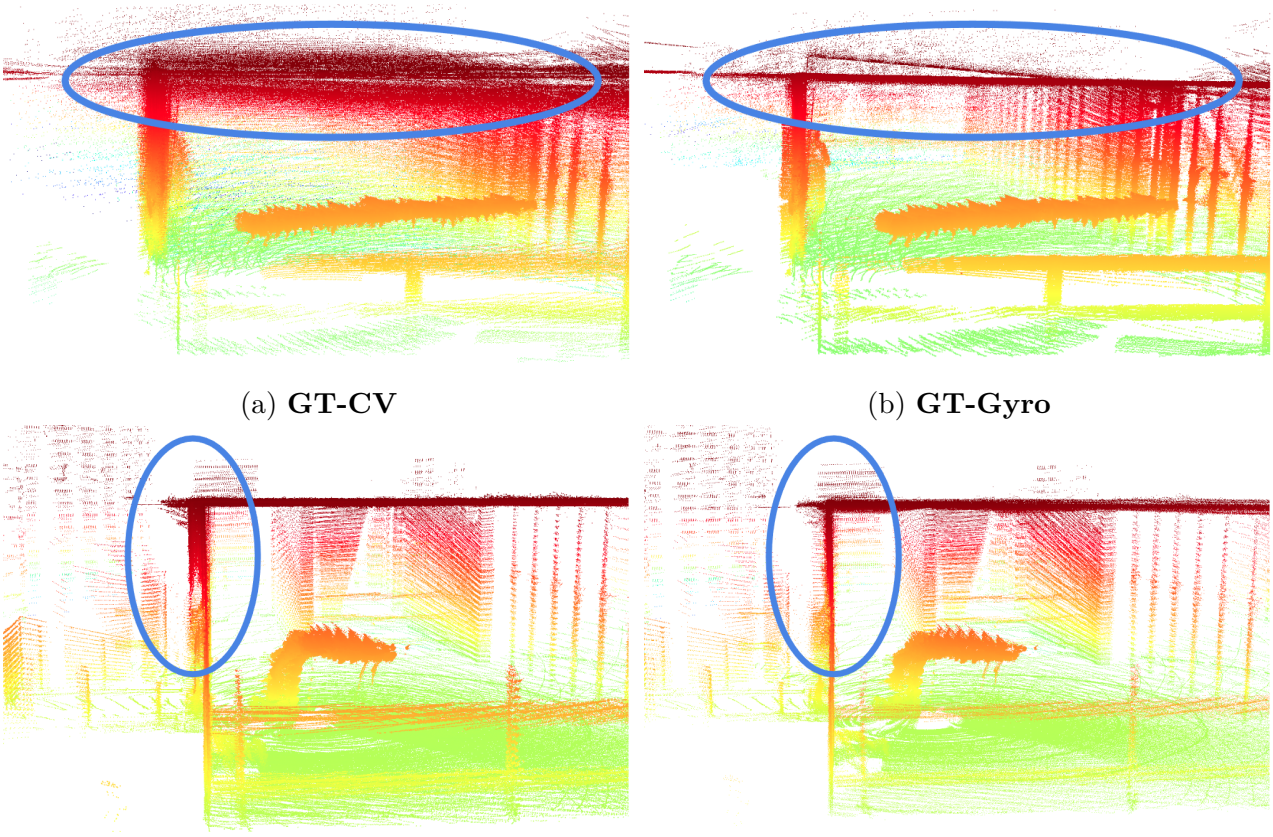
Then, we use the raw gyroscope measurements to distort the point cloud frames: as described in section equation 5.9, we integrate the gyroscope measurements to estimate poses at the frequency of the gyroscope. For a given frame, starting at time t_b and ending at time t_e, the estimate poses at the instants $\tau_0 = t_b, ..., \tau_k = t_e$ of the gyroscope measurements received by only integrating gyroscope measurements. Then, for each point, we interpolate the pose between the two temporally closest poses τ_i and τ_{i+1}, and transform the point in the world frame using this interpolated pose. We call this method of distortion **GT-Gyro**.

Finally, we reincorporate the constant velocity model for the translation to the **GT-Gyro** presented just above. This leads to the final distortion method **GT-Gyro-CT**.

In figure 5.4 we show the point cloud aggregated for these different methods on the sequence `exp06` of the HILTI 2022 dataset [136]. We can see that the Constant Velocity model (**GT-CV**) leads to highly polluted aggregated point clouds, compared to the other two. The HILTI 2022 handheld dataset is particularly challenging, and we selected a sequence where the carrier makes particularly abrupt motions. In this context, thus we see that the error due to the constant velocity model in terms of rotation estimation leads to the most significant distortion errors of the point cloud. This demonstrates, that even if we estimated perfectly the trajectory with our algorithm CT-ICP, we would still introduce significant map pollutions, to which we know our algorithm is particularly sensitive.

We proposed the **GT-Gyro** as it is often used in the literature to distort point clouds [112, 83], thus neglecting the distortion errors due to the translation between two frames. While

[1]Here **CV** applies to the modelization of the trajectory, not to be mistaken to the initialization method presented in chapters 3 and 4



(a) **GT-CV** (b) **GT-Gyro**

(c) **GT-Gyro** (d) **GT-Gyro-CT**

Figure 5.4: Comparison of the different distortion methods **GT-CV**, **GT-Gyro**, **GT-Gyro-CT** on the sequence `exp06` of the HILTI 2022 dataset. We first compare the aggregated point cloud obtained with **GT-CV** and **GT-Gyro** (Top), and then (for a different part of the trajectory), **GT-Gyro** and **GT-Gyro-CT** (Bottom). The figure shows that the constant velocity distortion leads in these cases to high levels of noise in the aggregated point cloud, even when using the ground truth (**GT-CV**). Using gyroscope (**GT-Gyro**) measurements helps reduce the noise of the point cloud, and using a linear interpolation of the translation (**GT-Gyro-CT**) helps even more.

this might be valid for certain contexts, figure 5.4 shows that neglecting this effect still leads to map pollution errors. In this sequence, this is particularly noticeable when the user puts the platform down to the ground or lifts it. For these two scenarios, the translation between two frames ranges between 20 to 30 cm over 3 or 4 frames. Neglecting to consider this distortion does lead to problematic pollution.

These observations have the benefit of making the roadmap for improving CT-ICP clear: *ie* using gyroscope measurements to pre-distort the point cloud and use the constant velocity model to distort the translation. We detail this approach in the next section.

5.3.3 Proposed methods

In this section, taking into consideration the observations of the previous section, we construct a model-less approach to improve CT-ICP. More precisely, we propose gradually more precise extensions following the motion models exposed in the previous section. This will allow us to compare these approaches in section 5.4, and confirm quantitatively the analysis presented above.

CT-ICP-Gyro Our starting point is our method CT-ICP presented in section 3.4. This method implements a constant velocity model and uses 12 degrees of freedom accompanied by constraints to refine the distortion during the optimization. Thus, a first natural extension, using the gyroscope measurements, is to propose a new set of constraints using the integrated gyroscope measurements.

More specifically, this method, which we call **CT-ICP-Gyro**, is different from **CT-ICP** from two aspects: initialization and trajectory constraint. First, the initialization is performed using the integrated gyroscope measurement to predict the relative rotation of the new frame:

$$\mathbf{T}_{b,ini}^{k+1} = \mathbf{T}_{e,ini}^{k} \tag{5.18}$$

$$\mathbf{R}_{e,ini}^{k+1} = \mathbf{T}_{b,ini}^{k+1} * \mathbf{R}_{gyro}^{k,k+1} \tag{5.19}$$

$$\mathbf{R}_{gyro}^{k,k+1} = \prod_{l=1..n} \exp[\mathbf{T}^{\wedge}(\tau_l) * (\tau_{l+1} - \tau_l)] \tag{5.20}$$

$$\mathbf{t}_{e,ini}^{k,k+1} = \mathbf{t}_{b,ini}^{k} \tag{5.21}$$

Then, we add a different set of constraints to the registration algorithm (recalling equation 3.39), which leads to the following registration error $E_{\text{Reg}}(\mathbf{X})$ to optimize at each iteration of the ICP:

$$\mathbf{X} = [\mathbf{T}_b, \mathbf{T}_e] = [(\mathbf{R}_b, \mathbf{t}_b), (\mathbf{R}_b, \mathbf{t}_b)] \in SE(3)^2 \tag{5.22}$$

$$E_{\text{Reg}}(\mathbf{X}) = E_{\text{CT-ICP}}(\mathbf{X}) + E_{\text{C}}(\mathbf{X}) \tag{5.23}$$

$$\begin{aligned} E_{\text{C}}(\mathbf{X}) = {} & \beta_{gyro} \cdot \|\mathbf{R}_b^T * \mathbf{R}_e * (\mathbf{R}_{gyro}^{k,k+1})^T - \mathbf{1}_3\|_F^2 \\ & + C_{\text{loc}}(\mathbf{X}) + C_{\text{vel}}(\mathbf{X}) \end{aligned} \tag{5.24}$$

Where $C_{\text{loc}}(\mathbf{X})$ and $C_{\text{vel}}(\mathbf{X})$ are the same constraints defined in equations 3.46 and 3.47, and β_{gyro} is a weight parameter. These constraints enforce the constant velocity model and add a prior to the estimation of the relative rotation.

As we showed in the previous section, this model might be invalid for abrupt rotations. Thus we propose next a popular approach: use the gyroscope measurements to pre-distort the point cloud.

ICP-DistGyro Following many popular approaches [112, 83], in this approach, we use only gyroscope measurements to apply a distortion of the point cloud as a preprocessing step. Importantly, we distort the point cloud (similarly to **GT-Gyro** presented in the previous section) at the frequency of the gyroscope measurements. This allows us to correct abrupt rotations much more precisely than the previous approach.

More precisely, we transform the point of the incoming frame, and express them in the reference frame of the end pose of the frame, as follows:

$$\mathbf{p}^i_{raw}(t_i) := \text{Point in the sensor frame at time } t_i \tag{5.25}$$

$$\mathbf{p}^{i,dist}_{raw}(t_i) = \mathbf{R}_e^T * \mathbf{R}_{gyro}(t_i) * \mathbf{p}^i_{raw}(t_i) \tag{5.26}$$

$$\mathbf{R}_{gyro}(t_i) = \text{Slerp}(\text{Quat}(\tau_k), \text{Quat}(\tau_{k+1}), \frac{t_i - \tau_k}{\tau_{k+1} - \tau_k}),\ t_i \in [\tau_k, \tau_{k+1}[\tag{5.27}$$

Then, our registration algorithm does not need to correct the distortion during the optimization. Thus, we only optimize the 6 degrees of freedom of a standard point-to-plane rigid transformation:

$$\mathbf{X} = \mathbf{T}_e = (\mathbf{R}_e, \mathbf{t}_e) \in SE(3) \tag{5.28}$$

We keep however the prior constraints on the relative rotation similar to equation 5.23.

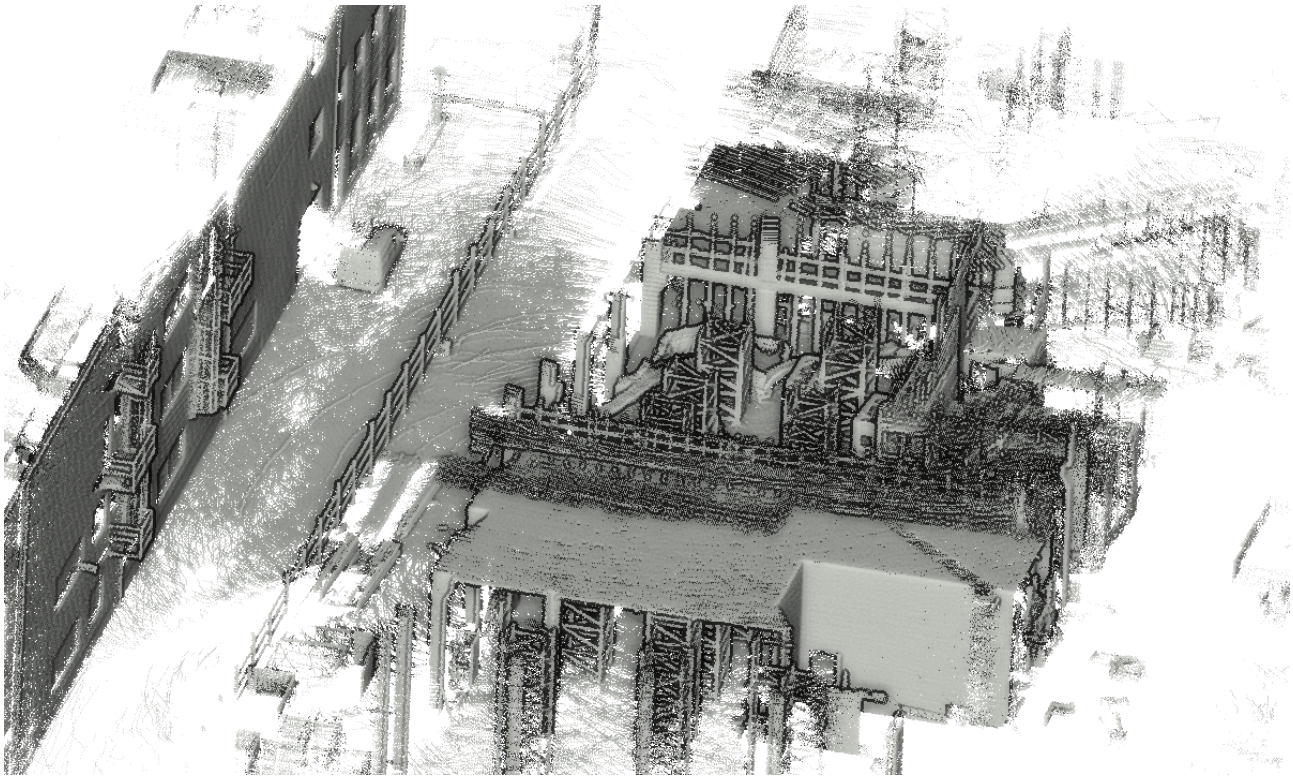
The main reason that this approach is popular, even among state-estimation models (such as [112]), is that angular velocity measurements, contrary to linear acceleration, do not need an absolute orientation estimation to be correctly integrated. This means that for a trajectory that has drifted in orientation, the prediction of the relative orientation will still be valid, while this might not be the case for the relative translation.

However, when the relative translation between two frames is significant, we saw that this approach still introduces noise due to bad distortions. Which motivates our last iteration.

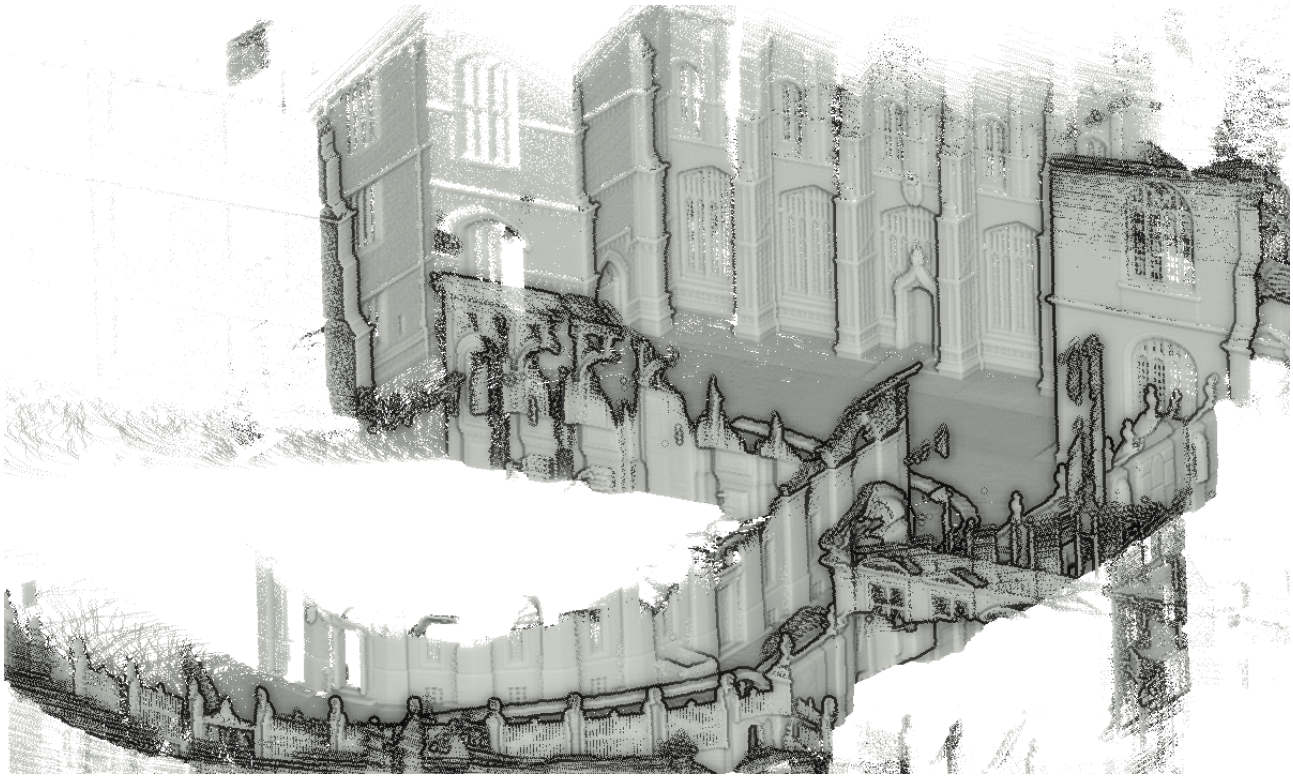
ICP-DistGyro-CT-Trans So finally, our final approach combines the two previous approaches. CT-ICP did prove the superior performance obtained using 12 degrees of freedom when the constant velocity model was valid, see section 3.5 for more details. Thus, compared to **ICP-DistGyro**, we reintroduce the 3 degrees of liberty which allow our ICP to estimate the distortion due to the translation during the optimization. Thus in total 9 parameters are optimized at each iteration of the ICP:

$$\mathbf{X} = (\mathbf{t}_b, \mathbf{R}_e, \mathbf{t}_e) \in \mathbb{R}^3 \times SE(3) \tag{5.29}$$

The rotation from the gyroscope is still applied as a preprocessing step, similar to the previous method. The initialization is also identical. In the next section, we demonstrate the gradual improvements each of these methods provides. But before this, we show in figure 5.5 point clouds aggregated using the **ICP-DistGyro-CT-Trans** method. They provide qualitative results of our LiDAR Inertial Odometry on the HILTI sequences.



(a) Aggregated point cloud using (**ICP-DistGyro-CT-Trans**). **Dataset:** `HILTI 2022, exp01`



(b) Aggregated point cloud using (**ICP-DistGyro-CT-Trans**). **Dataset:** `HILTI 2022, exp21`

Figure 5.5: Aggregated point clouds using our best LiDAR-Inertial Odometry, **ICP-DistGyro-CT-Trans**.

Figure 5.6: **ICP-DistGyro** (Blue), **ICP-DistGyro-CT-Trans**(Orange) and ground truth poses (Green). Dataset `HILTI-2022, exp06`

5.4 Experiments

In this work, we use the sequences from the NCLT [13], HILTI 2022 [136] and NCD [98] datasets, which were all presented in chapter 2. For the NCLT and NCD datasets, we use the Relative Pose Error (**RPE**) metric, as the acquisition is made within a large environment, thus with a trajectory prone to drift. For the HILTI dataset, however, we use the absolute trajectory error (**ATE**) because the sequences we consider are acquired in confined spaces, thus less prone to drift.

We present the trajectory errors for the three methods on these three datasets in figure 5.1. There are two aspects to focus on: first, the robustness of the method, ie whether it fails on some of the sequences. And, secondly, the overall precision achieved by the method.

The first thing to notice is that the default version of **CT-ICP** but also **CT-ICP-Gyro** fail on two sequences (HILTI-`exp06` and NCD-`05`). These two sequences were designed to be challenging, and the operator moves the handheld device abruptly, to test the robustness of SLAM methods. The sequence `exp06` is specifically the sequence presented in figure 5.4, and the failure does occur when the motion is so abrupt that the constant-velocity model does not distort the point clouds well enough. This leads to map pollution and then catastrophic failures. Thus, we see that the second class of method *ie* **ICP-GyroDist** and **ICP-GyroDist-CT-Trans** is more robust than using the constant velocity model of CT-ICP. Performing the distortion as a preprocessing step here does correct the distortion of the point cloud at a high enough frequency that no bad insertions are performed in the map. Thus, these two methods, provided that the accuracy of the gyroscope is satisfying, are more robust than CT-ICP to dynamic motions.

Table 5.1: Trajectory errors for the different improvements of CT-ICP on a set of sequences from LiDAR-Inertial datasets.

NCLT (RPE(%) per sequence)			
	2012-01-08	2012-04-29	2012-05-11
CT-ICP	0.89	0.87	0.92
CT-ICP-Gyro	**0.87**	0.89	0.90
ICP-GyroDist	0.87	0.87	0.89
ICP-GyroDist-CT-Trans	0.88	**0.86**	**0.87**
HILTI-2022 (ATE(m) per sequence)			
	exp04	exp05	exp06
CT-ICP	0.08	0.09	✗
CT-ICP-Gyro	0.07	0.08	✗
ICP-GyroDist	0.06	0.07	0.06
ICP-GyroDist-CT-Trans	**0.04**	**0.05**	**0.05**
NCD (RPE(%) per sequence)			
	01	02	05
CT-ICP	1.13	1.21	✗
CT-ICP-Gyro	**1.10**	1.15	✗
ICP-GyroDist	1.12	1.16	1.32
ICP-GyroDist-CT-Trans	1.11	**1.13**	**1.27**

Secondly, looking at the performance of each method, we first notice that the gain between each layer of complexity, though it is noticeable, is marginal. **CT-ICP** is already by itself a very precise odometry, and adding the gyroscope for the distortion does improve the robustness of the method as it prevents failure cases, but does little to improve the precision.

Comparing **ICP-GyroDist** to **ICP-GyroDist-CT-Trans**, we see that again if the gain of the latter is still marginal, it is more noticeable on the **HILTI 2022** dataset. This is because the method better handles the translation distortion occurring when the sensor is lifted from or put down on the ground. Still, the gain is only 1 or 2 cm on average over the trajectory.

Finally, inspecting the precision of CT-ICP on **HILTI-2022**, where the acquisition is performed in a confined area (see figure 5.4), we see that the range of the absolute error is 5 cm on average. While this is a high level of precision, professional solutions obtained 2 cm average errors on the HILTI-2022 benchmark.

We conjecture that the road to higher levels of precision will come from combining both geometry and more sophisticated motion models. That is, in our current approach, we do not model the noise of the sensor (which typically ranges between 2 and 5 cm for the LiDAR we presented). This noise is added to the map, and we neither refine the model of the surface of the map nor model this noise in our optimization. Thus, to leverage more complicated trajectory models (such as B-Spline or Iterative Kalman filters), we believe that strong geometric constraints are needed, with finer management of the noise, both at the map level and for the objective of the registration minimized.

5.5 Conclusion

In this chapter, we presented an extension of our method CT-ICP (see chapter 3), to integrate IMU measurements. Compared to different approaches, presented in section 5.2, our method uses a model-less and naive approach. However, we showed that despite its simplicity, this approach does improve CT-ICP, and helps compensate for one of the shortcomings presented above: the limitations of the constant velocity model.

We took a layered approach where each layer slightly improves our base LiDAR-Odometry, CT-ICP. Yet, we saw that the principal gain of these approaches is in robustness, rather than in precision (as the precision gains were marginal). This illustrates again that CT-ICP is a solid backbone for precise odometry.

As we saw in the related work, many different strategies exist to fuse LiDAR with IMU measurements, and most could easily be integrated within CT-ICP to try and improve it even further, and we let this for future work. Another strategy would be to set the frequency of the LiDAR at 20Hz (which is available by most 3D LiDAR). This would reduce greatly the need for IMU measurements, such that the standard CT-ICP would remain a valid option.

This work is part of a publication in preparation, where we extend and improve CT-ICP, as previously mentioned in the conclusion of Chapter 3.

Conclusion

The work in this thesis proposed a panoramic view of LiDAR Odometry methods. The presented contributions aimed to clarify, and build upon the excellent previous work which was, and continues to be produced in this area. To achieve this goal, we presented a range of methods of LiDAR odometries, from classical to Deep-Learning based, and also a step toward the fusion with inertial sensors. We aimed to focus on producing profound analyses, rather than producing obscure novelties, to build a solid foundation, on which such novelties could be built later if desired.

In Chapter 2 of this thesis, we presented evaluation methods which we used in the rest of this thesis to evaluate and compare different LiDAR odometries. Then, in Chapter 3, we presented a classical LiDAR odometry pipeline, implemented in `pyLiDAR-SLAM`, including a naive, but precise point-to-plane, which achieves near state-of-the-art precision on public benchmarks. We also produced a novel state-of-the-art LiDAR odometry, *CT-ICP* which sets a novel state-of-the-art level of precision on public benchmarks. This work was analyzed in depth, on a wide variety of datasets, and we produced an extensive ablation study to understand the impact of each contribution on the overall performance of the method. In Chapter 4, we investigated deep and hybrid LiDAR odometries. More precisely, we focused on Deep end-to-end LiDAR odometries, and their relevance compared to classical approaches, both by themselves or as part of a hybrid LiDAR odometry pipeline. We showed that, as of today, such methods show no real relevance compared to classical LiDAR odometries. We repeat here, that we do not disavow Deep Learning in general in the context of LiDAR SLAM, and believe that it does have a place. Finally, in chapter 5, we proposed a natural extension of our reference odometry, *CT-ICP*, by incorporating Inertial measurements. We showed in this work, that we could improve *CT-ICP*, notably in terms of robustness using a better distortion model thanks to gyroscope measurements. This approach, though naive, and only a first step towards making a proper LiDAR-Inertial Odometry out of *CT-ICP*, is promising, and illustrates even further the precision and potential of our work *CT-ICP*.

The content of this work has already been published in two international conferences, and a third publication is in preparation at the time of this writing. Much of this work has been released as open-source: first `pyLiDAR-SLAM`[1] a python module for LiDAR odometry, and then *CT-ICP* [2]. Throughout this thesis, we have made consistent efforts to improve the quality of this work, which has been well-received by the robotics community. Multiple works have been built on top of, or motivated by *CT-ICP* ([121]).

[1] `https://github.com/Kitware/pyLiDAR-SLAM`

[2] `https://github.com/jedeschaud/ct_icp`

Bibliography

[1] Debaditya Acharya, Kourosh Khoshelham, and Stephan Winter. Bim-posenet: Indoor camera localisation using a 3d indoor model and deep learning from synthetic images. *ISPRS journal of photogrammetry and remote sensing*, 150:245–258, 2019.

[2] Hernan Badino, Daniel Huber, Yongwoon Park, and Takeo Kanade. Fast and accurate computation of surface normals from range images. In *2011 IEEE International Conference on Robotics and Automation*, pages 3084–3091. IEEE, 2011.

[3] Timothy D Barfoot. *State estimation for robotics*. Cambridge University Press, 2017.

[4] Jens Behley and Cyrill Stachniss. Efficient surfel-based slam using 3d laser range data in urban environments. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[5] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[6] Igor Bogoslavskyi and Cyrill Stachniss. Fast range image-based segmentation of sparse 3d laser scans for online operation. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 163–169. IEEE, 2016.

[7] Dorit Borrmann, Jan Elseberg, Kai Lingemann, and Andreas Nüchter. The 3d hough transform for plane detection in point clouds: A review and a new accumulator design. *3D Research*, 2(2):1–13, 2011.

[8] Michael Bosse and Robert Zlot. Continuous 3d scan-matching with a spinning 2d laser. In *2009 IEEE International Conference on Robotics and Automation*, pages 4312–4319, 2009.

[9] Michael Bosse and Robert Zlot. Place recognition using keypoint voting in large 3d lidar datasets. In *2013 IEEE International Conference on Robotics and Automation*, pages 2677–2684, 2013.

[10] Michael Bosse, Robert Zlot, and Paul Flick. Zebedee: Design of a spring-mounted 3-d range sensor with application to mobile mapping. *IEEE Transactions on Robotics*, 28(5):1104–1119, 2012.

[11] Michael Bosse, Robert Zlot, and Paul Flick. Zebedee: Design of a spring-mounted 3-d range sensor with application to mobile mapping. *IEEE Transactions on Robotics*, 28(5):1104–1119, 2012.

[12] Yixi Cai, Wei Xu, and Fu Zhang. ikd-tree: An incremental kd tree for robotic applications. *arXiv preprint arXiv:2102.10808*, 2021.

[13] Nicholas Carlevaris-Bianco, Arash K. Ushani, and Ryan M. Eustice. University of Michigan North Campus long-term vision and lidar dataset. *International Journal of Robotics Research*, 35(9):1023–1035, 2015.

[14] Andrea Censi, Luca Iocchi, and Giorgio Grisetti. Scan matching in the hough domain. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 2739–2744. IEEE, 2005.

[15] Kenny Chen, Brett T Lopez, Ali-akbar Agha-mohammadi, and Ankur Mehta. Direct lidar odometry: Fast localization with dense point clouds. *IEEE Robotics and Automation Letters*, 7(2):2000–2007, 2022.

[16] Shoubin Chen, Hao Ma, Changhui Jiang, Baoding Zhou, Weixing Xue, Zhenzhong Xiao, and Qingquan Li. Ndt-loam: A real-time lidar odometry and mapping with weighted ndt and lfa. *IEEE Sensors Journal*, 22(4):3660–3671, 2021.

[17] Xieyuanli Chen, Andres Milioto, Emanuele Palazzolo, Philippe Giguère, Jens Behley, and Cyrill Stachniss. Suma++: Efficient lidar-based semantic slam. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4530–4537, 2019.

[18] Yu Chen, Chunhua Shen, Xiu-Shen Wei, Lingqiao Liu, and Jian Yang. Adversarial posenet: A structure-aware convolutional network for human pose estimation. In *Proceedings of the IEEE international conference on computer vision*, pages 1212–1221, 2017.

[19] Younggun Cho, Giseop Kim, and Ayoung Kim. Deeplo: Geometry-aware deep lidar odometry. *arXiv preprint arXiv:1902.10562*, 2019.

[20] Younggun Cho, Giseop Kim, and Ayoung Kim. Unsupervised geometry-aware deep lidar odometry. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 2145–2152. IEEE, 2020.

[21] Younggun Cho, Giseop Kim, and Ayoung Kim. Unsupervised geometry-aware deep lidar odometry. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2145–2152, 2020.

[22] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[23] Frank Dellaert, Richard Roberts, Varun Agrawal, Alex Cunningham, Chris Beall, Duy-Nguyen Ta, Fan Jiang, lucacarlone, nikai, Jose Luis Blanco-Claraco, Stephen Williams, ydjian, John Lambert, Andy Melim, Zhaoyang Lv, Akshay Krishnan, Jing Dong, Gerry Chen, Krunal Chande, balderdash devil, DiffDecisionTrees, Sungtae An, mpaluri, Ellon Paiva Mendes, Mike Bosse, Akash Patel, Ayush Baid, Paul Furgale, matthewbroadwaynavenio, and roderick koehle. borglab/gtsam, May 2022.

[24] Pierre Dellenbach, Jean-Emmanuel Deschaud, Bastien Jacquet, and François Goulette. What's in my lidar odometry toolbox? In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4429–4436, 2021.

[25] Pierre Dellenbach, Jean-Emmanuel Deschaud, Bastien Jacquet, and François Goulette. Ct-icp: Real-time elastic lidar odometry with loop closure. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5580–5586, 2022.

[26] Jean-Emmanuel Deschaud. Imls-slam: Scan-to-model matching based on 3d data. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2480–2485, 2018.

[27] Jean-Emmanuel Deschaud. Kitti-carla: a kitti-like dataset generated by carla simulator. *arXiv e-prints*, page arXiv:2109.00892, August 2021.

[28] Luca Di Giammarino, Emanuele Giacomini, Leonardo Brizi, Omar Salem, and Giorgio Grisetti. Photometric lidar and rgb-d bundle adjustment. *IEEE Robotics and Automation Letters*, 2023.

[29] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 13–15 Nov 2017.

[30] David Droeschel and Sven Behnke. Efficient continuous-time slam for 3d lidar-based online mapping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5000–5007. IEEE, 2018.

[31] David Droeschel, Jörg Stückler, and Sven Behnke. Local multi-resolution representation for 6d motion estimation and mapping with a continuously rotating 3d laser scanner. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5221–5226. IEEE, 2014.

[32] Renaud Dubé, Hannes Sommer, Abel Gawel, Michael Bosse, and Roland Siegwart. Non-uniform sampling strategies for continuous correction based trajectory estimation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4792–4798. IEEE, 2016.

[33] Jan Elseberg, Dorit Borrmann, and Andreas Nüchter. One billion points in the cloud–an octree for efficient processing of 3d laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing*, 76:76–88, 2013.

[34] Epic Games. Unreal engine.

[35] Gonzalo Ferrer. Eigen-factors: Plane estimation for multi-frame and time-continuous point cloud alignment. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1278–1284. IEEE, 2019.

[36] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[37] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. 07 2015.

[38] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[39] Matteo Frosi and Matteo Matteucci. Art-slam: Accurate real-time 6dof lidar slam. *IEEE Robotics and Automation Letters*, 7(2):2692–2699, 2022.

[40] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[41] Patrick Geneva, Kevin Eckenhoff, Yulin Yang, and Guoquan Huang. Lips: Lidar-inertial 3d plane slam. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 123–130. IEEE, 2018.

[42] Adrian Haarbach, Tolga Birdal, and Slobodan Ilic. Survey of higher order rigid body motion interpolation methods for keyframe animation and continuous-time trajectory estimation. In *2018 International Conference on 3D Vision (3DV)*, pages 381–389. IEEE, 2018.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[44] Michael Helmberger, Kristian Morin, Beda Berner, Nitish Kumar, Giovanni Cioffi, and Davide Scaramuzza. The hilti slam challenge dataset. *IEEE Robotics and Automation Letters*, 7(3):7518–7525, 2022.

[45] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[46] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[47] Xingliang Ji, Lin Zuo, Changhua Zhang, and Yu Liu. Lloam: Lidar odometry and mapping with loop-closure detection based correction. In *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 2475–2480. IEEE, 2019.

[48] Jianwen Jiang, Jikai Wang, Peng Wang, and Zonghai Chen. Pou-slam: Scan-to-model matching based on 3d voxels. *Applied Sciences*, 9(19):4147, 2019.

[49] Lu Jie, Zhi Jin, Jinping Wang, Letian Zhang, and Xiaojun Tan. A slam system with direct velocity estimation for mechanical and solid-state lidars. *Remote Sensing*, 14(7):1741, 2022.

[50] Nikhil Jonnavithula, Yecheng Lyu, and Ziming Zhang. Lidar odometry methodologies for autonomous driving: A survey. *arXiv preprint arXiv:2109.06120*, 2021.

[51] Gilmar P Cruz Júnior, Adriano MC Rezende, Victor RF Miranda, Rafael Fernandes, Héctor Azpúrua, Armando A Neto, Gustavo Pessin, and Gustavo M Freitas. Ekf-loam: an adaptive fusion of lidar slam with wheel odometry and inertial data for confined spaces with few geometric features. *IEEE Transactions on Automation Science and Engineering*, 19(3):1458–1471, 2022.

[52] Michael Kaess. Simultaneous localization and mapping with infinite planes. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4605–4611. IEEE, 2015.

[53] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[54] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[55] Myoung-Jun Kim, Myung-Soo Kim, and Sung Yong Shin. A general construction scheme for unit quaternion curves with simple high order derivatives. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 369–376, 1995.

[56] Kenji Koide, Jun Miura, and Emanuele Menegatti. A portable three-dimensional lidar-based system for long-term and wide-area people behavior measurement. *International Journal of Advanced Robotic Systems*, 16(2):1729881419841532, 2019.

[57] Kenji Koide, Masashi Yokozuka, Shuji Oishi, and Atsuhiko Banno. Globally consistent 3d lidar mapping with gpu-accelerated gicp matching cost factors. *IEEE Robotics and Automation Letters*, 6(4):8591–8598, 2021.

[58] Kenji Koide, Masashi Yokozuka, Shuji Oishi, and Atsuhiko Banno. Voxelized gicp for fast and accurate 3d point cloud registration. EasyChair Preprint no. 2703, EasyChair, 2020.

[59] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss, and Alexander Kleiner. On measuring the accuracy of slam algorithms. *Autonomous Robots*, 27:387–407, 2009.

[60] Tilman Kühner and Julius Kümmerle. Large-scale volumetric scene reconstruction using lidar. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6261–6267, 2020.

[61] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011.

[62] Cedric Le Gentil, Teresa Vidal-Calleja, and Shoudong Huang. In2lama: Inertial lidar localisation and mapping. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6388–6394. IEEE, 2019.

[63] Sheng-Wei Lee, Chih-Ming Hsu, Ming-Che Lee, Yuan-Ting Fu, Fetullah Atas, and Augustine Tsai. Fast point cloud feature extraction for real-time slam. In *2019 International Automatic Control Conference (CACS)*, pages 1–6. IEEE, 2019.

[64] Kailai Li, Meng Li, and Uwe D. Hanebeck. Towards high-performance solid-state-lidar-inertial odometry and mapping. *IEEE Robotics and Automation Letters*, 6(3):5167–5174, 2021.

[65] Qing Li, Shaoyang Chen, Cheng Wang, Xin Li, Chenglu Wen, Ming Cheng, and Jonathan Li. Lo-net: Deep real-time lidar odometry. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8473–8482, 2019.

[66] Qing Li, Shaoyang Chen, Cheng Wang, Xin Li, Chenglu Wen, Ming Cheng, and Jonathan Li. Lo-net: Deep real-time lidar odometry. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8465–8474, 2019.

[67] Zhichao Li and Naiyan Wang. Dmlo: Deep matching lidar odometry. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6010–6017, 2020.

[68] Yiyi Liao, Jun Xie, and Andreas Geiger. KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *arXiv preprint arXiv:2109.13410*, 2021.

[69] Jiarong Lin and Fu Zhang. Loam livox: A fast, robust, high-precision lidar odometry and mapping package for lidars of small fov. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3126–3131, 2020.

[70] Jiarong Lin, Chunran Zheng, Wei Xu, and Fu Zhang. R ˆ2 live: A robust, real-time, lidar-inertial-visual tightly-coupled state estimator and mapping. *IEEE Robotics and Automation Letters*, 6(4):7469–7476, 2021.

[71] Xiao Liu, Lei Zhang, Shengran Qin, Daji Tian, Shihan Ouyang, and Chu Chen. Optimized loam using ground plane constraints and segmatch-based loop detection. *Sensors*, 19(24):5419, 2019.

[72] Zheng Liu and Fu Zhang. Balm: Bundle adjustment for lidar mapping. *IEEE Robotics and Automation Letters*, 6(2):3184–3191, 2021.

[73] Kok-Lim Low. Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, 4(10):1–3, 2004.

[74] Lun Luo, Si-Yuan Cao, Bin Han, Hui-Liang Shen, and Junwei Li. Bvmatch: Lidar-based place recognition using bird's-eye view images. *IEEE Robotics and Automation Letters*, 6(3):6076–6083, 2021.

[75] Jiajun Lv, Kewei Hu, Jinhong Xu, Yong Liu, Xiushui Ma, and Xingxing Zuo. Clins: Continuous-time trajectory estimation for lidar-inertial system. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6657–6663. IEEE, 2021.

[76] Martin Magnusson, Achim Lilienthal, and Tom Duckett. Scan registration for autonomous mining vehicles using 3d-ndt. *Journal of Field Robotics*, 24(10):803–827, 2007.

[77] Gyeongsik Moon, Ju Yong Chang, and Kyoung Mu Lee. V2v-posenet: Voxel-to-voxel prediction network for accurate 3d hand and human pose estimation from a single depth map. In *Proceedings of the IEEE conference on computer vision and pattern Recognition*, pages 5079–5088, 2018.

[78] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Trans. Graph.*, 32(6), November 2013.

[79] Farzan Erlik Nowruzi, Dhanvin Kolhatkar, Prince Kapoor, and Robert Laganiere. Point cloud based hierarchical deep odometry estimation. *arXiv preprint arXiv:2103.03394*, 2021.

[80] Julian Nubert, Shehryar Khattak, and Marco Hutter. Self-supervised learning of lidar odometry for robotic applications. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9601–9607. IEEE, 2021.

[81] Andreas Nuchter, Kai Lingemann, and Joachim Hertzberg. Cached kd tree search for icp algorithms. In *Sixth International Conference on 3-D Digital Imaging and Modeling (3DIM 2007)*, pages 419–426. IEEE, 2007.

[82] Andreas Nüchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. 6d slam—3d mapping outdoor environments. *Journal of Field Robotics*, 24(8-9):699–722, 2007.

[83] Matteo Palieri, Benjamin Morrell, Abhishek Thakur, Kamak Ebadi, Jeremy Nash, Arghya Chatterjee, Christoforos Kanellakis, Luca Carlone, Cataldo Guaragnella, and A. Aghamohammadi. LOCUS - A Multi-Sensor Lidar-Centric Solution for High-Precision Odometry and 3D Mapping in Real-Time. *IEEE Robotics and Automation Letters*, 6(2):421–428, 2020.

[84] Yue Pan, Pengchuan Xiao, Yujie He, Zhenlei Shao, and Zesong Li. Mulls: Versatile lidar slam via multi-metric linear least square. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[85] Gaurav Pandey, James R McBride, and Ryan M Eustice. Ford campus vision and lidar data set. *The International Journal of Robotics Research*, 30(13):1543–1552, 2011.

[86] Chanoh Park, Soohwan Kim, Peyman Moghadam, Clinton Fookes, and Sridha Sridharan. Probabilistic surfel fusion for dense lidar mapping. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2418–2426, 2017.

[87] Chanoh Park, Peyman Moghadam, Soohwan Kim, Alberto Elfes, Clinton Fookes, and Sridha Sridharan. Elastic lidar fusion: Dense map-centric continuous-time slam. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1206–1213, 2018.

[88] Soon-Yong Park and Murali Subbarao. An accurate and fast point-to-plane registration technique. *Pattern Recognition Letters*, 24(16):2967–2976, 2003.

[89] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[90] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[91] Chao Qin, Haoyang Ye, Christian E. Pranata, Jun Han, Shuyang Zhang, and Ming Liu. Lins: A lidar-inertial state estimator for robust and efficient navigation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8899–8906, 2020.

[92] Kaihuai Qin. General matrix representations for b-splines. In *Proceedings Pacific Graphics' 98. Sixth Pacific Conference on Computer Graphics and Applications (Cat. No. 98EX208)*, pages 37–43. IEEE, 1998.

[93] Peilin Qin, Chuanwei Zhang, Xiaowen Ma, and Zhenghe Shi. High-precision motion compensation for lidar based on lidar odometry. *Wireless Communications and Mobile Computing*, 2022, 2022.

[94] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[95] Jan Quenzel and Sven Behnke. Real-time multi-adaptive-resolution-surfel 6d lidar odometry using continuous-time trajectory optimization. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5499–5506. IEEE, 2021.

[96] Jan Quenzel and Sven Behnke. Real-time multi-adaptive-resolution-surfel 6d lidar odometry using continuous-time trajectory optimization. *arXiv e-prints*, page arXiv:2105.02010, May 2021.

[97] Milad Ramezani, Kasra Khosoussi, Gavin Catt, Peyman Moghadam, Jason Williams, Paulo Borges, Fred Pauling, and Navinda Kottege. Wildcat: Online continuous-time 3d lidar-inertial slam, 2022.

[98] Milad Ramezani, Yiduo Wang, Marco Camurri, David Wisth, Matias Mattamala, and Maurice Fallon. The newer college dataset: Handheld lidar, inertial and vision with ground truth. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4353–4360, 2020.

[99] Andrzej Reinke, Matteo Palieri, Benjamin Morrell, Yun Chang, Kamak Ebadi, Luca Carlone, and Ali-Akbar Agha-Mohammadi. Locus 2.0: Robust and computationally efficient lidar odometry for real-time 3d mapping. *IEEE Robotics and Automation Letters*, pages 1–8, 2022.

[100] Zhuli Ren, Liguan Wang, and Lin Bi. Robust gicp-based 3d lidar slam for underground mining environment. *Sensors*, 19(13):2915, 2019.

[101] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[102] Nivedita Rufus, Unni Krishnan R. Nair, A. V. S. Sai Bhargav Kumar, Vashist Madiraju, and K. Madhava Krishna. Srom: Simple real-time odometry and mapping using lidar data for autonomous vehicles. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1867–1872, 2020.

[103] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation*, pages 1–4. IEEE, 2011.

[104] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4938–4947, 2020.

[105] Torsten Sattler, Qunjie Zhou, Marc Pollefeys, and Laura Leal-Taixe. Understanding the limitations of cnn-based absolute camera pose regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3302–3312, 2019.

[106] Mark Schadler, Jörg Stückler, and Sven Behnke. Multi-resolution surfel mapping and real-time pose tracking using a continuously rotating 2d laser scanner. In *2013 ieee international symposium on safety, security, and rescue robotics (ssrr)*, pages 1–6. IEEE, 2013.

[107] Markus Schütz, Stefan Ohrhallinger, and Michael Wimmer. Fast out-of-core octree generation for massive point clouds. In *Computer Graphics Forum*, volume 39, pages 155–167. Wiley Online Library, 2020.

[108] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics: science and systems*, volume 2, page 435. Seattle, WA, 2009.

[109] Jacopo Serafin and Giorgio Grisetti. Nicp: Dense normal based point cloud registration. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 742–749. IEEE, 2015.

[110] James Servos and Steven L Waslander. Multi channel generalized-icp. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3644–3649. IEEE, 2014.

[111] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765, 2018.

[112] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Rus Daniela. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5135–5142. IEEE, 2020.

[113] Christiane Sommer, Vladyslav Usenko, David Schubert, Nikolaus Demmel, and Daniel Cremers. Efficient derivative computation for cumulative b-splines on lie groups. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11148–11156, 2020.

[114] Hartmut Surmann, Andreas Nüchter, and Joachim Hertzberg. An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments. *Robotics and Autonomous Systems*, 45(3-4):181–198, 2003.

[115] Andrea Tagliabue, Jesus Tordesillas, Xiaoyi Cai, Angel Santamaria-Navarro, Jonathan P How, Luca Carlone, and Ali-akbar Agha-mohammadi. Lion: Lidar-inertial observability-aware navigator for vision-denied environments. In *Experimental Robotics: The 17th International Symposium*, pages 380–390. Springer, 2021.

[116] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420, 2019.

[117] Yiming Tu. Unpwc-svdlo: Multi-svd on pointpwc for unsupervised lidar odometry. *arXiv preprint arXiv:2205.08150*, 2022.

[118] Yiming Tu and Jin Xie. Undeeplio: Unsupervised deep lidar-inertial odometry. In *Pattern Recognition: 6th Asian Conference, ACPR 2021, Jeju Island, South Korea, November 9–12, 2021, Revised Selected Papers, Part II*, pages 189–202. Springer, 2022.

[119] Abhinav Valada, Noha Radwan, and Wolfram Burgard. Deep auxiliary learning for visual localization and odometry. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6939–6946. IEEE, 2018.

[120] I. Vizzo, X. Chen, N. Chebrolu, J. Behley, and C. Stachniss. Poisson surface reconstruction for lidar odometry and mapping. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[121] Ignacio Vizzo, Tiziano Guadagnino, Benedikt Mersch, Louis Wiesmann, Jens Behley, and Cyrill Stachniss. KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way. *arXiv preprint*, 2022.

[122] Guangming Wang, Xinrui Wu, Zhe Liu, and Hesheng Wang. Pwclo-net: Deep lidar odometry in 3d point clouds using hierarchical embedding mask optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15910–15919, 2021.

[123] H. Wang, C. Wang, C. Chen, and L. Xie. F-loam: Fast lidar odometry and mapping. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[124] Yuchen Wu, David J Yoon, Keenan Burnett, Soeren Kammel, Yi Chen, Heethesh Vhavle, and Timothy D Barfoot. Picking up speed: Continuous-time lidar-only odometry using doppler velocity measurements. *arXiv preprint arXiv:2209.03304*, 2022.

[125] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. Fast-lio2: Fast direct lidar-inertial odometry. *IEEE Transactions on Robotics*, 2022.

[126] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. Fast-lio2: Fast direct lidar-inertial odometry. *IEEE Transactions on Robotics*, 2022.

[127] Wei Xu and Fu Zhang. Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter. *IEEE Robotics and Automation Letters*, 6(2):3317–3324, 2021.

[128] Yan Xu, Zhaoyang Huang, Kwan-Yee Lin, Xinge Zhu, Jianping Shi, Hujun Bao, Guofeng Zhang, and Hongsheng Li. Selfvoxelo: Self-supervised lidar odometry with voxel-based deep neural networks. In *Conference on Robot Learning*, pages 115–125. PMLR, 2021.

[129] Zongxin Yang, Xin Yu, and Yi Yang. Dsc-posenet: Learning 6dof object pose estimation via dual-scale consistency. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3907–3916, 2021.

[130] Haoyang Ye, Yuying Chen, and Ming Liu. Tightly coupled 3d lidar inertial odometry and mapping. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3144–3150. IEEE, 2019.

[131] Masashi Yokozuka, Kenji Koide, Shuji Oishi, and Atsuhiko Banno. Litamin: Lidar-based tracking and mapping by stabilized icp for geometry approximation with normal distributions. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5143–5150. IEEE, 2020.

[132] Masashi Yokozuka, Kenji Koide, Shuji Oishi, and Atsuhiko Banno. Litamin2: Ultra light lidar-based slam using geometric approximation applied with kl-divergence. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11619–11625, 2021.

[133] Zhu Youji, Chen Qijun, Zhang Hao, Li Dairong, and Wei Penghao. A slam method based on loam for ground vehicles in the flat ground. In *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*, pages 546–551, 2019.

[134] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: low-drift, robust, and fast. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2174–2181, 2015.

[135] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Auton. Robots*, 41(2):401–416, February 2017.

[136] Lintong Zhang, Michael Helmberger, Lanke Frank Tarimo Fu, David Wisth, Marco Camurri, Davide Scaramuzza, and Maurice Fallon. Hilti-oxford dataset: A millimetre-accurate benchmark for simultaneous localization and mapping, 2022.

[137] Ce Zheng, Yecheng Lyu, Ming Li, and Ziming Zhang. Lodonet: A deep neural network with 2d keypoint matching for 3d lidar odometry estimation. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 2391–2399, 2020.

[138] Lipu Zhou, Daniel Koppel, and Michael Kaess. Lidar slam with plane adjustment for indoor environment. *IEEE Robotics and Automation Letters*, 6(4):7073–7080, 2021.

[139] Lipu Zhou, Shengze Wang, and Michael Kaess. π-lsam: Lidar smoothing and mapping with planes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5751–5757. IEEE, 2021.

[140] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1851–1858, 2017.

RÉSUMÉ

Les LiDARS 3D se sont largement démocratisés ces dernières années, poussés notamment par le développement des véhicules autonomes, et la nécessité de redondance et de sécurité. Contrairement aux caméras, les LiDAR 3D fournissent des mesures 3D de l'environnement très précises. Cela a conduit au développement de différents algorithmes de cartographie et de SLAM (Simultaneous Localization and Mapping), utilisant ces nouvelles modalités. Ces algorithmes ont vite dépassé les capacités des systèmes basés sur les caméras. Un élément crucial de ces systèmes est le problème d'odométrie LiDAR, qui désigne le problème d'estimation de trajectoire du capteur, en utilisant uniquement le flux continu de mesures LiDAR. Ce travail se concentre sur ce problème. Plus précisément, dans ce manuscrit nous visons à repousser les performances des odométries LiDAR.

Pour atteindre cet objectif, nous explorons d'abord les méthodes classiques (ou géométriques) d'odométrie LiDAR. Nous proposons notamment deux nouvelles méthodes d'odométrie LiDAR dans le chapitre 3. Nous en montrons les forces et les faiblesses. Pour tâcher de répondre à ces limites, nous regardons de plus près les méthodes d'odométrie utilisant le Deep Learning dans le chapitre 4, en nous concentrant notamment sur les méthodes de type "boîte noires". Finalement, dans le chapitre 5 nous fusionnons les mesures LiDAR et les mesures inertielles pour rechercher encore plus de précision et de robustesse.

MOTS CLÉS

SLAM LiDAR. Odométrie. Localisation. Reconstruction 3D.

ABSTRACT

3D LiDARs have become increasingly popular in the past decade, notably motivated by the safety requirements of autonomous driving requiring new sensor modalities. Contrary to cameras, 3D LiDARs provide direct, and extremely precise 3D measurements of the environment. This has led to the development of many different mapping and Simultaneous Localization And Mapping (SLAM) solutions leveraging this new modality. These algorithms quickly performed much better than their camera-based counterparts, as evidenced by several open-source benchmarks. One critical component of these systems is LiDAR odometry. A LiDAR odometry is an algorithm estimating the trajectory of the sensor, given only the iterative integration of the LiDAR measurements. The focus of this work is on the topic of LiDAR Odometries. More precisely, we aim to push the boundaries of LiDAR odometries, both in terms of precision and performance.

To achieve this, we first explore classical LiDAR odometries in depth, and propose two novel LiDAR odometries, in chapter 3. We show the strength, and limitations of such methods. Then, to address to improve them we first investigate Deep Learning for LiDAR odometries in chapter 4, notably focusing on end-to-end odometries. We show again the limitations of such approaches and finally investigate in chapter 5 fusing inertial and LiDAR measurements.

KEYWORDS

SLAM LiDAR. Odometry. Localization. 3D Reconstruction.