Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

# 3D Point Clouds Reconstruction of Environment Subject to Thin Structures

Matěj Boxan

Supervisor: François Pomerleau, Dr., Pr., P.Eng., SMIEEE
August 2022

## Acknowledgements

I would like to sincerely thank François, who agreed to supervise me on the remarkable journey of writing this thesis. His guidance and all the time he has provided throughout the past six months have been a gift I will always deeply appreciate. I have been extremely lucky to receive such a kind welcome at the Northern robotics laboratory, where I could draw knowledge and passion for robotics from the most inspiring people. I would also like to thank Johann for his helpful insights and interesting ideas throughout this work. My great thanks belong to the whole Norlab team, among which I have not only found great colleagues but, more importantly, life-long friends. Special thanks go to those who helped me with the tunnel experiments, despite taking place on Sunday night. This work would not have even started without Mr. Vladimír Smutný, whose support allowed me to spend the marvellous half a year abroad. Finally, I thank my family and girlfriend for their neverending support.

## Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, on 15 August 2022

Matěj Boxan .................................

## Abstract

Featureless environments represent a major challenge for deploying LiDAR-based Simultaneous localisation and mapping (SLAM) systems. These systems, often relying on the Iterative closest point (ICP) algorithm, are vulnerable to failures caused by the lack of constraints in 3D point clouds. Furthermore, the increasing number of points produced by today's LiDARs is no longer suitable for practical applications and must often be reduced. However, the point cloud sampling operation inevitably erases geometric relations that might otherwise be critical for a correct convergence of the ICP algorithm. We argue that many unconstrained environments are often not wholly featureless, containing cables, lights and other equipment. The effect of these small objects on localisation quality is reduced since, after sampling, they become unrecognisable. In this work, we evaluate sampling methods in the context of LiDAR-based SLAM in an underconstrained environment. Several open-source sampling filters are first analysed and the filters are classified with a proposed taxonomy. Then, we investigate the representation of thin structures in point clouds recorded with four LiDAR sensors. We evaluate the sampling methods employing three datasets with varying feature complexity, ranging from an empty to a fully constrained tunnel. The methods are evaluated on diverse point cloud compression rates and with a precise total station ground truth trajectory. We show that octree-based space subdivision methods are superior to other sampling strategies, but the experiments highlight that no state-of-the-art filter achieves a reliable localisation in an environment constrained only by thin structures.

**Keywords:** Iterative closest point (ICP), point cloud, sampling, 3D mapping, SLAM, LiDAR, unconstrained environment, mobile robotics

**Supervisor:** François Pomerleau, Dr., Pr., P.Eng., SMIEEE
Université Laval, Fac. des sciences et génie, Dépt. d'informatique et de génie logiciel, 1065, Ave de la Médecine, Québec, Québec, Canada, G1V 0A6

## Abstrakt

Prostředí bez výrazných struktur představuje zásadní výzvu pro systémy Simultánní lokalizace a mapování (SLAM) využívající LiDARy. Tyto systémy, často spoléhající na Iterative closest point (ICP) algoritmus, jsou náchylné k selháním způsobeném nedostatečnou komplexitou prostředí. Neustále se zvyšující množství bodů generovaných současnými LiDARy také není vhodné pro nasazení v reálných aplikacích. Na mraky bodů proto musí být aplikovány filtry pro redukci počtu bodů. Aplikace těchto filtrů nicméně nevyhnutelně odstraňuje geometrické vztahy mezi body, které by jinak mohly mít kritický efekt pro správnou konvergenci lokalizačního algoritmu. Mnohá geometricky nepodmíněná prostředí nicméně nejsou prázdná, ale obsahují kabely, světla a další vybavení. Efekt těchto malých objektů na lokalizaci je ale limitovaný, jelikož se po filtraci mohou stát nerozlišitelnými. Tato diplomová práce vyhodnocuje filtry redukující počet bodů v kontextu SLAM s daty z LiDARu v nepodmíněném prostředí. V její první části nejprve analyzuji implementaci několika open-source filtrů a filtry následně roztřídím pomocí navržené taxonomie. Poté zkoumám reprezentaci tenkých objektů v mracích bodů, zaznamenaných čtyřmi LiDARy. Filtry jsou vyhodnoceny na třech datasetech s různou složitostí prostředí, od prázdného po kompletně podmíněný tunel. Metody jsou porovnány na rozličných poměrech komprese a za pomoci přesné referenční trajektorie, získané z totální stanice. Ačkoliv metody založené na rozdělení prostoru pomocí oktálového stromu dosahují lepších výsledků než ostatní metody, experimenty zároveň zvýraznily fakt, že žádná současná metoda nedosahuje spolehlivé lokalizace v prostředí podmíněném pouze tenkými objekty.

**Klíčová slova:** Iterative closest point (ICP), mrak bodů, komprese, 3D mapování, SLAM, LiDAR, geometricky nepodmíněné prostředí, mobilní robotika

**Překlad názvu:** Rekonstrukce trojrozměrných mraků bodů v prostředí s tenkými objekty

# Contents

# Figures

# Tables

## Abbreviations

**CNN** Convolution neural network

**CR** Compression ratio

**DBH** Diameter at breast height

**DNSS** Dual normal space sampling

**DOF** Degree-of-freedom

**FPPS** Feature-preserved point cloud simplification

**FPS** Farthest point sampling

**GNSS** Global Navigation Satellite System

**ICP** Iterative closest point

**IMU** Inertial Measurement Unit

**LiDAR** Light Detection and Ranging

**MEMS** Microelectromechanical systems

**MLE** Maximum likelihood estimation

**MRE** Mean rotation error

**MSE** Mean square error

**NN** Neural network

**NNSearch** Nearest neighbour search

**NSS** Normal space sampling

**OptD** The Optimum Dataset

**RANSAC** Random sampling consensus

**REDE** Relative explored distance error

**RPDE** Relative point distance error

**RTDE** Relative travelled distance error

**SLAM** Simultaneous localisation and mapping

**SpDF** Spectral Decomposition Filter

**SSNormal** Sampling surface normal

**SVD** Singular value decomposition

**TOF** Time of flight

## Synonyms

Point cloud subsampling; Point cloud sampling; Point cloud downsampling; Point cloud simplification.

# Chapter 1
# Introduction

The recent rapid advancement in Light Detection and Ranging (LiDAR) technology found its application in many fields, such as building reconstruction, medical imaging, augmented and virtual reality (AR, VR), object reconstruction, inspection, robotics and others. LiDAR sensors typically produce data in the form of raw point clouds, which are sets of points of x, y, and z coordinates, representing the recorded scene. They may also provide additional values such as the intensity, describing the strength of the reflection of the particular ray from the object's surface. Other attributes can be derived from a point cloud, such as surface normals or observation directions. These can find their use in improved performance of the downstream task. However, unlike the attributes provided directly from the sensor, they come with an additional cost in terms of computation time. Although goals of the downstream applications may differ significantly, from quality control [1] to the creation of 3D reconstructions of entire cities [2], there are similarities that each application needs to address.

Firstly, the recorded point clouds may be subject to noise and outliers [3]. The noise can either be an internal characteristic of the employed sensor or caused by externalities, such as direct sunlight or partial reflection of the laser ray from the edge of an object. Any employed filtering technique should aim at preserving fine details of the point cloud. With the ever-growing complexity of point clouds and more demanding downstream tasks, the problem of raw 3D point cloud filtering remains a challenge.

Secondly, since point clouds are unordered sets of points and therefore lack any topological ordering, it is unclear how to segment, classify, or detect objects easily. The decades of research in image processing techniques cannot be simply transferred to point clouds, which, unlike images, are not row-columns indexed. Attempts to rasterise point clouds into regular grids suffer from low data density. Consequently, the attention has recently shifted towards working directly on unstructured data.

Last but not least, the current generation of LiDAR instruments has seen considerable improvements in all their parameters. Better precision, range, density and higher data rate allow for uses in previously unfeasible domains. Furthermore, some systems are nowadays equipped with multiple LiDAR sensors, observing different directions. The ability to produce millions of points per second causes significant demands on point cloud processing that can get lengthy or even downright impossible. Also, raw point clouds can overload the communication network or quickly occupy a considerable storage space of the computer. The common approach to handle this is to subsample, or simplify, the given point cloud. Naturally, we would like this sampling strategy to preserve important features of the original data, either the shape of objects in the case of 3D reconstruction or minor defects on the examined surface in case of quality control. At the same time, the number of remaining points needs to be taken into account. While too many points cause high computational time, excessive sampling may lead to inaccurate results. Even though the goal is the same for most domains, the actual implementation may strongly depend on the specific downstream task we have in mind when constructing the pipeline.

One such demanding task is the localisation with LiDAR sensors and point clouds. Although digital cameras are getting ground in robot position estimation, the recent Darpa Subterranean

Challenge[1] and Hilti Slam Challenge[2] competitions show that a LiDAR-based solutions still have a firm ground in robot localisation in the Global Navigation Satellite System (GNSS) denied environment. Indeed, at Hilti Slam Challenge, 25 first teams all used a combination of a LiDAR and an Inertial Measurement Unit (IMU), with some also embracing cameras. The best four teams were then relying exclusively on LiDAR and IMU sensors.

The LiDAR-based localisation can be formulated as the problem of aligning two point clouds. The first point cloud, often called sensor reading (or simply reading), is the point cloud being aligned to the second point cloud, the reference. The reference can be, for example, a reading point cloud recorded earlier or a static map point cloud generated offline. Common solutions, such as the Iterative closest point (ICP) algorithm, align the reading to the reference using either raw points or derived features. One situation when point cloud-based localisation techniques fail is in a degenerate environment [4]. An easy example of such an environment in two dimensions is a robot situated in the middle of a circle. Any rotation around the robot's z-axis leads to the same reading, with differences coming only from the sensor's noise. Therefore, the localisation algorithm will report any rotation as an admissible solution. A human-made, real environments carrying similar properties are tunnels or long corridors, which typically lack enough features to constrain the algorithm in one translational Degree-of-freedom (DOF). However, it rarely happens that tunnel's walls are entirely smooth and plain. Indeed, the facades are often equipped with lights, pipes, electricity cables and other thin and small objects. Their influence on the localisation results is nevertheless limited. They may be filtered out as outliers, are too sparse to overweight noise present in the point cloud, or the sampling algorithm does not emphasise their possible contribution to the localisation.

In this work, we will focus on the influence of small structures on the performance of the ICP algorithm. After a brief introduction of the LiDAR sensor modelling and the ICP, we will present the state-of-the-art methods for point cloud sampling. We will also mention the existing methods emphasising thin objects in point clouds. Then, we will describe two point cloud processing libraries and selected subsampling techniques in more detail. Additionally, we will classify these techniques with a taxonomy based on several derived mathematical properties. We will investigate the representation of thin structures in point clouds recorded with four LiDAR sensors. Finally, we will present a detailed comparison of the performance of six sampling strategies with a localisation task in an underconstrained tunnel environment.

[1]`https://www.darpa.mil/program/darpa-subterranean-challenge`
[2]`https://www.hilti-challenge.com/index.html`

# Chapter 2
# State of the art

In its first part, this chapter provides a background on the LiDAR technology, introducing different types, working principles and important parameters of these active sensors. Then, the problem of aligning two point clouds is formulated with a focus on a popular algorithm for point cloud registration, the ICP. Next, we present the state-of-the-art techniques for point cloud sampling. After highlighting the similarities between mesh and point cloud simplification, we further concentrate on spatial-based and local methods of the latter. Additionally, we briefly mention an emerging class of methods employing Neural networks (NNs). In the last section of this chapter, we name several works that emphasise thin objects in 3D point clouds.

## 2.1 LiDAR technology

LiDARs are optical remote sensors that measure a distance to an object or possibly other properties of the object's surface. As an active sensor, a LiDAR measures the distance by emitting a beam of modulated laser light and detecting the light reflected from the object's surface. Assuming an invariable environment and a static speed of light, the distance to the object can be calculated in two ways, as depicted in Figure 2.1. The first option is to consider the LiDAR as a Time of flight (TOF) sensor [5]. For this type of sensors, the distance is computed as the half travelled distance $d = \frac{t \cdot c}{2}$, where $t$ is the time between emission and detection and $c$ is the speed of light in the environment. The second option is to determine the distance from the phase shift between the transmitted and reflected signals [6]. For this to work, the emitted signal is amplitude modulated. The advantage of the TOF scanners is the longer range, which makes them beneficial in terrestrial applications. On the other hand, phase shift sensors have a better range accuracy and higher measurement speeds [7].

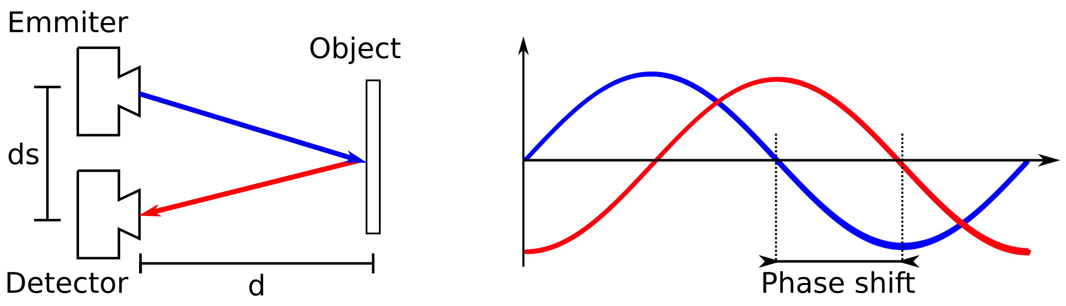


**Figure 2.1:** Two types of LiDAR sensors. Time of flight (left) and the Phase-shift (right).

By directing the laser beam either by rotating the whole device or by adding a rotating mirror, we get a 2D LiDAR. The output of such a device is no longer a single number, i.e., the distance, but an array of recordings situated on a plane. After conversion from polar to Cartesian coordinates, we get a point cloud. By employing multiple emitters, often called channels, each aiming at a different vertical angle, we get multiple lines around the sensor. A conversion from spherical to Cartesian coordinates gives us a 3D point cloud. Modern 3D LiDARs usually have the number of channels in multiples of 2, e.g., 16, 32, 64 or even 128 channels. Together with the ability to identify multiple returns per pulse, the final representation of the environment is even denser. Microelectromechanical systems (MEMS) LiDARs are another category gaining ground in 3D scanning. Unlike their motorised counterparts, MEMS LiDARs steer the laser beam without moving any optical components.

We will now describe several essential parameters of a 3D LiDAR sensor. Lidar beams do not move as straight lines through the surroundings, but are modelled as cones [8]. Beam divergence is the opening angle of the beam cone. The value is usually expressed in millidegrees or milliradians. Beam divergence impacts scanning of large distances because the total amount of pulse energy is constant, and the signal-to-noise ratio is therefore lower. For example, the Velodyne VLP-16 LiDAR used in the experimental section of this work has a horizontal divergence value of 3.0 mrad [9]. This divergence value creates, over the distance of 100 m, a circle with approximately 0.3 m in diameter. Accuracy and precision are standard observation errors described in a sensor's datasheet. The first describes how close the measurements are to the actual value. The second describes the distribution of measurements after repeated experiments. For LiDARs, we distinguish two resolution types. Range resolution describes the smallest distance between two objects to differentiate them in a scan. Angular resolution is the smallest angle difference between two objects to differentiate them in a scan. The scan rate describes how many points the sensor produces per unit of time. The current generation of 128-channel LiDARs generates millions of points per second.

In the context of Simultaneous localisation and mapping (SLAM), there are two more important characteristics of data recorded by a LiDAR sensor. Firstly, the platform the LiDAR is equipped to may be subject to a movement. Even though indoor mobile robots usually move at relatively low speeds of at most units of meters per second, the movements still play a role for sensors with low scan rates. To compensate for the scan distortion, Deschênes *et al.* [10] proposed a deskewing preprocessing step which is applied to the reading, exploiting the high data rate of IMUs. Secondly, the incidence angle of a LiDAR beam has an effect on the reported distance of the laser beam reflection. A high incidence angle creates a bias of up to 20 cm, bending the map to the site where the wall is closer to the sensor, as shown by Laconte *et al.* [11]. After the introduction of the LiDAR sensors, their modelling and characteristics, we will further focus on one specific application for the data they provide, the ICP algorithm.

## 2.2 Iterative closest point algorithm

Having two point clouds, a reference ${}^{\mathcal{B}}\mathcal{Q}$ in the reference frame $\mathcal{B}$ and a reading ${}^{\mathcal{A}}\mathcal{P}$ in the reading frame $\mathcal{A}$, the registration task is introduced as the problem of finding the transformation ${}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T}$, which best aligns the reading to the reference [12]. Finding the best transformation $\mathbf{T}^*$ is formalised as the optimisation problem

$$ {}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T}^* = \arg\min_{\mathbf{T}}(\mathsf{error}(\mathbf{T}(\mathcal{P}), \mathcal{Q})) \quad . \tag{2.1}$$

The error function is calculated on tuples of reading points and their $k$ closest neighbours in the reference. The match function

$$\mathcal{M} = \mathsf{match}(\mathcal{P}, \mathcal{Q}) = \{(\mathbf{p}, \mathbf{q}_i); \mathbf{p} \in \mathcal{P}, \mathbf{q}_i \in \mathcal{Q}, i \in [1 \dots k]\} \tag{2.2}$$

returns a set $\mathcal{M}$, which consists of a point $\mathbf{p} \in \mathcal{P}$ and its $k$ nearest matches from $\mathcal{Q}$, indexed with the index $i$. Outliers may be removed from $\mathcal{M}$ in order to improve performance. This can be done through weights

$$\mathcal{W} = \mathsf{outlier}(\mathcal{M}) = \{w(\mathbf{p}, \mathbf{q}_i); \forall (\mathbf{p}, \mathbf{q}_i) \in \mathcal{M}\} \quad . \tag{2.3}$$

Knowing the associated points, the error function $\mathsf{error}(\cdot)$ can be further expressed as the distance $d(\cdot)$

$$\mathsf{error}(\mathcal{P},\mathcal{Q}) = \sum_{(\mathbf{p},\mathbf{q}_i)\in\mathcal{M}} d(\mathbf{p},\mathbf{q}_i) \quad , \tag{2.4}$$

and after adding the weights $w(\mathbf{p},\mathbf{q}_i)$

$$\mathsf{error}(\mathcal{P},\mathcal{Q}) = \sum_{(\mathbf{p},\mathbf{q}_i)\in\mathcal{M}} w(\mathbf{p},\mathbf{q}_i) d(\mathbf{p},\mathbf{q}_i) \quad . \tag{2.5}$$

From now on, we will assume $k = 1$ for brevity.

As the name suggests, the main idea of the ICP algorithm is that it performs the optimisation step in Equation 2.1 iteratively [13], [14]. The initial misalignment of point clouds $\mathcal{Q}$ and $\mathcal{P}$ may not lead to a perfect association returned from the match function, which in turn may not lead to the best possible alignment of the point clouds. However, even an imperfect association in the iteration $t$ gives us a better estimate for the next iteration $t+1$. By building a sequence of transformations ${}^{t}_{t-1}\mathbf{T}$ and applying it to the reading $\mathcal{P}$, a new set of matches $\mathcal{M}_t$ is obtained. These matches are then used in the successive iteration to get ${}^{t+1}_{t}\mathbf{T}$:

$${}^{t+1}_{t}\mathbf{T} \leftarrow \arg\min_{\mathbf{T}} \left(\mathsf{error}\left(\mathbf{T}\left({}^{i}\mathcal{P}'\right), \mathcal{Q}'\right)\right) \quad . \tag{2.6}$$

Finally, we obtain the final transformation as

$${}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T}^{*} = {}^{\mathcal{B}}_{t}\mathbf{T} \odot \cdots \odot {}^{2}_{1}\mathbf{T} \odot {}^{1}_{0}\mathbf{T} \odot {}^{0}_{\mathcal{A}}\mathbf{T}_{\text{init}} \quad , \tag{2.7}$$

where $\odot$ is the symbol of the transformation composition, $t$ is the number of iteration and ${}^{0}_{\mathcal{A}}\mathbf{T}_{\text{init}}$ is the initial transformation. This iteratively improving process can be eventually stopped with a convergence criterion, e.g., when the maximal number of iterations or the distance between two successive iterations is reached. The procedure is summarised in Algorithm 1.

**Algorithmus 1** Iterative closest point algorithm

```
Require: $^{\mathcal{A}}\mathcal{P}$                                    ▷ Reading point cloud
Require: $^{\mathcal{B}}\mathcal{Q}$                                    ▷ Reference point cloud
Require: ${}^{0}_{\mathcal{A}}\mathbf{T}_{\text{init}}$                 ▷ Initial transformation
 1: $^{t-1}\mathcal{P}' \leftarrow {}^{\mathcal{A}}\mathcal{P}$
 2: $\mathcal{Q}' \leftarrow {}^{\mathcal{B}}\mathcal{Q}$
 3: ${}^{t}_{t-1}\mathbf{T} \leftarrow {}^{0}_{\mathcal{A}}\mathbf{T}_{\text{init}}$
 4: while not converge do
 5:     $^{t}\mathcal{P}' \leftarrow {}^{t}_{t-1}\mathbf{T}(^{t-1}\mathcal{P}')$
 6:     $\mathcal{M}_t \leftarrow \mathsf{match}(^{t}\mathcal{P}', \mathcal{Q}')$
 7:     $\mathcal{W}_t \leftarrow \mathsf{outlier}(\mathcal{M}_i)$
 8:     ${}^{t+1}_{t}\mathbf{T} \leftarrow \arg\min_{\mathbf{T}} (\mathsf{error}(^{t}\mathcal{P}', \mathcal{Q}'))$
 9: end while
10: return ${}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T}^{*} = {}^{\mathcal{B}}_{t}\mathbf{T} \odot \cdots \odot {}^{2}_{1}\mathbf{T} \odot {}^{1}_{0}\mathbf{T} \odot {}^{0}_{\mathcal{A}}\mathbf{T}_{\text{init}}$
```

We will now introduce several popular distance functions $d(\mathbf{p},\mathbf{q}_i)$. We can arrange the output of the matching function $\mathsf{match}(^{\mathcal{A}}\mathcal{P}, {}^{\mathcal{B}}\mathcal{Q})$ as matrices $\mathbf{P} \in \mathbb{R}^{4\times n}$ and $\mathbf{Q} \in \mathbb{R}^{4\times n}$ of points $\mathbf{p}_j \in \mathbb{R}^4$ and $\mathbf{q}_j \in \mathbb{R}^4$ in homogeneous coordinates with $j \in [1 \dots n]$. In the context of robotics, we usually assume ${}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T}$ to be rigid $\mathbf{T}(\mathbf{t},\mathbf{R})$ with translation parameters $\mathbf{t} \in \mathbb{R}^3$ and rotation parameters $\mathbf{R} \in \mathcal{SO}(3)$. Employing transformation matrices and homogeneous coordinates, the transformation composition $\odot$ becomes matrix multiplication, which we can apply on points $\mathbf{p}_j$ and $\mathbf{q}_j$. The point-to-point error function is then defined as the sum of Euclidean distances between point cloud ${}^{\mathcal{B}}\mathcal{Q}$ and ${}^{\mathcal{A}}\mathcal{P}$ [13]. The optimal transformation can be found as

$$ {}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T}^{*} = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{j=1}^{n} ||(\mathbf{q}_j - \mathbf{T}\mathbf{p}_j)||_2 \quad . \tag{2.8} $$

Equation 2.8 has a closed form solution using the centroids of the point clouds and the Singular value decomposition (SVD) of their covariance matrix [15].

Another commonly used distance function is the point-to-plane error [14]. Here the error represents the distance between a reading point and a reference plane, where the plane is defined by a point in the reference point cloud $\mathcal{Q}$ and a unit normal vector $\mathbf{n}$ associated to it

$$ \mathsf{error}(\mathcal{P}, \mathcal{Q}) = \sum_{j=1}^{n} \|(\mathbf{q}_j - \mathbf{T}\mathbf{p}_j) \cdot \mathbf{n}_j\|_2 \quad . \tag{2.9} $$

The optimal transformation can be found as

$$ {}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T}^{*} = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{j=1}^{n} ||((\mathbf{q}_j - \mathbf{T}\mathbf{p}_j) \cdot \mathbf{n}_j)||_2 \quad , \tag{2.10} $$

relying on the linearization of the rotation matrix $\mathbf{R}$ using small angles [14]. In structured environments, point-to-plane generally outperforms point-to-point [16]. Segal *et al.* [17] proposed a generalised version of the ICP, assuming the existence of sets of point $\hat{\mathcal{P}} = \{\hat{\mathbf{p}}_j\}$ and $\hat{\mathcal{Q}} = \{\hat{\mathbf{q}}_j\}$ that generate $\mathcal{P}$ and $\mathcal{Q}$. Points $\mathbf{p}_j \sim \mathcal{N}(\hat{\mathbf{p}}_j, \mathbf{C}_j^{\mathcal{P}})$ and $\mathbf{q}_j \sim \mathcal{N}(\hat{\mathbf{q}}_j, \mathbf{C}_j^{\mathcal{Q}})$ are draw from multivariate normal distributions with $\mathbf{C}_j^{\mathcal{P}}$ and $\mathbf{C}_j^{\mathcal{Q}}$ being covariance matrices. The optimal transformation can be found using the Maximum likelihood estimation (MLE) as

$$ {}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T}^{*} = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{j=1}^{n} (\mathbf{q}_j - \mathbf{T}\mathbf{p}_j)^T (\mathbf{C}_j^{\mathcal{Q}} + \mathbf{T}\mathbf{C}_j^{\mathcal{P}}\mathbf{T}^T)^{-1} (\mathbf{q}_j - \mathbf{T}\mathbf{p}_j) \quad . \tag{2.11} $$

The point-to-point, point-to-plane and plane-to-plane errors can be seen as special cases of the covariance matrices $\mathbf{C}_j^{\mathcal{P}}$ and $\mathbf{C}_j^{\mathcal{Q}}$.

The domain of the ICP error metric is still a subject of active research. Recently Kubelka *et al.* [18] proposed to reduce the 6DOF to 4DOF by constraining the roll and pitch angles to be aligned with the gravity vector.

The matching function $\mathsf{match}(\mathcal{P}, \mathcal{Q})$ requires an efficient way to look up the closest point of $\mathbf{p}$ in $\mathcal{Q}$. The naive implementation of the Nearest neighbour search (NNSearch) problem has a computational complexity $O(nm)$, with $n = |\mathcal{P}|$ and $m = |\mathcal{Q}|$ being the number of points in the reading and the reference point clouds, respectively. A more efficient data structure to do this look-up is the k-d tree [19]. In 3D, each node that is not a leaf represents a plane dividing the space into two parts. The axis with the highest variance is chosen for the space division. Each node contains the median used for splitting, together with the axis along which it was split. By partitioning the data in space this way, we get a search complexity $O(n\ log(m))$ with a construction phase of $O(m\ log(m))$.

In the context of robotics, we usually assume a LiDAR sensor feeding data to the localisation pipeline at a fixed rate. Ideally, we would like to keep the localisation rate as close as possible to the rate of the LiDAR. This requirement becomes highly important for vehicles travelling at greater speeds or when the vehicle rapidly explores a new environment e.g., a robot going through a door without any prior knowledge of what is behind. The standard option to increase the rate of the ICP is to subsample the reading point cloud. However, since the numbers of points in today's maps have not reached the asymptotic properties of the big $O$ function, limiting the number of points in $\mathcal{Q}$ also plays a role, especially when the ICP performs multiple iterations of point cloud matching and alignment. Furthermore, a carefully chosen subsampling strategy can improve the quality of the results [20]. In the following section, we will analyse the methods to reduce the number of points in a point cloud.

## 2.3 Point cloud sampling

Point cloud sampling has been widely studied in the past decades. Many solutions focus on dense, low-noise data, such as those used for object reconstruction. In these applications, the goal is, broadly speaking, to reconstruct the object's surface as close to the original model as possible. They are generally evaluated on well-described 3D model classification benchmarks, such as ModelNet40 [21]. The experiments are usually not evaluated in terms of speed, as real-time solutions are not required, and other tasks related to object reconstruction may be more time-demanding, e.g., surface reconstruction.

Three-dimensional point clouds, used for retrospective processing of large urban environments, can contain billions of points [2]. Subsequent 3D models could enable better city planning, public facility inspection, virtual tourism, and more. Localisation of objects of interest, segmentation, and following classification can take tens of hours of computations. Even though there are heavy requirements for storage space, the system obviously does not require any real-time capabilities.

These capabilities are nevertheless crucial for navigation and mapping of unknown environments, search and rescue missions and mobile robot localisation. Another example from mobile robotics is collision avoidance, where we must recognise and avoid close and dangerous objects in the robot's surroundings. Clearly, no other than real-time solutions are admissible. A loss of an important yet small part of the original point cloud during sampling could lead to severe consequences for both the robot and the environment it is moving in.

In this section, we first define the problem of point cloud subsampling. Then, we present an overview of different sampling strategies.

### 2.3.1 Problem definition

Given an unordered point cloud $\mathcal{P}$ with $|\mathcal{P}| = n$ points, each point is represented as a vector $\mathbf{x} \in \mathbb{R}^d$, with $d$ being the number of dimensions. In our case, $d = 3$ and $\mathbf{x} = [x,\ y,\ z]^T$ describes the coordinates in the Euclidean space. The goal of sampling is to generate a set $\mathcal{P}'$ of $n' < n$ points that preserves geometrical and other features of $\mathcal{P}$. The reduction ratio is then given as $\alpha = 100\frac{n'}{n}$ and the Compression ratio (CR) is defined as $\mathrm{CR} = 100 - \alpha$. Note that in some cases, the resulting point cloud $\mathcal{P}'$ is not a subset of $\mathcal{P}$. Depending on the application, a post-processing neighbour search in the original cloud may be then performed [22]. Naturally, down-sampling preserves data structure and thus can be incorporated as a preprocessing step into any point cloud pipeline. After sampling, part of the original point cloud information is lost irretrievably.

### 2.3.2 Mesh-based methods

Point cloud simplification is closely related to the mesh simplification problem, which is usually discussed in the context of computer graphics, solid modelling, reverse engineering, video gaming, 3D animation and others. The motivations to simplify large, complex meshes are based not only on the need to simplify rendering on less-powerful devices but also to allow fast transmission rates and efficiently store large numbers of 3D models of, for example, scans of cultural heritage.

A review and comparison of five mesh simplification algorithms are presented in [23]. The employed taxonomy distinguishes seven different simplification approaches. The first approach is the coplanar facets merging, where coplanar areas of the mesh are merged into larger polygons. In controlled voxel, edge or face decimation, the geometric shapes are removed iteratively. The next approach is re-tiling, where new vertices are inserted in specific locations and the original vertices are removed. Energy function optimisation uses a function describing the mesh quality. The mesh simplification is formalised as an optimisation problem and the function value is maximised. In vertex clustering, several vertices are grouped and then replaced by a single vertex. Wavelet-based simplification approximates the mesh with a multi-resolution version of the surface. And finally, simplification via intermediate hierarchical representation uses an octree as an intermediate step.

Mesh-based methods are implemented in both open-source and commercial 3D tools. Uccheddu *et al.* [24] compared several such tools in terms of noise robustness and stability, motivated by 3D Watermarking. The mesh simplification algorithm provides a coarse mesh, which is then used to hide some secret information describing copyright protection. Even today, mesh simplification is a subject of active research. Asgharian *et al.* [25] presented a novel approach based on the Nyquist theorem and adaptive sampling strategy. Li *et al.* [26] employed mesh filtering to generate a compact version of building models.

A possible solution to the point cloud simplification problem could thus be to generate a 3D mesh from a point cloud, process it and turn the resulting vertices back into a point cloud. However, the construction of polygonal mesh is computationally expensive since it requires a large amount of geometric information about the points relations. Also, some methods do not preserve the original mesh topology, such as mesh decimation. Mesh simplification methods are mostly tested on uniform, low noise, and small individual objects with no background and hence are not suitable for our task.

### 2.3.3 Mesh-free methods

Already in 2001, Pauly *et al.* [27] noticed the shift from mesh-based to point-based methods. Since then, the algorithms working directly on point clouds have seen rapid development. The most widely used taxonomy distinguishes two classes, global and local methods. On the one hand, global methods work on the whole point cloud and do not need any information about a particular point's neighbourhood. Uniform random sampling may be the simplest example, where a fixed number of points is selected randomly from the input point cloud. An alternative is to subdivide space into a  grid and choose the most representative sample from each cell. Other example is the work of Błaszczak-Bąk *et al.* [28]. The Optimum Dataset (OptD) [29] method was used to extract off-road objects, classified into four categories: traffic signs, light poles, trees and power lines. OptD divides the point cloud area into strips, which are processed separately. On the other hand, local methods require the computation of geometric features of individual points, leading to the need for a nearest neighbours search. As discussed earlier, the search has a complexity $O(m \log(m))$, with $m$ being the number of points in the point cloud. The adjoining points can enhance the point cloud with descriptors, such as normals, which can then be used for more informed sampling.

We will also briefly mention two more classes: Iteration-based and Formulation-based methods. As an example of the former, we can use a criterion to find the least important point, remove it and simplify the point cloud iteratively. Farthest point sampling (FPS) works similarly. As a greedy algorithm, it iteratively selects a subset of points that are the farthest apart from each other, starting from a random sample [30], [31]. A batch version of FPS allows for a parallel implementation[1] which makes it a popular choice for use with methods employing NNs. The latter methods opt for a more mathematically rigorous approach and desire point cloud sampling with a clear proof of optimality [32].

Although most NNs need either a nearest neighbour computation to process geometric information or a preliminary rasterization step to process the point cloud, Nezhadarya *et al.* [22] presented a method for deterministic sub-sampling of unordered point clouds using the proposed Critical Points Layer. Even though they argue that the motivation for sub-sampling arises from a large number of points generated from nowadays LiDAR sensors, in the experiment section, they start with only 1024 points sampled from each 3D model of the used dataset. Another option for NN deployment on point clouds is to project

[1] `https://github.com/open-mmlab/OpenPCDet`

the 3D point cloud onto a 2D image and process it using a standard 2D Convolution neural network (CNN), as in the work of Nubert *et al.* [33]. The employed NN computes the transformation **T** directly, without the need for a separate registration calculation. It works on raw point clouds, thus entirely omitting the sub-sampling step. The network is based on unsupervised learning and requires nearest neighbour search only during an offline training phase. The results reported on three datasets captured by three different sensors on three distinct platforms show state-of-the-art results and a real-time processing time.

We will further study three categories - Clustering-based, Geometric and NN-based methods.

### 2.3.4 Spatial-based methods

Methods based on the subdivision of space are a subset of global methods. The idea of these methods, sometimes also called volumetric-based or clustering-based, is to portion the points into cells aligned with the xyz-basis of a defined coordinate frame. One or several points from each cell is then selected as a representative sample. This saves the need for nearest neighbour computation but at the same time creates quantization artefacts and ignores inter-point features. The number of output points and level of detail can be controlled only indirectly through the grid resolution. Nevertheless, the strictly structured output of clustering-based methods was widely used as a pre-step for applying CNN on point clouds [34].

In Uniform grid sampling, points are projected onto a uniform grid plane, and one point is selected based on a median-filtering rule [35]. Lee *et al.* [36] extended the idea to Non-uniform grid sampling, where the grid size is not constant and may vary based on the object shape.

Another option is to divide the environment into a grid of cubes of equal size, called voxels. In the case of sparse data from laser scanners, grids are memory inefficient since most voxels are empty. Furthermore, the bounding box of the environment needs to be known in advance. Otherwise a costly expansion operation needs to be computed [37]. From each voxel, a representative point can be selected as the sample [38]. Common choices include the voxel centre, the closest point to the cell centre, the centroid or medoid of all points falling into the voxel, a random or the first point from all points falling into the voxel or a simple binary value to model a Boolean property, usually the occupancy of the given voxel.

An alternative, more memory and computation-efficient approach in 2D is the quadtree, or its 3D variant, the octree [39]. Figure 2.2 shows an example of the octree, a tree structure where the first node is assumed non-empty and subdivided into eight octans. An octan is then recursively subdivided until the desired depth is reached, or based on the number of points in the octan. The representative point can be chosen similarly to the voxel grid. The apparent advantage of using octrees is the memory efficiency. Efficient octree implementations for point cloud compression and processing were presented in [40], [41]. Probabilistic occupancy octrees were used in the 3D mapping framework of [37].

### 2.3.5 Local methods

Methods based on geometric features profit from the geometric information encoded in individual points' neighbourhoods. The analysis of distinct geometric features decides which points are worth preserving. At the same time, the nearest neighbour search is computationally expensive. Therefore, some sources include this computation in preprocessing steps. This section will present several works related to local or geometry-based point sampling.

Pauly *et al.* [27] presented a framework to process point clouds with normals through spectral methods by extending windowed Fourier transforms to geometry. Experiments on 3D models were presented together with computation time for different pipeline stages. Rusinkiewicz *et al.* [42] also used normals. Their Normal space sampling (NSS) deals with translation uncertainty in registration. The method chooses points uniformly to their normal orientation. The work of Gelfand *et al.* [4] extended NSS and presents an algorithm capable of handling both translation and rotation uncertainty. They describe a greedy algorithm that selects

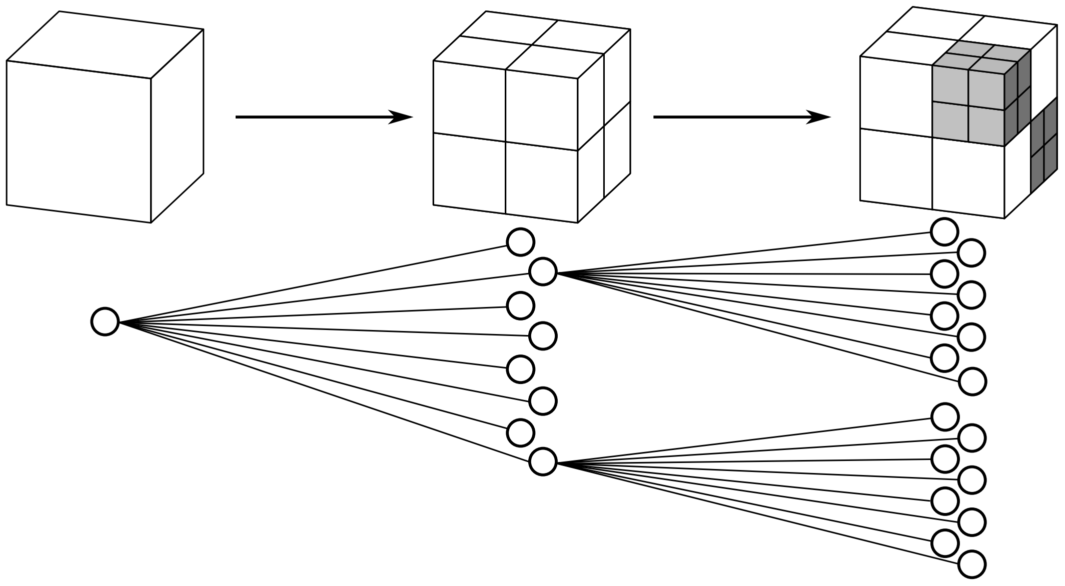


**Figure 2.2:** The concept of an octree. Top: The space is recursively divided into octans. Bottom: The graph of the corresponding octree.

points for ICP registration to limit uncertainty in the pose. The unstable transformations are avoided by evaluating the covariance matrix $\mathbf{C}$, which indicates the presence of sliding, e.g., the situation when the transform $\mathcal{T}$ is not constrained in one direction. The evaluation is conducted on two types of synthetic and real-world data against uniform and normal space samplings. Kwok *et al.* [43] further developed the idea of Covariance sampling. They argue that to constrain the rotation, it is necessary to have a correct estimate of its centre. Therefore, they propose a two-phase algorithm that addresses this nonlinear optimisation problem. Kwok [44] extended the idea of NSS to Dual normal space sampling (DNSS). He redefines the normal space as not only the space of translational normals $\mathbf{n}$, but also rotational normals, defined as the cross product $\mathbf{p} \times \mathbf{n}$, where $\mathbf{p}$ is a sampling point. The DNSS has the same complexity as the NSS, but unlike the NSS, it can achieve a convergence even from an orthogonal initial position.

Golovinskiy *et al.* [2] examined a pipeline for object detection in urban environments, composed of four steps: localisation, segmentation, characterisation and clustering. To subsample the point cloud, they first filter out points estimated to be on the ground through the iterative plane fitting. After removing isolated points, they filter out the remaining large connected components assumed to be part of buildings. The method is evaluated on a large urban dataset containing approximately 100 million points.

Weber *et al.* [45] showed a new Gauss map-based method for sharp feature detection in point clouds. In subsequent work, Weber *et al.* [46] presented an overview of methods for feature detection in point clouds but do not compare the presented Gauss map-based method to others. The method looks for k nearest neighbours of each point $p$, creates a set $T$ of triangles between $p$ and two of its neighbours, computes the triangle's normal and projects it onto the Gaussian sphere. Since different geometric features lead to different Gauss map patterns, the method can detect lines and corners.

Qi *et al.* [32] argued that sampling is a trade-off between uniformity and sharp feature preservation. The proposed graph-based sampling solution is visually evaluated with a registration downstream task. Their method aims to enhance contours but also to maintain uniform density, based on a user-defined parameter. The task is formulated as graph signal processing, where the assumed undirected graph $\mathcal{G}$ is a K-NN graph with an adjacency matrix $W$. The defined objective function combines loss in feature with loss in density uniformity and can be, after relaxation, solved with existing optimisation algorithms. The whole point cloud is divided into cubes to accelerate the computation, and each cube is processed separately.

Experiments compare the proposed method to three others on four 3D models. However, no time results are provided. Similarly, Chen *et al.* [47] also interpreted raw point clouds as graphs (Figure 2.3), which they construct similarly. They subsample the original point cloud randomly but using a non-uniform distribution. This optimal resampling strategy is introduced individually for three filter types. An all-pass filter assumes uniform importance of all points in the point cloud. A high-pass filter extracts contours and a low-pass filter removes noise. Experimental validation is given for the proposed filters. The high-pass filter is evaluated on a downstream registration task but is compared only to uniform sampling, and no time results are presented. Wu *et al.* [48] extended this graph-based approach with a strategy to handle point clouds with more than 100 million points, inspired by the *divide and conquer* philosophy. The point cloud is decomposed into smaller subsets based on natural neighbours and processed separately. As part of the evaluation, they test their approach on one scene from an urban dataset [49].

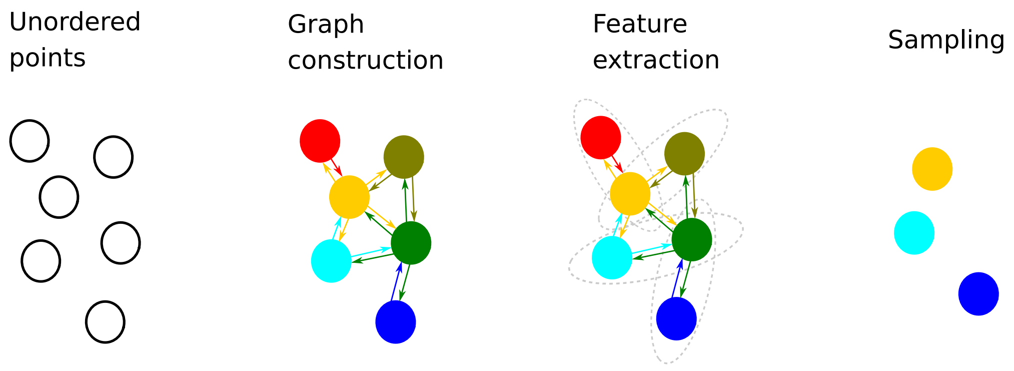


**Figure 2.3:** Graph-based methods exploit the decades of research on graph theory. The graph is first constructed with nodes representing 3D points. Features are extracted from the edges and neighbour nodes. Nodes are then discarded or preserved based on these features.

Zhang *et al.* [50] presented Feature-preserved point cloud simplification (FPPS). Natural quadric surface is used as a simplification model of the geometric in each key point's neighbourhood. The key points are found using three entropies: scale-keeping, profile-keeping and curve-keeping simplification entropies. The results are evaluated on 3D models, compared to Uniform, Grid and Curvature based samplings and shown together with the running time and memory usage statistics.

Labussière *et al.* [20] presented a sampling algorithm created in the context of ICP and robotics applications called Spectral Decomposition Filter (SpDF). The tensor voting framework [51] was used to identify basic geometric features and their saliencies. The method then iteratively subsamples each geometric feature until the density is lower than the desired one. The method, which is compared to seven others, has an error within the LiDAR sensor's range accuracy for 97 % compression ratio. For a compression ratio of 95 %, the calculation takes about 500 ms.

In the work of Gong *et al.* [52], each point's local geometry is represented as a Delaunay neighbourhood. The extracted feature points are joined with the rest of the point cloud, uniformly sampled. The method is evaluated on four 3D models against Grid and Curvature based simplification and an algorithm based on conformal geometric algebra [53].

## 2.3.6 Methods employing Neural networks

The first NN working directly on unstructured point clouds, called PointNet, was presented in [54]. The inherent point cloud's lack of structure was solved using a max-pooling layer. According to the presented experiments, the network is robust to noise and outliers. It was capable of processing up to 1 million points per second with a linear space and time complexity in the number of points in the input point cloud. The network was designed to segment and classify 3D objects, and the set of optimisation functions it learns leads to the selection of the most critical points for these two tasks. These points, called critical points, correspond to the skeleton of the object's shape. A follow-up improvement of PointNet, PointNet++, was presented in [55]. Unlike the original network, the improvement can learn local features by applying PointNet recursively on nested parts of the original point cloud. The point cloud is first sampled using FPS. A grouping area created from neighbours of the samples is encoded into feature vectors by a PointNet layer. They also propose a method to deal with non-uniform sampling density, so the proposed network is more robust to local density variations. The network is evaluated on four 2D and 3D object datasets based on classification and segmentation tasks. Similarly, Shen *et al.* [56] introduced changes to PointNet for efficient representation of local structures, called KCNet. However, they argue that PointNet++ leads to a complicated architecture and instead proposes treating a point's surrounding, which they call a kernel, as the source of local geometric information. The kernel correlation is let to adjust freely through backward propagation. The learning process finds the best point encodings for the most valuable local geometric structures. KCNet was evaluated on both 2D shape and 3D model datasets.

He *et al.* [57] presented the combination of PointNet++ and ICP as a new registration method. The idea is to let PointNet++ find multiple feature identifiers and feed the points corresponding to those multidimensional features into ICP instead of the whole or sampled point cloud. For the experiments, they trained PointNet++ on a large dataset of 3D models, ModelNet40 [21]. The pipeline was evaluated on the registration task using the testing part of ModelNet40, ScanNet [58] and KITTI [59] datasets. The results showed Mean square error (MSE) lower than $10^{-4}$ m for both ModelNet40 and ScanNet. For the KITTI dataset, which contains sparse outdoor data captured on a mobile platform, the MSE is 5.83 cm, and the pipeline performs more than 30 times faster than a stand-alone ICP.

Lang *et al.* [60] proposed omitting the widely popular FPS and instead provided subsamples as an approximation of points in the original point cloud. Performance of the proposed network, SampleNet is presented on the classification, reconstruction and registration of objects in the ModelNet40 dataset [21]. Unlike [57], where the network was trained strictly on point clouds to learn their most important features, the training also includes the registration task. No translation is considered between the two scans, and the performance is evaluated only in terms of Mean rotation error (MRE). SampleNet outperforms ICP with FPS and Random sampling as the sub-sampling methods in this metric.

Recently, He *et al.* [61] stored information about the local density in a point feature and proposed a method for point cloud compression employing an autoencoder. The encoder subsamples the point cloud, learning a density, a local position and an ancestor embedding. These embeddings are meant to store the geometry and density of the removed points. Finally, the encoder compresses the per-point features and quantises the point cloud. An adaptive module efficiently decoding the compressed point cloud is also proposed. More detailed surveys on deep learning on 3D point clouds can be found in the reviews of Bello *et al.* [62] and Guo *et al.* [63].

The operation of point cloud sampling has an impact on every imaginable downstream task. From localisation to path planning, the tasks would not be possible without a genuine reconstruction of the scanned environment. The reconstruction may be harder for objects whose size is close to the accuracy of the sensor or whose dimensions are minor compared to other present objects such as walls. In the next section, we will examine related works done on the topic of thin structures in 3D point clouds.

## 2.4 Thin structures

The study of thin structures in point clouds primarily focuses on classification, segmentation and object detection for autonomous cars. Since road-based vehicles typically have access to GNSS data and the surrounding environment has enough features, the detection of small structures is mostly focused on safety, obstacle and collision avoidance. In the context of these applications, typical small structures present in the labelled data are pedestrians, cyclists, poles and traffic signs [64]. For example, Ye *et al.* [65] presented a solution to voxel grid rasterisation artefacts in the form of a special detector layer called Hybrid Voxel Feature Extractor. The results are evaluated on the classification task of the KITTI dataset. Alternative use of LiDAR data on roads is for quality inspection and infrastructure inventory. Ma *et al.* [66] presented a review on the detection and extraction of road objects using mobile laser scanning devices. Their possible employments go from detecting cracks and utility holes to quality control of road markings and driving lines. Interestingly, all except one of the reviewed surface extraction methods simplified the acquired point cloud with a voxel grid, with the remaining one employing a Random sampling consensus (RANSAC) algorithm for plane fitting.

One more type of thin object often occurring in point clouds captured by road vehicles are trees. Trees are a common sight along many roads and can significantly affect road safety. Zhong *et al.* [67] presented a system for street tree inventory from point cloud data. Tree trunk and foliage are extracted to estimate its height, Diameter at breast height (DBH) or crown width. The minimal tree trunk diameter in the showed results was 0.121 m. Tremblay *et al.* [68] also describe tree trunk diameter estimation, but in the context of forestry. Even though trees with DBH of less than 10 cm were segmented and measured, most of them were not included in the final evaluation. The reason for this is the low signal-to-noise ratio [69]. A typical forest is not an environment lacking features. However, monitoring young trees and avoiding branch-like obstacles requires a correct representation of the environment. We see that although the objective of the above methods is to recognise and label small objects correctly, the goal is typically not to improve localisation performance.

Another category of literature discussing small objects is the modelling-based approach. The critical property of a LiDAR beam concerning thin structures is its sparseness. While interpolating missing points to achieve a more dense resolution can be feasible on smooth surfaces, it does not help detect object boundaries. Long *et al.* [70] proposed a model for LiDAR beam divergence and a calibration method for divergence angle estimation, together with an explication of the effects on dilation and erosion. Dilation and erosion are the consequences of width measurements computed as the distance between the two farthest points on a row. Dilation, caused by the finite beam width, is caused by the beam centre located outside of the object but still producing a return. Erosion, on the other hand, is related to azimuth sampling. The beam centres are, in this case, inside the true object's boundaries. For experiments, they used cylinders with a width of 5.08, 7.62 and 10.16 cm and street lamps, together with a moving platform.

# Chapter 3

# Analysis of filtering algorithms for 3D localisation and mapping

After the broad review, we will introduce two open-source libraries employed in this work. After that, we will formulate several mathematical properties of a subsampling filtering strategy. Finally, we will use these properties to present a classification of subsampling strategies available in the `libpointmatcher`[1] library.

## 3.1 Libpointmatcher

The `libpointmatcher` is an open-source library for point cloud matching [16]. Written in C++11 and configurable at runtime using YAML text files, every part of the ICP chain is fast and parameterised. `libpointmatcher` thus allows for an easy comparison of different strategies of each part of the ICP pipeline. Different modules, or processing blocks, are available for each part of the chain. Figure 3.1 shows the processing blocks of the `libpointmatcher` library. The chain takes two point clouds on the input. Data filters are applied to the input

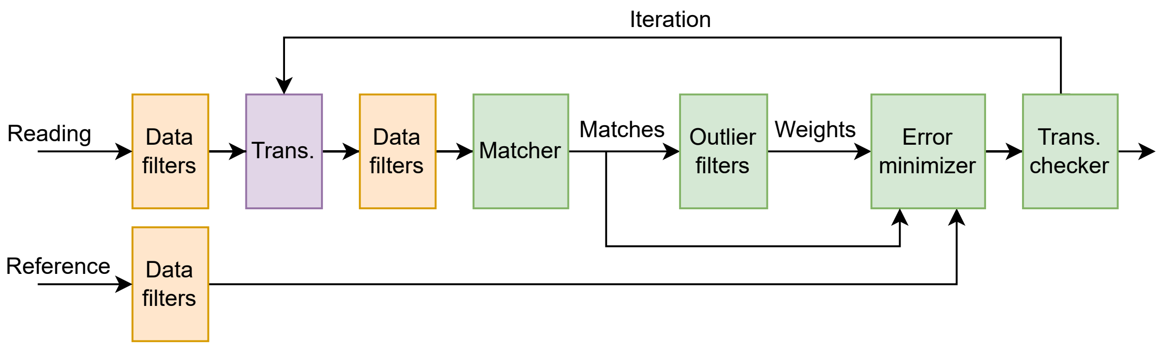


**Figure 3.1:** Processing chain of the `libpointmatcher` library. Data filters, depicted in orange apply sampling and descriptor augmenting filters to the point clouds. Trans. stands for transformation, with the purple block transforming the reading point cloud. Processing blocks marked in green are part of the ICP algorithm, defined in Algorithm 1.

point clouds before starting the ICP loop. There are two types of these filters: Downsampling and Descriptor augmenting. The effect of the first is that they reduce the number of points in the point cloud. However, some of them also add descriptors, such as normals, to the remaining points. The goal of the descriptor filters is, on the other hand, to augment the data with additional information. The data filters are applied directly to the point cloud. Some assume that the sensor capturing the point cloud is located in the origin of the point cloud coordinate frame. The fact that the location of the origin of the coordinate frame of the point cloud matters is apparent for uncomplicated filters. For example, the bounding box filter simply removes points from inside or outside of a rectangular region. However, we

[1] `https://github.com/ethz-asl/libpointmatcher`

will later see that the origin's location is important even for more complex down-sampling filters.

Inside the ICP loop, the reading point cloud is first transformed. Then, another set of data filters can be applied to the reading. This is useful, for example, when we want to gradually decrease the number of points in the scan to speed up the computation or to provide the matcher with a different random subset of points in each iteration. Next, a data association step is performed to link the points in the reading point cloud to points in the reference. The association is done with a k-nearest neighbour matcher. Outlier filters add weights or altogether remove links between points in the reading and the reference. The Error minimiser performs the optimisation step in the search for the next transformation. Finally, multiple Transformations checkers can be used to stop the ICP after a condition is met. For example, the condition can be related to the number of iterations of the ICP loop or the matching error.

## 3.2 ICP Mapper

The `norlab_icp_mapper`[2] is a lightweight open-source package wrapping the ICP functionality and combining it with map creation. The ICP, imported from `libpointmatcher`, provides the transformation between the sensor and the map frame while the mapper manages map updates. The mapper does not contain any global optimisation, and without a loop closure, it is, in fact, performing a lidar-odometry task. The difficulty of providing a reliable covariance estimate is the main obstacle to revising any previous pose estimates. A missing covariance, in turn, restricts data fusion with other pose estimation methods, such as the IMU data, in an Extended Kalman or another Bayesian filter [71] [72]. The mapper also provides memory management, offloading parts of the map that are too far from the robot. This offloading is parameterised with the maximal range of the sensor used. However, in our scenario, we will not leave the circle defined by the radius of the sensor's range.

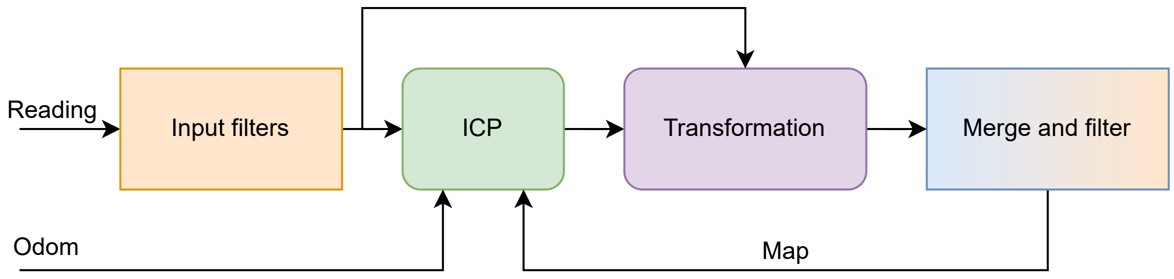


**Figure 3.2:** Processing blocks of the `norlab_icp_mapper`. The reading is first processed in the filtering block, depicted in orange. The odometry prior is used to align the filtered reading to the map in the ICP block. The reading is transformed using the optimal transformation and merged into the map, which is then filtered. The colour of the processing blocks corresponds to the blocks used in Figure 3.1, with the reading input filters being the first reading data filter block.

Algorithm 2 contains a detailed description of a modified `norlab_icp_mapper` while Figure 3.2 depicts a diagram of its processing blocks. We will emphasise the differences from the version available online later. An offline version of the mapper was used. Even though we recognise the importance of real-time solutions for robotics, our goal is to provide a fair comparison of different sampling methods. However, we will discuss the time requirements of the individual sampling methods in the results section of this work.

The algorithm takes a list of reading point clouds and odometry poses as the input. After setting the initial transformation to Identity and applying initial filters to the reading, we

[2]`https://github.com/norlab-ulaval/norlab_icp_mapper`

merge the reading into the initially empty map and apply map filters. Unlike in the standard version of the algorithm, where the merging and map filtering is applied as two separate steps, we emphasise their interconnectedness. The combined mergeAndFilter() function has access to the sum of the number of points of all readings until now. For simplicity, we will assume that it stores this number in its internal memory. In the main loop, we process the remaining readings and poses. The estimated transformation, filtered reading and latest map are used in Algorithm 1 to acquire a corrected transformation. This transformation is used to transform the reading into the map frame. The reading is then merged into the map, which is finally filtered. The merging into the map is conditioned by the updateMap() function. Currently, `norlab_icp_mapper` offers three conditions for a map update: a time difference, a distance difference and an overlap. The time difference takes the minimal number of seconds between two map updates as an argument. The distance difference takes the minimal distance between two map updates as an argument. The overlap condition ensures that the reading is merged into the map only if there is enough overlap between the two. The algorithm returns the final map and the trajectory.

**Algorithmus 2** Norlab ICP offline mapping

```
Require: $\mathcal{P}_{0:t}$                                   ▷ Reading point clouds
Require: $O_{0:t}$                                             ▷ Odometry
 1: $\hat{\mathbf{T}}_0 \leftarrow \mathbf{1}$                  ▷ Initial pose
 2: $\hat{\mathcal{P}}_0 \leftarrow$ inputFilters($\mathcal{P}_0$)      ▷ Filter first reading
 3: $\mathcal{M}_0 \leftarrow$ mergeAndFilter($\emptyset, \hat{\mathcal{P}}_0$)   ▷ Get map from the initial reading
 4: while i ≤ t do
 5:     $\hat{\mathbf{T}}_i \leftarrow O_{i-1}^{-1} O_i \hat{\mathbf{T}}_{i-1}$      ▷ Pose estimate
 6:     $\hat{\mathcal{P}}_i \leftarrow$ inputFilters($\mathcal{P}_i$)      ▷ Filter reading
 7:     $\mathbf{T}_i \leftarrow$ icp($\hat{\mathcal{P}}_i, \mathcal{M}_{i-1}, \hat{\mathbf{T}}_i$)   ▷ Update pose estimate with ICP
 8:     $\mathcal{P}'_i \leftarrow \mathbf{T}_i \hat{\mathcal{P}}_i$      ▷ Transform reading to the map frame
 9:     if updateMap() then
10:         $\mathcal{M}_i \leftarrow$ mergeAndFilter($\mathcal{M}_{i-1}, \mathcal{P}'_i$)   ▷ Merge reading into map and filter map
11:     end if
12: end while
13: return map $\mathbf{M}_t$, trajectory $\mathbf{T}_{1:t}$
```

Unlike the original version, both the inputFilters() and the mergeAndFilter() functions apply the filters in the frame of their input point clouds. The reading is processed in the sensor frame and the map in the map frame whereas in the original version even the map was processed in the frame of the sensor.

## 3.3 Sampling filters

We already explained in Section 2.2 the effects of the number of points in the reference and the reading point clouds on the ICP performance. Furthermore, lidar point clouds are generally sparse and incomplete, thus unsuitable to be used with advanced local filters. The high sparsity level makes it complicated to derive any underlying object properties or to interpret object structure correctly. The low density is sometimes overcome with an up-sampling, an operation inverse to sub-sampling where the number of points in the point cloud increases [73][74]. However, interpolating between beams introduces new noise to the data.

We will, therefore, further focus on the sampling of the reference. In our scenario, the reference is the environment map, which is continuously updated as more readings are merged into the map. Given the two `libpointmatcher` and `norlab_icp_mapper` chains we introduced earlier in this chapter, we have two options. Either we subsample the map in the Data filters block of the `libpointmatcher` pipeline, or we subsample it as part of the mergeAndFilter() operation of the mapper. The diagram of the two variants is present in Figure 3.3.

The clear advantage of the first variant is that the filter has all points from the previous iterations accessible for feature extraction. An ideal filter should select the most information-dense regions, which should, in turn, lead to the best ICP performance. However, for the first variant, the number of points in the map would grow rapidly without bounds. This would not only make the kd-tree construction and the nearest neighbour lookup lengthy, but eventually, we would run out of memory. On the other hand, the second variant filters the map continuously after each reading merge, and the resulting map is used in the next iteration. This, at least for the density and spatial-based methods, effectively limits the maximal number of points in the map.

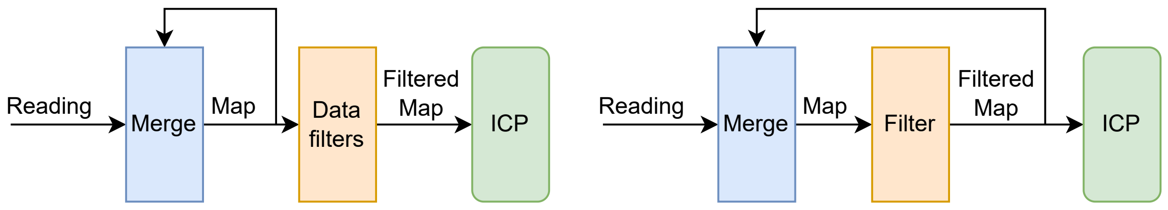


**Figure 3.3:** Map sampling approaches. Left: The map is only filtered before the ICP chain, and the filter has access to points from all preceding readings. Right: The filter is applied continuously after each map update. The blue and orange blocks correspond to the gradient colour margeAndFilter() of Figure 3.2.

### 3.3.1 Theory

From now on, we will use $\mathcal{M}$ to denote the map, a point cloud with $|\mathcal{M}| = m$ points. A filter $\mathbf{F}$ subsamples the map so that $|\mathbf{F}(A, \mathcal{M})| \leq m$, where $A = \{a_1, \ldots a_n\}$ is a set of optional parameters of the filter. $\mathbf{F}(\mathcal{M})$ will be used for a filter with no parameters. In this section, we will describe certain mathematical properties that we will later use in the taxonomy of several popular sampling filters.

In Section 2.3.1, we defined the Compression ratio as the fraction of the number of points on the output and the number of points on the input of the filter: $\mathrm{CR} = 100 \cdot (1 - \frac{m'}{m})$. Although this definition works well for the first variant of map sampling, it does not work for the second. This is because the important property of a continuous sampling is not the instantaneous value of the compression ratio but more the long-term behaviour of ignoring redundant information already present in the map point cloud.

**Definition 3.3.1** Compression ratio. The updated compression ratio is given as

$$\mathrm{CR}_t = 100 \cdot (1 - \frac{|\mathbf{F}(\mathcal{M}_{t-1} \sqcup \mathcal{P}_t)|}{\sum_t |\mathcal{P}_t|}) \quad , \tag{3.1}$$

where $\sqcup$ is the union operator denoting the operation of merging the reading into the map and $t$ is time.

We see that for growing time $t$, the number of points in the map $|\mathcal{M}|$ either converges to a fixed value and CR$\rightarrow$100 % or keeps pace with the incoming points and grows indefinitely. This, of course, would not be a good description value for an experiment of infinite duration or deployment in an environment without spatial bounds. However, this definition allows for a comparison between different sampling methods in a bounded experimental setup. Indeed, the former is the desired behaviour for closed static environments, where we assume growing redundancy of information in the readings with respect to the information already present in the map. This brings us to the first property, the density convergence, or simply convergence.

**Definition 3.3.2 Convergence.** Assuming that the readings come from the same, static environment, the number of points in the filtered map converges to a constant number $m$:

$$\lim_{t\to\infty} |\mathbf{F}(\mathcal{M}_t)| \to m \quad . \tag{3.2}$$

The convergence highlights the importance of keeping the number of points in a map bounded to prevent a memory overflow. It is thus a crucial property of any time unlimited deployment of a robotic system in a known environment.

**Definition 3.3.3 Idempotence.** Idempotence is the mathematical property describing an operation which can be applied multiple times without changing the result. In our case,

$$\mathbf{F}(\mathcal{M}) = \mathbf{F}(\mathbf{F}(\mathcal{M})) \quad . \tag{3.3}$$

A filter meeting this property is, for example, the voxel grid with a fixed origin. The points in voxels are sub-sampled only once, and no subsequent application of the filter will affect them. In contrast, the random filter is not idempotent, with each application removing more and more points. With an idempotent filter, we could, theoretically, unload parts of the map already sampled and only sample the parts where the reading ends, saving computation time.

We saw earlier that some filters approximate existing points by creating new points, for example, centroid, while others use exclusively points already present in the point cloud. This property is called image-inclusion.

**Definition 3.3.4 Image-inclusion.** The resulting point cloud is a subset of the original point cloud:

$$\mathbf{F}(\mathcal{M}) \subseteq \mathcal{M} \quad . \tag{3.4}$$

While the Image-inclusion is unsuitable for tasks like obstacle avoidance, it may be advantageous to replace a high number of points with their statistics, such as the mean and covariance. This, however, comes at the cost of replacing points with descriptors, so the gains in terms of saved memory can be limited. A more complex computation also takes more time.

We cannot define the linearity properly since it is not clear what a scalar multiplication of a point cloud means. Instead, we will only define the additivity:

$$\mathbf{F}(\mathcal{M} \sqcup \mathcal{P}) = \mathbf{F}(\mathcal{M}) \sqcup \mathbf{F}(\mathcal{P}) \quad . \tag{3.5}$$

The equation describes that the filtering operation has the same effect whether applied on the merged point clouds or on each of them individually. Together with the idempotence (Equation 3.3), any series of readings of length $t$ merged into the map can be simplified as

$$\mathcal{M}_t = \mathbf{F}(\mathcal{P}_0) \sqcup \mathbf{F}(\mathcal{P}_1) \sqcup \cdots \sqcup \mathbf{F}(\mathcal{P}_t) \quad . \tag{3.6}$$

This assumption would have, if possible, strong consequences. Instead of applying $\mathbf{F}$ on the map and the merged reading $\mathcal{M} \sqcup \mathcal{P}$, one could simply filter the reading. Filtering only the reading would provide significant time improvements since $|\mathcal{M} \sqcup \mathcal{P}| \gg |\mathcal{P}|$.

We can weaken the additivity by an approximation using a residual point cloud $\mathcal{R}$

$$\mathbf{F}(\mathcal{M} \sqcup \mathcal{P}) = \mathbf{F}(\mathcal{M}) \sqcup \mathbf{F}(\mathcal{P}) \sqcup \mathcal{R} \quad . \tag{3.7}$$

In fact, even a simple random filter does not hold this property. We will therefore work with the additivity in the number of points

**Definition 3.3.5 Additivity in number of points.** The number of points in a filtered point cloud, obtained by merging the reading and the map, is equal to the sum of numbers of points when filtering the reading and the map separately:

$$|\mathbf{F}(\mathcal{M} \sqcup \mathcal{P})| = |\mathbf{F}(\mathcal{M})| + |\mathbf{F}(\mathcal{P})| \quad . \tag{3.8}$$

**Definition 3.3.6** Monotonicity. Monotonicity, here defined in the number of points

$$|\mathbf{F}(\mathcal{M}_i \sqcup \mathcal{P}_{i+1})| \geq |\mathbf{F}(\mathcal{M}_{i-1} \sqcup \mathcal{P}_i)| \quad , \tag{3.9}$$

implies that the number of points in the map grows or stays constant over time. The monotonicity property is, in fact, equal to CR$\leq$100 %.

Controllability is a term we borrowed from the control system theory. In our case, the quantity we want to control is the number of points on the output of $\mathbf{F}$.

**Definition 3.3.7** Controllability. A sampling filter is controllable if

$$\forall\; l \in \langle 0, |\mathcal{M}|\rangle \;\exists\; A; |\mathbf{F}(A, \mathcal{M})| = l \quad , \tag{3.10}$$

implying the existence of a set of arguments $A$ that allow for direct control of the number of points $l$ after applying the filter. We deduce that if a filter is controllable, an appropriate choice of parameters makes it also monotonic. The controllability provides the user the picture of how many points to expect and therefore additional information for system design.

Finally, some filters, from their nature, consider the age of points.

**Definition 3.3.8** Age-biasness. In a time-limited experiment from $t_0 = 0$ to $t_f$, the age of a point $\mathbf{p}$ is the time $t$ when it was added into the map $\mathcal{M}$. We will call a filter aged-biased if

$$\exists \mathbf{p} \in \mathcal{M}_t |\; \exists t_1, t_2 |\; t < t_1 < t_2,\; p(\mathbf{p} \in \mathbf{F}(\mathcal{M}_{t_1})) > p(\mathbf{p} \in \mathbf{F}(\mathcal{M}_{t_2})) \quad , \tag{3.11}$$

with $p(\cdot)$ denoting the probability of a point being kept in the map after filtering. In other words, a filter $\mathbf{F}$ is age-biased if the probability that a point is kept in the map does decrease over time.

**Definition 3.3.9** Age-unbiasness. A filter is age-unbiased if it is not age-biased.

We can see an example of age-biasness on simulated data in Figure 3.4. The simulation consisted of generating readings in the form of cubes with 1 m of side size and four faces. The front and rear faces were missing to simulate a tunnel-like environment. The points were generated randomly with a fixed seed between different methods. At each iteration, the reading was merged into the map, and then the map was filtered. A new reading was generated, with a displacement in the direction of the x-axis corresponding to the iteration number. The overlap between generated point clouds was 0.5 m. This way, a point cloud of 10 m in length was created. As can be seen in the figure, some filters, such as `Random`, do not contain any points from the early iterations. On the other hand, density-aware filters are characterised by keeping a constant number of points per distance, as in the case of `Max Density`.

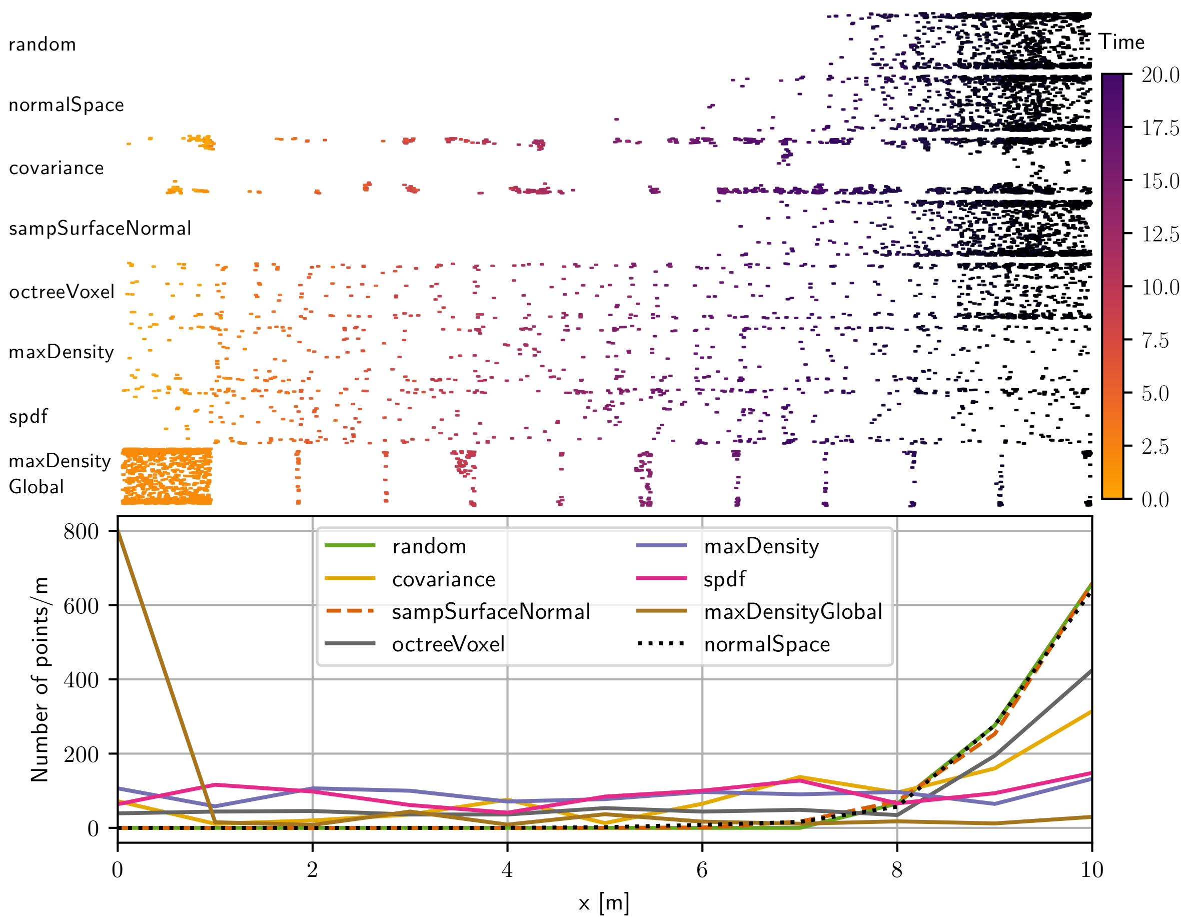


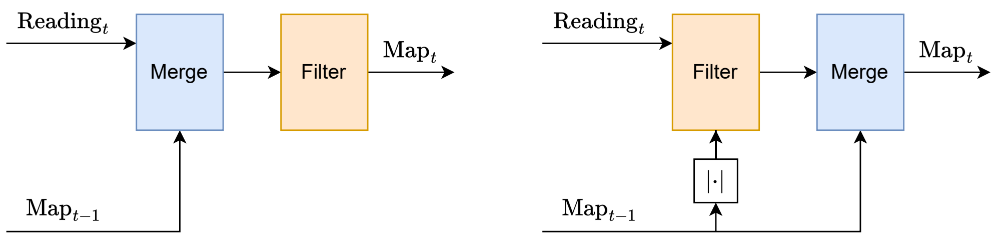
**Figure 3.4:** Age-bias effect of different filters. On the top: A top view of the resulting point clouds sampled by different filters. The colour of the points corresponds to the iteration when they were added to the map, ranging from 0 to 20. On the bottom: Point density as a function of bins displacement in the direction of the x-axis. The x-axes of the point clouds and the figure are aligned.

The age-biasness property has crucial implications for many robotics applications. For example, a teach-and-repeat scenario is a procedure where we first record a representation of the environment in the form of a map, together with the robot's trajectory in a teach task. The map and the trajectory are later used for localisation in the repeat task. Such a scenario cannot work with age-biased filters. Moreover, such a map could not be used for any subsequent online or offline tasks, such as planning, loop closure or inspection.

We will now describe the down-sampling filters implemented in the `libpointmatcher` library based on the properties defined above.

## 3.3.2 Random filter

The random filter is the most simple filter on our list. The points in the point cloud are kept with a probability set as an argument. Since the filter does not need any feature extraction or spatial division steps, it naturally executes the fastest of all filters on our list. The number of points in the output grows monotonically and can be controlled using the probability argument. The number of points in the map does not converge. The filter is not idempotent except for the trivial case of zero removal probability, is image-inclusive and additive in the number of points. However, the filter is age-biased, as we show in Figure 3.4. The age-biasness of the random filter is an implication of probability multiplication. The probability of still having a point $\mathbf{p_0}$ added at iteration 0 at iteration $n$ is $\prod_1^n(1-\frac{c}{100})$ with $c$ being the compression ratio. Therefore, we propose an alternative way to sample the point cloud randomly but without the age bias. Figure 3.5a shows two diagrams of the classical and the proposed approach. We already described the version on the left, which sustains age-biasness. The alternative version filters the reading with the help of additional information - the number of points in the map. Because of the additional information, we named it `Random Informed` filter. The `Random Informed` filters only the reading but ensures a correct compression ratio, since the filter can derive the desired number of points after the merge.



**(a) :** Random sampling **(b) :** Informed random sampling

**Figure 3.5:** Two versions of the random filter. While the original random sampling filters the map after the merge operation, the informed version capitalises on the information about the number of points in the map and samples the reading only. The symbol $|\cdot|$ denotes a function that return the number of points in an input point cloud

## 3.3.3 Octree Grid filter

The first representative of spatial methods in our selection is the octree grid filter. As described earlier, the filter divides the space into cubes and selects a representative sample point from each cube. The available sampling methods are: random, first point, medoid and centroid. The filter has two variants, depending on the condition selected to stop branching. The first criterion is an integer, corresponding to the the number of points in a leaf. The octree stops dividing when this number is reached. As for the second variant, the octree stops dividing either when there is only one remaining point in the leaf or when a given bounding box size is achieved. The parameter is a real number, determining the smallest box size in meters. To distinguish between them, we will denote the former version as `Octree Point` and the latter as `Octree Voxel`.

Both variants are spatial-based, meaning neither offers direct control of the number of output points. The `Octree Point` is strongly age-biased, because the number-of-points-based branching condition makes it create larger and larger cells in places already visited. As an example, let us consider a 2D version of the octree, the quadtree. We will set the branching condition to $n$ points. Now, we will generate a point cloud with $4n$ points. The points will lie in four clusters around the corners of an imaginary square. In the first iteration, the filter will divide the quadtree once into four quadrants. Then, the points in these quadrants will be filtered, leaving only one point left representing each corner. In the next iteration, the filter will not divide as the number of points in the initial square voxel is already smaller than the branching condition. It will still filter the remaining points, leaving only one point. The `Octree Voxel` can be called semi-age-biased. In fact, it removes points based on their age only in particular settings, related to the spatial distribution of points. The filter tends to remove points near the edges of the smallest cube, containing all points in the point cloud. Older points thus have a higher chance of being removed if the experiment is designed in a linear way. If we take a spiral instead, with the first points added in the centre, those points would not be affected by the filter's spatial behaviour.

Figure 3.6 shows an example of this on data acquired from the octree sampling executions of experiment in Figure 3.4 for both `Octree Point` and `Octree Voxel`. The medoid was used as the sampling method in this case. Firstly, while the solid lines correspond to the same data processing as in Figure 3.4, the dashed lines were generated differently. We first generated all the cubes statically,then we sub-sampled the resulting point cloud iteratively, applying the same filter 20 times. We see that both the final minimal x value as well the final number of points differ between the respective iterative and static variants. We also see that the variable initial cube size makes both variants non-monotonic since the number of points can drop dramatically as the cubes bounding boxes move and merge two previously adjacent voxels. The only case when the filters are monotonic is when the parameters ensure that no more than one point ends in a single cube. This is not present in the static case, where the points on the edge of the first cube converge to a static state and do not change anymore.

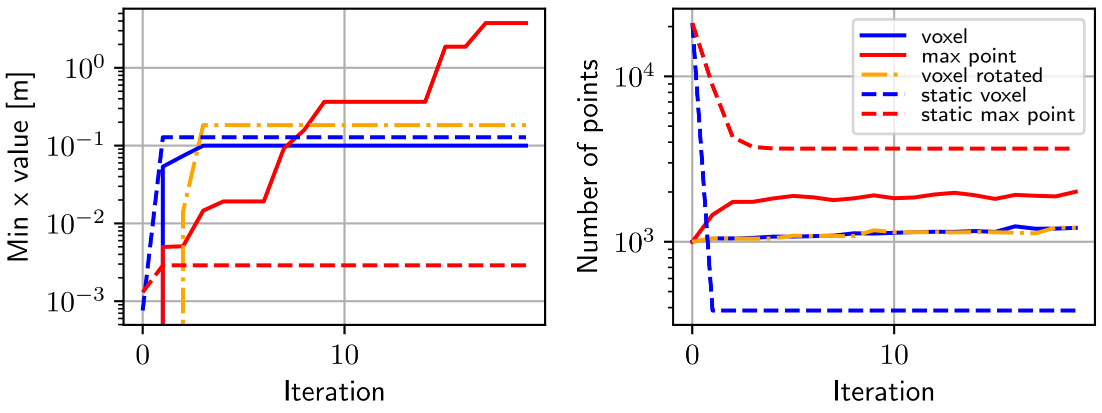


**Figure 3.6:** Rasterisation effect of the octree filter. Left: repeated rasterisation leads to a change in bounding box edge points. Right: resulting number of points in the point cloud differ based on octree type, coordinate frame and if applied statically or not.

The filter is also sensible to the frame in which is it applied, respectively to its orientation. To demonstrate this, the figure also contains lines depicting the minimal value in the direction of the x-axis and the number of points in a point cloud rotated by 30 deg in pitch. We can see that while the minimal x value is close to the one the static voxel setup converged to, the number of points is closer to voxel. We also see the idempotence holds for both static max size and static voxel from the third iteration. As spatial, the filters converge to a fixed number of points in a static, closed environment. For all sampling methods except centroid, they fulfil the image inclusion property. Finally, except for special cases such as the map and reading point clouds being at a specific distance from each other with no overlap, they are not additive.

### 3.3.4 Sampling surface normal filter

Sampling surface normal (SSNormal) is a spatial filter. It splits the space into a binary tree along the largest dimension, giving the two children an equal number of points. The recursive tree division is stopped when a given number of points in the leaf is reached. These points are used to compute a common normal before being decimated randomly, based on a ratio parameter. Similarly to the maxPoint version of the octree filter, the repeated sampling of points in boxes of increasing size makes the SNN strongly age biased. The method is controllable through the ratio parameter. Since the number of output points can be controlled, the user can also control the monotonicity property. Figure 3.7 shows the rasterisation effects of the SNN. Again, for the dashed lines, we generated all the data and then sub-sampled it iteratively 20 times. The difference is visible for both the minimal point x-axis value and the number of points. We see that for static versions, the number of points goes to 0 with as the exponential function $n \cdot r^i$, where $n$ is the initial number of points, $r$ is the ratio parameter, and $i$ is the iteration number. The method does not converge, is not idempotent and is image inclusive. SNN is not additive because of how the space is split.

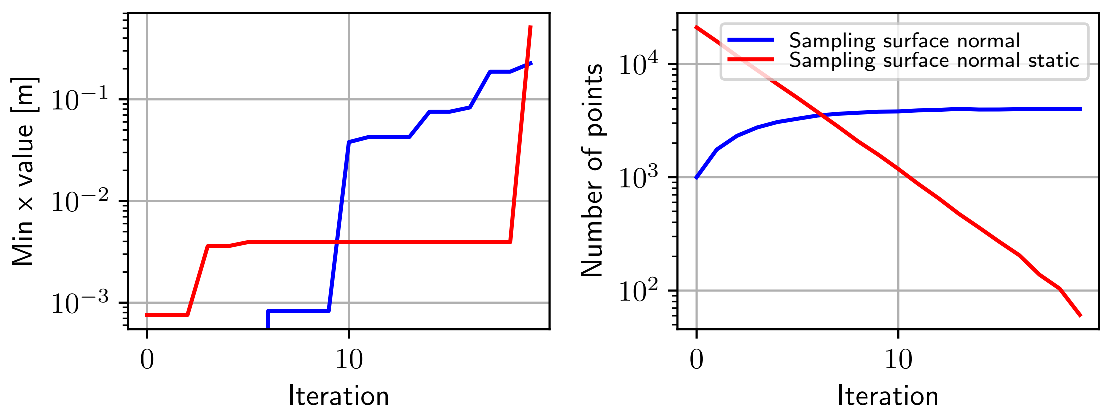


**Figure 3.7:** Rasterisation effects of the Sampling surface normal filter. Left: repeated rasterisation leads to a change of edge points for space splitting. Right: resulting number of points either converges to a constant value as new points are added or goes to 0.

### 3.3.5 Normal space and Covariance sampling filters

Unlike the SSN, which calculates the normals but does not use them to sample, NSS and Covariance filter come from a family of filters that rely on the use of normals for sampling. Both NSS and Covariance filters work locally and are controllable through the desired number of output points. If the user decides to keep the number of points non-decreasing, they are also monotonic. The NSS is aged biased, while the Covariance filter is not, but neither it does keep uniform densities. The actual outcomes of the Covariance filter are strongly implementation-dependent since multiple estimations and approximations can be performed throughout the calculation. Neither of the two filters converges to a constant number of points. However, both are idempotent, image inclusive and additive in the number of points. Labussière *et al.* [20] showed that the noise present in the data highly influences these two methods. The Covariance sampling performs best on small-scale point clouds with uniform density.

Normal space sampling projects the normals on a sphere and then uniformly subsamples points on this sphere. This, on the one hand, ensures that in the resulting point cloud, all normals are represented equally, as shown in Figure 3.8. However, normals only consider the translational components of the transformation. The rotation component can diverge even when the registration error lowers [44].

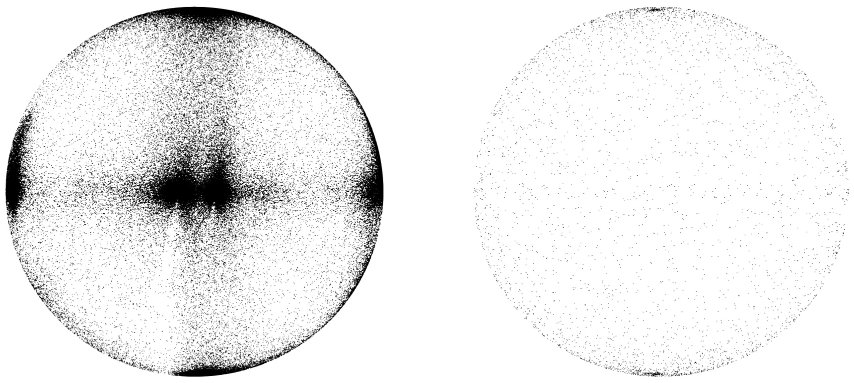


**Figure 3.8:** Unit sphere before and after normal sampling. The data were recorded in a structured office environment with right-angle junctions. Therefore most of the points on the left unsampled sphere are drawn from normal vectors of floors, ceilings and walls. The sphere on the right depicts the same data after sampling. F The higher density in pole regions is related to the implementation of uniform sampling on a sphere.

### 3.3.6 Maximum Density Filter

Lastly, we describe three density-based filters. The first one is the Maximum density filter. The filter makes the density homogeneous by rejecting points in high-density regions. It requires the data to contain the density descriptor. Currently, this descriptor in `libpointmatcher` is computed on spheres via a radius search. It is a local, non-controllable method, not age-biased and monotonic in the number of points. As a density-based method, it converges in density, making it a popular choice for time-unbounded applications. It is idempotent, no new points are created, and the filter is additive only for clouds with no overlap.

### 3.3.7 Global Maximum Density Filter

The Global maximum density filter is not part of the `libpointmatcher` library but instead is being applied before the concatenation of the reading into the map in `norlab_icp_mapper`. In another NNSearch, each point from the reading is assigned the nearest point from the map. If the distance between the two is higher than a threshold, the point's index is saved, and the point is later added to the map. Otherwise, the point is discarded. This ensures that no more points are added to the sphere defined by this point and a given radius for a given point in the map. However, there is an apparent weakness in this approach. While checking only the map, no distance measurement between the points in the scan is calculated. The filter thus expects an overlap between the reading and the map. Without the overlap, all points from the reading are added to the map. This is illustrated in Figure 3.4. All points from the first iteration are kept since the map is empty and no **NNSearchFF!** (**NNSearchFF!**) search is executed on the reading. Only the most distant points from the newly processed reading are added to the map in the following iteration. In this case, the minimal distance parameter was set to 1 m, so only points from every second iteration were merged.

While the naive solution of testing also the distance between points in the reading may be straightforward, an analysis of the computation complexity shows that it is not viable. Indeed, the computation complexity of checking for the distance of points newly added to the map is quadratic. An alternative approach would be to insert new points to the kd-tree and do a lookup in this tree instead. However, adding more points to the tree decreases lookup performance since the tree becomes unbalanced. Although this would not be a problem for small reading point clouds, `libnabo`,[3] the library for knn in low-dimensional spaces employed

[3]https://github.com/ethz-asl/libnabo

by both `libpointmatcher` and `norlab_icp_mapper` currently does not support the insert operation.

Otherwise, the filter is neither controllable nor age-biased. It is monotonic, image inclusive and in a close environment converges to a constant number of points. Because of its nature, we cannot describe this filter's idempotence and additivity properties.

### 3.3.8 Spectral Decomposition Filter

SpDF is based on Tensor voting, firstly introduced by Guy *et al.* [75] and a resulting decomposition of the point cloud into three distinct geometrical types: unction, curve and surface. Point cloud density is defined with respect to each of these geometrical primitives. The filter attempts to make the density uniform for each of them while labelling and rejecting outliers or primitives with low confidence. The filter iteratively decimates the primitives with values derived from the points' eigenvalues using saliencies. Spdf is not controllable, age-unbiased and due to its minor spatial nature, non-monotonic. It does converge in density, is idempotent, image inclusive and generally is not additive in the number of points.

Table 3.1 contains a summarised comparison of the filters and mathematical properties presented in this chapter. The filters chosen for final evaluation are: `Octree Voxel`, `Covariance`, `Random Informed`, `Max Density`, `Global Max Density` and `SpDF`. In the following chapter, we will first describe several experiments depicting the difficulties encountered when reconstructing an environment subject to these structures. Then, a deep comparison of the sampling strategies using several evaluation techniques will be presented.

| | Type | Converging | Idempotent | Img. inclusive | Additive in number of points | Monotonic | Controllable | Age-unbiased |
|---|---|---|---|---|---|---|---|---|
| `Random` | global | ✗ | ✗ | ✓ | ✓ | • | ✓ | ✗ |
| `Octree Point` | spatial | ✓ | ✗ | • | ✗ | ✗ | ✗ | ✗ |
| `SSNormal` | spatial | ✓ | ✗ | ✓ | ✗ | • | ✓ | ✗ |
| `NSS` | local | ✗ | ✗ | ✓ | ✓ | • | ✓ | ✗ |
| `SpDF` | local | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |
| `Random Informed` | global | ✗ | - | ✓ | - | • | ✓ | ✓ |
| `Covariance` | local | ✗ | ✗ | ✓ | ✓ | • | ✓ | ✗ |
| `Max Density` | local | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ |
| `Global Max Density` | local | ✓ | - | ✓ | - | ✓ | ✗ | ✓ |
| `Octree Voxel` | spatial | ✓ | ◊ | • | ✗ | ✗ | ✗ | ✓ |

**Table 3.1:** Types and properties of the selected sampling algorithms. Symbols ✓ and ✗ denote that the filter satisfies, respectively does not satisfy a property described in Section 3.3.1. The symbol • implies that the property depends on the choice of a set of parameters. Properties, labelled with the symbol - are not defined for the corresponding filter. `Octree Voxel` is, strictly speaking, not idempotent. However, as we saw in Figure 3.6, it quickly converges to a stable state. We denote this special characteristics with ◊. The horizontal line separates the filters in the bottom half, selected for further evaluation, from the rest.

# Chapter 4
# Experiments

The focus of the first part of this chapter is to verify how are thin structures represented in point clouds recorded with state-of-the-art sensors. We will comment and describe the employed sensors, as well as the experimental platform. We will then outline the main experiment conducted in the unstructured environment of the tunnels under Université Laval. The recorded trajectories and the ground truth data required several preprocessing and synchronisation steps, which we will describe. We will also present three various error metrics used for evaluation of the ICP trajectories.

## 4.1 Thin structures in 3D point clouds

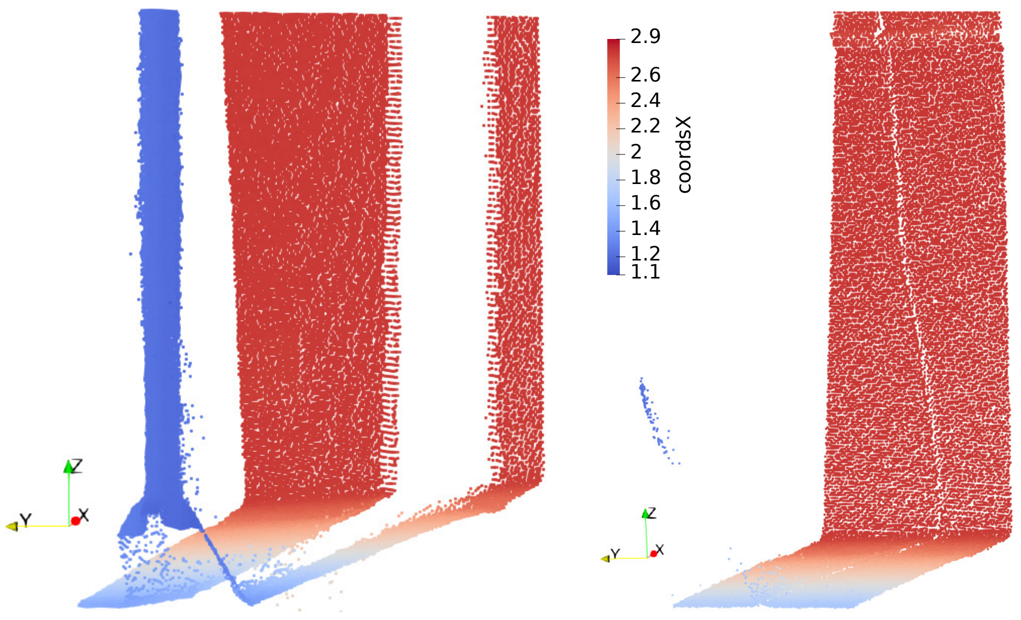
In this section, we will assess the quality of 3D representation of several thin structures, recorded with three LiDAR sensors and a surveying total station. We will inspect the representation from the perspective of both an individual scan and a point cloud created with the ICP algorithm. The three LiDARs used in these experiments were the LeiShen C16 [76], Robosense RS-LiDAR-16 [77] and Velodyne VLP-16 [9]. The number 16 indicates that each of them uses 16 channels. These sensors come from a family of small, middle-range LiDARs. Similar parameters, such as the 360 deg horizontal and 30 deg vertical fields of view, indicate that they target the same market. All sensors produce around 300 000 points per second and double that quantity in the dual mode. The maximal reported range differs, with the Velodyne offering 100 m, RS-LiDAR-16 150 m and LeiShen C16 up to 200 m of range distance. The range accuracy is 2 cm for RS and 3 cm for Velodyne and LeiShen. A summary of the parameters is presented in Table 4.1. The employed total station was the Trimble S7 [78], a scanner for

| | Channels | Horiz. FOV [°] | Vert. FOV [deg] | Rate [points/s] | Range [m] | Range accuracy [cm] |
|---|---|---|---|---|---|---|
| LeiShen C16 | 16 | 360 | 30 | 300k | 200 | 3 |
| Robosense RS-LiIDAR-16 | 16 | 360 | 30 | 300k | 150 | 2 |
| Velodyne VLP-16 | 16 | 360 | 30 | 300k | 100 | 3 |

**Table 4.1:** Parameters of the utilised LiDARs

surveying, imaging and 3D scanning. In the scanning mode, it has a operating range from 1 m up to 250 m with a minimal distance between points equal to 10 mm. The standard deviation of the measurements is in this mode 1.5 mm, and the accuracy of a single 3D point is 10 mm.

In the first experiment, we investigated the reconstruction of a thin rope with approximately 1 cm in diameter and a pole with 8 cm in diameter covered in the same material as the rope. The total station was located at little over 1 m distance from the objects, scanning both simultaneously in one experiment. The objects were about 1 m apart from each other. Figure 4.1 shows this as a point cloud scanned by the total station. While we can see that while the pole was recorded successfully, most of the rope was filtered out by the Trimble S7's software. Indeed, only a tiny section of points in the lower half of the point cloud was registered into the point cloud. These points are, however, noise, since the rope was hanging straight from the ceiling. The only other residue of the rope is the shadow visible on the back wall. The noisy points represent the phenomena of shadow, or artefact points [79]. Shadow points appear when only a part of the laser beam reflects from the obstacle, in our case, the rope, while the other part of the light beam continues and reflects from a different surface, typically a floor or a wall. The sensor's electronics then place the return in between of these two events, generating a false event.



**(a) :** Point cloud of the pole, recorded by the total station

**(b) :** Point cloud of the rope, recorded by the total station

**Figure 4.1:** Representation of thin objects in point clouds recorded by the Trimble S7 total station

Figure 4.2 depicts the top view of the pole from Figure 4.1a. The pole, drawn in grey, was, in fact, not a perfect circle nor perfectly homogeneous, but the approximation suffices to visualise the accuracy error of the scanner. As we can see, the error is truly in the range provided by the manufacturer, with points around 1 cm from the pole's surface. The points spread near the top and the bottom of the pole are shadow points. Some of them are, interestingly, in the direction opposite of the scanning.

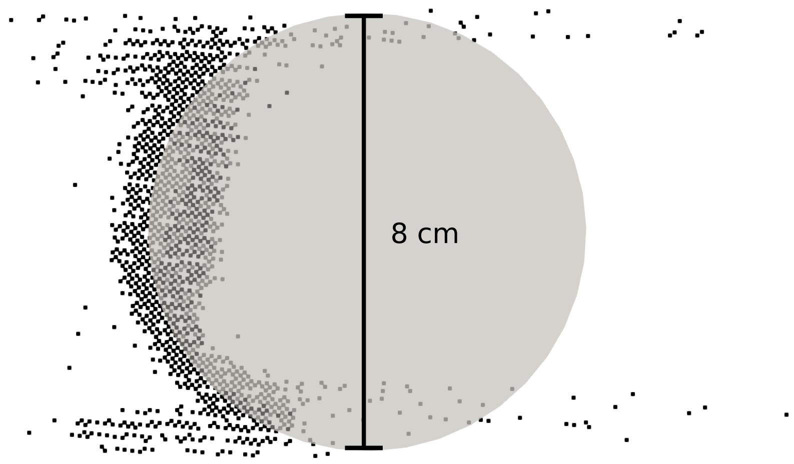


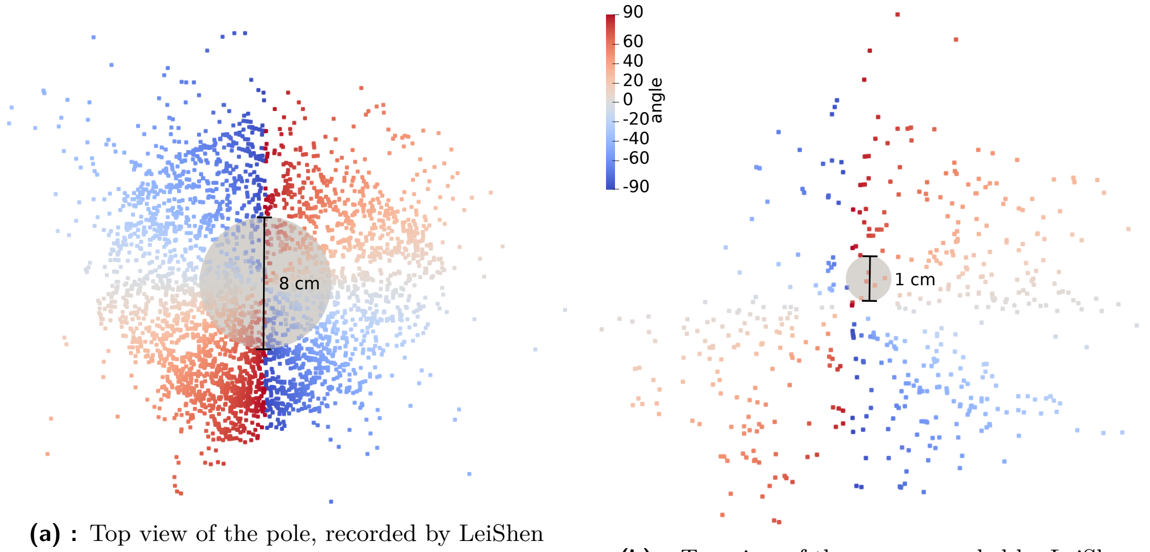
**Figure 4.2:** Top view of the pole, recorded by the Trimble S7 total station. The pole was scanned from left to right.

Now that we know how thin objects are represented in point clouds recorded with a high-accuracy device, we will investigate their reconstruction with the 3D LiDARs presented earlier.

Figure 4.3 contains top views of the pole and the rope, recorded with the LeiShen C16. Unlike in the case of the total station, the points do not come from a single scan but from a map created with the ICP algorithm. A mobile platform equipped with the LiDAR was driven manually around the pole or the rope. The readings and odometry priors were fed in the `norlab_icp_mapper`, which produces the final map point cloud. The colour of points corresponds to the observation angle by the robot, calculated as

$$\theta = \frac{180}{\pi} \cdot \arctan\left(\frac{y - c_y}{x - c_x}\right) \quad ,$$

with $\mathbf{c} = [c_x, c_y]^T$ being the centre of the object and $x$, $y$ the x and y coordinates of points in the point cloud. We can immediately see that compared to Figure 4.2, the deviation of points is much higher for the pole. Even though we can still distinguish the pole's shape and boundaries, the diameter is approximately three times the actual value. The boundaries are not distinguishable for the rope, where the LiDAR registered fewer reflections, the resulting point cloud is less dense and does not resemble a cylindrical object.

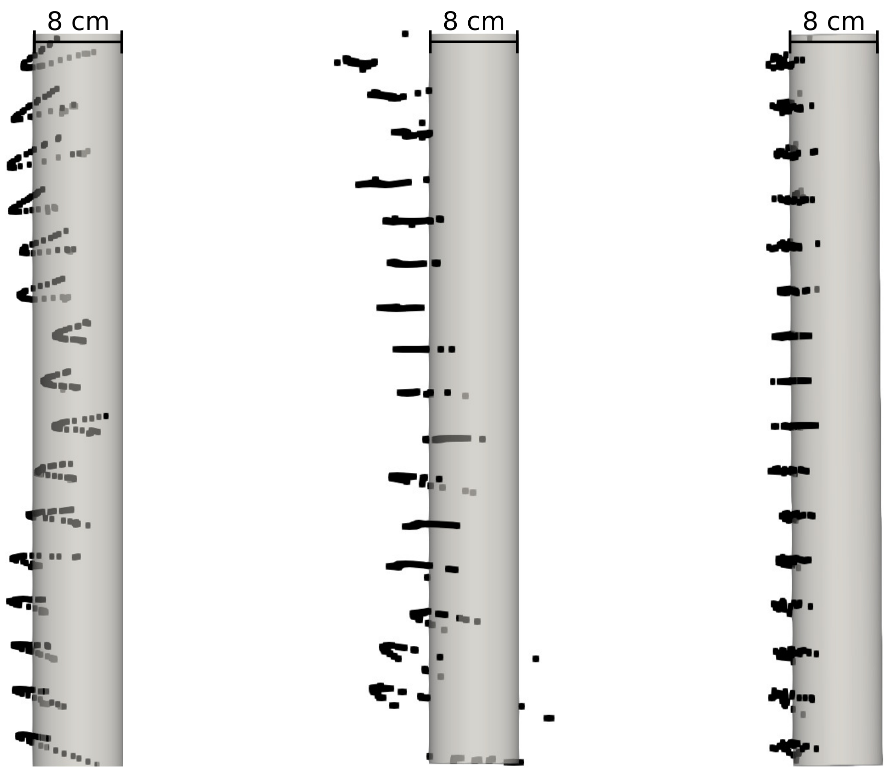


**(a) :** Top view of the pole, recorded by LeiShen C16

**(b) :** Top view of the rope, recorded by LeiShen C16

**Figure 4.3:** Top view of thin objects, recorded by LeiShen C16 and registered in the map using the ICP algorithm

Further examining the inflated diameter causing an error much greater than the accuracy given by the manufacturer, we discovered that the error differs based on the height. Figure 4.4a depicts a side view of a single pole scan, with returns on all 16 channels. While the points in an individual channel nicely copy the object's surface, the channels themselves are not synchronised. The inclination of points in channels portrays the vertical angle between individual channels inside the LiDAR, with the sensor being located roughly at the half height of the depicted pole. Indexing the channels from 1 to 16, bottom to top, we see that channels on indices 1-5 underestimate the distance to the object. Channels 6 and 7 are the closest to the true object boundary; channels 8-10 overestimate the distance, and the remaining channels once again underestimate it. This phenomenon increases the depth of objects artificially, creating thick walls at places where we should see only planar surfaces. Figure 4.4b and Figure 4.4c depict the same situation with the two other LiDARs. While the RS-LiDAR-16 shows even deeper synchronisation issues than the LeiShen, the Velodyne performs best. Since LiDAR channel calibration, such as the one in [80], is out of the scope of this work, we will further use the Velodyne VLP-16 only.



**(a) :** Side view of the pole, recorded by LeiShen C16

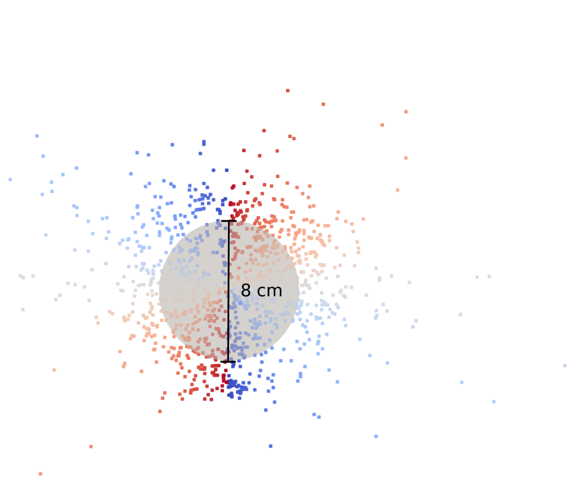
**(b) :** Side view of the pole, recorded by RS-LIDAR-16

**(c) :** Side view of the pole, recorded by Velodyne VLP-16

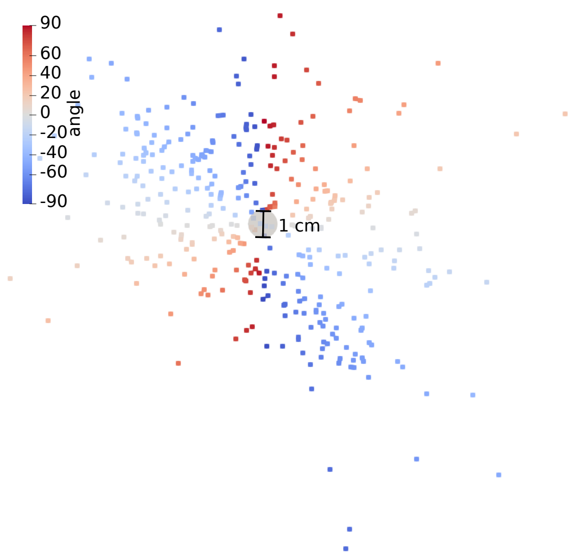
**Figure 4.4:** Side view of the pole as individual scans, recorded by different LiDARs

Finally, to offer a comparison between LeiShen C16 and Velodyne VLP-16, we conducted the same experiment of driving a robot equipped with the LiDAR around the pole and the rope. The results of this experiment are illustrated in Figure 4.5. While we can visually observe an improvement in the recording of the pole, this cannot be said about the rope. In the case of the pole, the points lie closer to the pole's surface. Except for several outliers, most of the points lie in the circle with twice the diameter of the actual value. As for the rope, the error is higher in the observation direction of 90 deg. This can be caused by the the rope being observed from this direction for a longer time. Nevertheless, we see again that the point cloud does not resemble a cylindrical object and that the estimated diameter would be entirely off. We conclude that the rope is too small compared to the accuracy and beam divergence of today's LiDARs. Therefore, in future experiments, we will focus on objects similar to or larger than the pole.

The reader may argue that the material and the shape of the studied object also plays a role in the laser beam reflection. While we agree with that, these properties go beyond the scope of this thesis. An analysis of noise characterisation of different surfaces, namely an aluminium plate, whiteboard, steel and rusted iron plates were discussed by Pomerleau *et al.* [81]. The observations were, however, only conducted on planar surfaces.



**(a) :** Top view of the pole, recorded by Velodyne VLP-16



**(b) :** Top view of the rope, recorded by Velodyne VLP-16

**Figure 4.5:** Top view of thin objects, recorded by Velodyne VLP-16 and registered in the map using the ICP algorithm

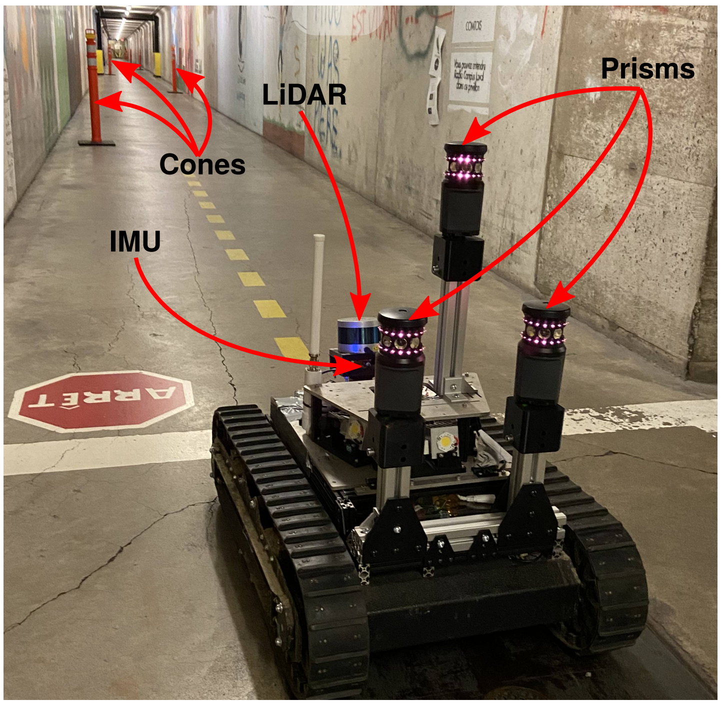
## 4.2 Experimental setup

In this section, we will describe the experimental setup, including the robotic platform and the environment where the experiments took place. We will then detail the data generation and preprocessing of the recorded trajectories, needed for the final evaluation. Finally, we will introduce three error metrics used for the evaluation of the localisation error.

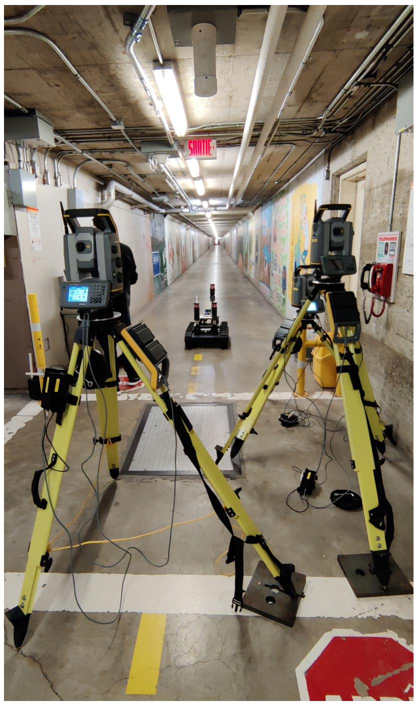
### 4.2.1 Experimental platform and environment

The experimental robotic platform, called Marmotte, is built upon the HD2 Treaded Tank Robot Platform [82]. The platform's four IG52-04 24VDC 285 RPM Gear Motors with encoders are controlled with two Roboclaw 30A motor controllers. The robot is powered by two lithium iron phosphate batteries with 25.6 V, providing approximately two hours of battery life. The computations are executed on a Dell Optiplex 3070 computer. Although the robot is also equipped with an Nvidia Xavier, this computer is destined for more computation-intensive tasks, such as image processing, and is not used in our experiments. Apart from several cameras that were not used during our experiments, the platform is equipped with a Velodyne VLP-16 LiDAR and an Xsens MTi 100 IMU. Both these sensors are attached to a common 3D printed part, with the IMU being attached upside-down to the bottom. This 3D part is slightly inclined with about 10 deg in pitch. The inclination makes a large part of the scan reflect from the floor in front of the robot, while the floor behind it is mostly not visible. Points reflected from parts of the robot were removed with a bounding box filter, because they would otherwise create a trail of points in the map point cloud $\mathcal{M}$. The robot is depicted in Figure 4.6a in the environment where the experiments took place. In the picture, we can also notice three total station prisms, one in the robot's centre at a higher position and the other at the back of the robot, at equal height. Although the manufacturer recommends the prisms to be placed at least 80 cm from each other, we did not encounter any problems with this setup. The prisms were tracked by the three Trimble S7 total station, visible in Figure 4.6b. The tracking enabled us to acquire a ground truth trajectory.

The experiments took place in the tunnels under Université Laval. The experiments were conducted over a distance of approximately 150 m, with the robot being manually controlled and driving straight through the tunnel. Figure 4.7 shows a drawing of the setting where



**(a) :** The experimental platform, robot Marmotte, in the tunnels



**(b) :** Three total stations, providing a ground truth trajectory, in the tunnels

**Figure 4.6:** The experimental platform and the ground truth acquisition setup in a tunnel under Université Laval. The platform is equipped with a Velodyne VLP-16 LiDAR and a Xsens MTi 100 IMU. The three surveying total stations Trimble S7 were tracking the prisms attached to the robotic platform.

the experiments took place. The robot started at the intersection visible in Figure 4.6a. The three total stations recording the prism movement were located at the same place. During three runs through the environment, the robot experienced three different levels of constrainess. In the first level, more than 25 objects were placed in the environment, leaving an empty lane in the middle of the corridor for the robot to drive through. The obstacles were different objects, such as crates and boxes or a trolley (depicted in green in Figure 4.7) and 10 construction reflective cones (in blue). The cones had 1.25 m in height and 0.1 m in width. The goal of this setup was to artificially create enough constraints in the tunnel, simulating a well-constrained environment with many structures for the ICP to use. The acquired trajectory will later be used for a reference comparison with the other experiments. Most objects were removed for the second experiment, and only the 10 cones were kept. The 10 cones served as the main experiment, which was later used to evaluate different subsampling methods. Finally, we also removed the cones and advanced with the final experiment in an empty corridor. Here we aimed to record data in a scenario where we knew the ICP would fail since the environment did not contain any constraints in the direction of the robot's movement.

A small niche can be found roughly in the middle of the tunnel. For the time of the empty and the 10 cones experiments, we covered it with a canvas (in red) to further increase the difficulty of localisation in the tunnel. Thus, the only major constraint left for the empty experiment was the supporting metal structure, located at the beginning of the third third of the tunnel (in purple). Other minor objects, such as cables, lights or networking antennas also unavoidably remained in the tunnel. Finally, for both the empty and the 10 cones experiment, we applied a bounding box filter to remove all the points near the origin of the robot's movement. Otherwise, because of how the LiDAR is installed on the platform, these points would help the localisation algorithm in the first part of the run. We will further refer to the three experiments as *empty*, *10 cones* and *constrained*.

For all three experiments, the robot travelled from the first to the second intersection, turning around and coming back to its initial position. However, we decided not to use the return part of the data for the evaluation. Firstly, the ground truth data were noisier in this case due to the middle prism temporarily blocking the sight between the prisms placed at the back of the robot and their total stations. Secondly, the one-way experiment demonstrates sufficiently the drift that happens with the robot travelling in an underconstrained environment.

The robot was used only for data recording in the form of `rosbag` files acquired with the Robot Operating System (ROS) [83]. The recorded data messages were:

- IMU messages published at 400 Hz
- Wheel odometry messages published at 30 Hz
- Raw point clouds published at 10 Hz
- Total station messages published at 2.5 Hz
- Time

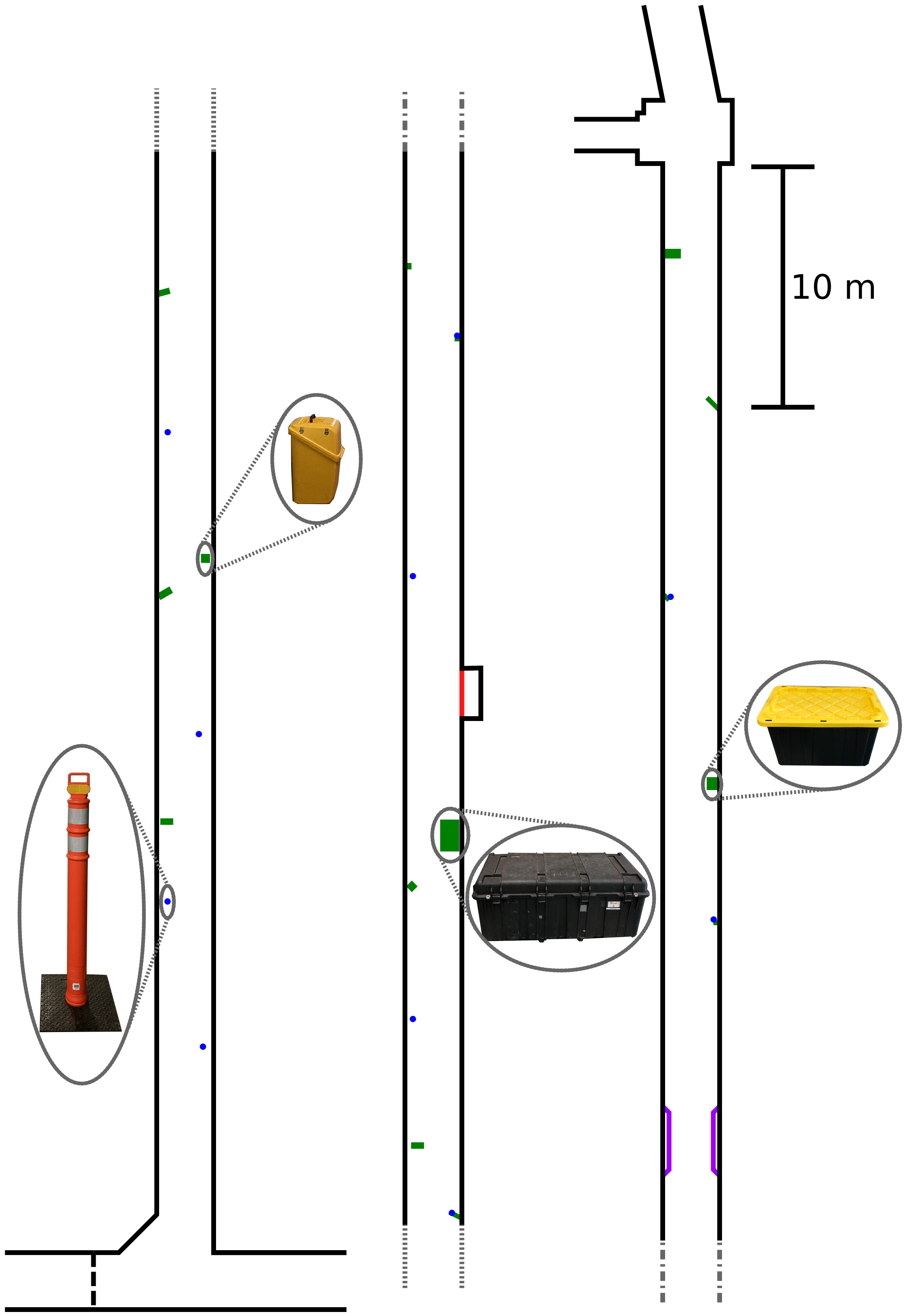


**Figure 4.7:** Scheme of the tunnels with different obstacles and their placements. The green obstacles were present only for the *constrained* experiment, with the cover of the niche in the middle removed (in red). Blue obstacles represent reflective cones, that were used for both *constrained* and *10 cones* experiments. The supporting metal (purple) is an inseparable part of the tunnel. The diagrammatic coloured obstacles are in scale. The total distance between the intersections is about 150 m.

### 4.2.2 Trajectory generation and preprocessing

We will start this section by describing the origin of the ground truth trajectory. Vaidis *et al.* [84] presented preliminary work on measuring a full 6DOF pose of a vehicle using total stations. Although we used a similar setup, we limited ourselves to only 3DOF, i.e., the robot's position in the tunnel. The reasons for this are twofold. Firstly, we made several simplification assumptions in our experimental setup. We assume that, for the most part, the surface the robot is moving on is flat, which removes the roll and pitch degrees of freedom, and the robot was moving at a constant speed with a fixed heading. We keep the z-coordinate to account for any errors coming from the tunnel bending. Secondly, the full pose trajectory protocol was not stable enough at the time of writing this work.

Each of the three total stations generates messages at 2.5 Hz. After transforming from the spherical to Cartesian coordinates and filtering, we have three trajectories in the coordinate frame $\mathcal{A}$, one for each total station-prism pair

$$ {}^{\mathcal{A}}_{\Theta}\mathbf{Q}_k = \{\mathbf{p}_{t_1}, \dots \mathbf{p}_{t_l}\}\,;\ \mathbf{p}_{t_i} \in \mathbb{R}^3;\ i \in [1, \dots, l];\ k \in \{1,2,3\} \quad , $$

where $k$ is the total station's index, $l$ is the number of positions $\mathbf{p}_{t_i}$ in the trajectory, recorded at the time $t_i$. $\Theta$ is the time offset between the time of the total stations and the rest of the system. This offset is unknown and differs between individual experiments. Computing the mean value of the three trajectories over points $\mathbf{p}_{t_i,k}$, e.g., points at time $t_i$ of the total station $k$

$$ {}^{\mathcal{A}}_{\Theta}\mathbf{Q} = \left\{ \frac{1}{3} \sum_{k=1}^{3} \mathbf{p}_{t_i,k} \middle|\ i \in [1, \dots, l] \right\} \quad , $$

we obtain the estimate of the centroid of the prism positions on the robot. The fusion of the translation vector from the wheel odometry with the orientation acquired from the IMU data with the Madgwick filter [85] gives us the odometry trajectory in the coordinate frame $\mathcal{B}$

$$ {}^{\mathcal{B}}\mathbf{O} = \{\mathbf{T}_{t_1}, \dots \mathbf{T}_{t_m}\}\,;\ \mathbf{T}_{t_i} \in SE(3);\ i \in [1, \dots, m] \quad , $$

where $SE(3)$ denotes the special Euclidean group of rigid transformation. We use this trajectory three times. Firstly, to synchronise the ground truth ${}^{\mathcal{A}}_{\Theta}\mathbf{Q}$ in time with the rest of the system. To do this synchronisation, we compute the instantaneous speeds for both ${}^{\mathcal{A}}_{\Theta}\mathbf{Q}$ and ${}^{\mathcal{B}}\mathbf{O}$. This gives us $\mathbf{v_Q} = \{v_{t_1}, \dots, v_{t_l}\}$ and $\mathbf{v_O} = \{v_{t_1}, \dots, v_{t_m}\}$. Drawing functions that linearly interpolate speeds from these two sets, we get functions: $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$. Using the first $T_1$ seconds of both trajectories, the synchronisation can be formulated as the optimisation problem

$$ \Theta^* = \arg\min_{\Theta} \int_0^{T_1} \|f(t-\Theta) - g(t)\|^2 \, dt \quad . \tag{4.1} $$

After shifting the ground truth positions in time with $\Theta^*$, we finally get the reference trajectory synchronised in time. We will further denote this trajectory as ${}^{\mathcal{A}}\mathbf{Q}$. The second time we use the odometry trajectory ${}^{\mathcal{B}}\mathbf{O}$ is to provide an initial transformation estimate for Algorithm 2, which outputs, together with the map, a trajectory of the ICP estimate in the coordinate frame $\mathcal{C}$

$$ {}^{\mathcal{C}}\mathbf{P} = \{\mathbf{T}_{t_1}, \dots \mathbf{T}_{t_n}\}\,;\ \mathbf{T}_{t_i} \in SE(3);\ i \in [1, \dots, n] \quad . $$

Lastly, we propose using the odometry trajectory ${}^{\mathcal{B}}\mathbf{O}$ to align the trajectories into a common frame. We could, alternatively, use part of or the whole estimate trajectory ${}^{\mathcal{C}}\mathbf{P}$ for this calibration as proposed by Umeyama [86] and Horn [87]. Nevertheless, calibrating with a section of the estimate trajectory ${}^{\mathcal{C}}\mathbf{P}$ would prevent us from using that trajectory section in the evaluation. This is because the alignment usually minimises the least square position error [88]. Since the position error is the quantity we want to measure in the quantitative evaluation, we would only measure the residual of the position alignment instead of the error.

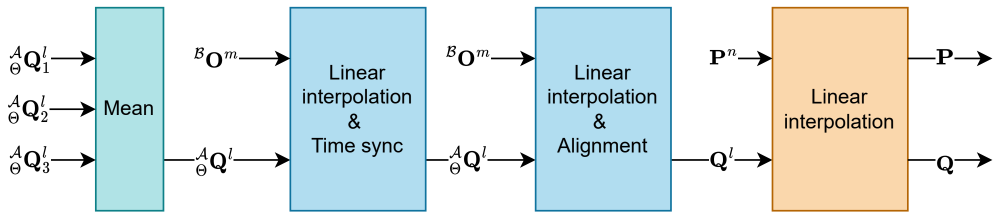
Furthermore, our trajectories do not contain any initial alignment sequence, such as a zig-zag movement. A trajectory like ours, i.e., a straight line carried out by the robot through the tunnel, has one free degree of freedom in the roll angle, around which the least square alignment could rotate freely. Instead, we propose to use the odometry trajectory ${}^{\mathcal{B}}\mathbf{O}$. The transformation ${}^{\mathcal{B}}_{\mathcal{C}}\mathbf{T}$ between coordinate frames of the estimate $\mathcal{C}$ and the odometry $\mathcal{B}$ is static and known in advance. To acquire the transformation from $\mathcal{A}$ to $\mathcal{B}$, let $\gamma : \mathbb{R}, SO(3) \to \mathbb{R}^3$ be a function that takes time $t$ and a rotation $\mathbf{R} \in SO(3)$ and outputs a point $\mathbf{p} \in \mathbb{R}^3$ corresponding to the rotated linearly interpolated point from the reference ${}^{\mathcal{A}}\mathbf{Q}$. Let $\zeta : \mathbb{R} \to \mathbb{R}^3$ be a similar function that takes time $t$, linearly interpolates and returns a point $\mathbf{p} \in \mathbb{R}^3$ corresponding to the linearly interpolated pose from the odometry ${}^{\mathcal{B}}\mathbf{O}$. Assuming all the trajectories start at the origin, we can use the first $T_2$ seconds to find the optimal rotation, parameterised by the roll, pitch and yaw angles $\phi, \psi, \theta$

$$\mathbf{R}^* = \underset{\mathbf{R}(\phi,\psi,\theta)}{\arg\min} \int_0^{T_2} \|\gamma(t, \mathbf{R}(\phi,\psi,\theta)) - \zeta(t)\|^2 \quad . \tag{4.2}$$

The roll angle is bounded to prevent the calculation of an inverted solution. Using the optimal rotation $\mathbf{R}^*$ to get the transformation ${}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T}$, we can transform all trajectories to the common frame $\mathcal{B}$:

$$\begin{aligned}
\mathbf{Q} &= \{{}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T} \odot \mathbf{p}_{t_i} | \mathbf{p}_{t_i} \in {}^{\mathcal{A}}\mathbf{Q}\}, \\
\mathbf{O} &= {}^{\mathcal{B}}\mathbf{O}, \\
\mathbf{P} &= \{{}^{\mathcal{B}}_{\mathcal{C}}\mathbf{T} \cdot \mathbf{T}_{t_i} | \mathbf{T}_{t_i} \in {}^{\mathcal{C}}\mathbf{P}\} \quad .
\end{aligned}$$

Finally, to be able to evaluate the trajectories, we linearly interpolate in time the reference $\mathbf{Q}$ with points from $\mathbf{P}$, so that $\mathbf{Q} = \{\mathbf{p}_{t_1} \dots \mathbf{p}_{t_n}\}$. Because we will not use full 6DOF poses for the evaluation, we can simplify the estimate trajectory to only its translation part: $\mathbf{P} = \{\text{trans}(\mathbf{T}_{t_1}) \dots \text{trans}(\mathbf{T}_{t_n})\}$. Both $T_1$ from Equation 4.1 and $T_2$ from Equation 4.2 were set to 20 s. The preprocessing pipeline is summarised in Figure 4.8



${}^{\mathcal{A}}_{\Theta}\mathbf{Q}^l_1$
${}^{\mathcal{A}}_{\Theta}\mathbf{Q}^l_2$
${}^{\mathcal{A}}_{\Theta}\mathbf{Q}^l_3$
Mean
${}^{\mathcal{B}}\mathbf{O}^m$
${}^{\mathcal{A}}_{\Theta}\mathbf{Q}^l$
Linear interpolation & Time sync
${}^{\mathcal{B}}\mathbf{O}^m$
${}^{\mathcal{A}}_{\Theta}\mathbf{Q}^l$
Linear interpolation & Alignment
$\mathbf{P}^n$
$\mathbf{Q}^l$
Linear interpolation
$\mathbf{P}$
$\mathbf{Q}$

**Figure 4.8:** Block diagram of the trajectory preprocessing

We will now describe the parameters used in the ICP and the mapper that were fixed for the experiment. We already mentioned that the mapper uses the odometry $\mathbf{O}$ as prior. In addition, we apply random perturbation drawn from a uniform distribution $\mathcal{U}[-0.25, 0.25]$ in the direction of the robot's movement, the x-axis, to each pose of the trajectory, starting from seed $s_1$. The point cloud readings are deskewed in a preprocessing step [10] and then randomly sampled with a compression ratio of 80 %, starting from the same seed. Although our data processing is not real-time, the random sampling preprocessing is still essential to allow for reasonable computation times, as showed by Baril *et al.* [89]. Fixing the seed ensures fair conditions for the evaluation of different subsampling methods.

The error minimiser was set to 4DOF point-to-plane. The use of IMU roll and pitch angle estimate requires precise calibration of the LiDAR-IMU coordinate frames. As suggested by Kubelka *et al.* [18], we firstly calibrated the angles between the two sensors on a flat floor, adjusting the roll and pitch value until the LiDAR reading appeared flat on the surface. We fine-tuned the transformation with a long circular mapping run, with a length of approximately 400 m, at the end returning to the initial position. The pitch angle correction was obtained from the tangent between the total loop distance and the elevation error. Since the IMU and LiDAR are attached to a shared rigid part, rotating this part by 90 deg and repeating the procedure allowed us to find the correction of the roll angle.

The update condition of the mapper was set to the time difference. As a matter of fact, this is the only condition that is working in our scenario. The distance difference does not function properly in the situation where the robot, from the perspective of the ICP, stays in place, for example, due to an underestimated position. Contrary to that, the overlap condition breaks when the position is overestimate, and there is no sufficient overlap. To find the best delay value to generate trajectories, we evaluated two sampling methods with changing values of the update delay. The two methods were the `Max Density` and the `Octree Voxel` with sampling set to medoid. We sampled the delay values between 0 s and 2 s with a step size of 0.1 s. For each delay value, we generated ten trajectories using the pipeline described above. We used the *constrained* dataset, together with zero random perturbation. We evaluated the trajectories using three error metrics which we will describe in detail in the next section.

Figure B.1 shows how the error values evolve with rising update delay. We can see that the `Octree Voxel` is more stable, while the `Max Density` experiences a decrease of error after 0.2 s. For the Relative explored distance error (REDE) and Relative travelled distance error (RTDE) metrics, the error values stay nearly flat after the initial decrease of `Max Density`. The Relative point distance error (RPDE) shows that the error is slowly rising. Finally, we decided to set the time difference's value to 1 s. This value showed among the best for explored and travelled distance metrics, while the difference for point distance error is less than 1.0 %. The condition, when and where to update the map depends on many variables, including the employed sensors, robot speed and the environment. Considering that the map update is a computationally intensive task that gets more lengthy with an increasing number of points, it is not reasonable to update the map too often, mainly if the environment contains lots of redundant information. In our case, the chosen value seems to provide a good compromise between the error and computation complexity.

### 4.2.3 Error metrics and trajectory evaluation

In this section, we will describe three error metrics used in the results evaluation process. The access to the ground truth allows us to avoid the calculation of a map matching metric or the visual comparison of individual maps. Furthermore, since our system randomly subsamples the input point cloud to 20 %, we must repeat each localisation and mapping run multiple times. Otherwise, the results would be strongly influenced by randomness and would not allow for a fair comparison between different map sampling configurations. Unlike visually comparing maps, the trajectory evaluation lets us aggregate multiple localisation and mapping runs using standard statistics.

The three error metrics are: Point distance error, Travelled distance error and Exploration distance error, all measured in meters. The latter two have the advantage of not requiring a trajectory alignment. Under the assumption that the drift only happens in one direction and there is no slip between the robot's wheels and the ground, the travelled and exploration distance errors could be, theoretically, used with only the wheel odometry as the ground truth. We will use the relative variants of all three error metrics. The reported value will thus be a percentage. The advantage of a relative metric is that it gives the drift, i.e., the information about the local accuracy of the trajectory over an interval $\Delta \in \mathbb{N}$. Absolute error is, on the other hand, suitable for evaluating a global consistency of a trajectory. It also has an intuitive visual representation. However, it can hide some important details, and the number of data points is limited to the number of points in the trajectory [90]. It is also sensitive to the time when the error appears.

Let us remind the reader of the two trajectories with which we finished the previous section: the reference $\mathbf{Q} = \{\mathbf{q}_{t_1} \dots \mathbf{q}_{t_n}\}$ and the estimate $\mathbf{P} = \{\mathbf{p}_{t_1} \dots \mathbf{p}_{t_n}\}$. We will now define a function

$$d(\mathbf{G}, t_i, \Delta) = \sum_{t_j = t_i}^{t_{i+\Delta-1}} \|\mathbf{g}_{t_{j+1}} - \mathbf{g}_{t_j}\|_2 \quad , \tag{4.3}$$

which takes a trajectory $\mathbf{G} = \{\mathbf{g}_{t_1} \dots \mathbf{g}_{t_n}\}$, time $t_i$ and time interval $\Delta$ and returns the distance travelled in this trajectory between timestamps $t_i$ and $t_{i+\Delta}$.

Let $e(\mathbf{P}, \mathbf{Q}, t_i, \Delta)$ be a general error function which returns a relative error distance. Algorithm 3 extends the relative error calculation to the full trajectories. The algorithm takes the reference and the estimate as inputs, together with a Boolean variable allPairs. This Boolean describes whether or not to use overlapping relative poses or only consecutive ones. On the one hand, using overlapping poses gives us more data points. On the other hand, the overlapping version also contains lots of redundant data. We found out that the alterations in results were negligible; therefore, we only used the consecutive version of the algorithm. The algorithm fills and outputs two arrays, the errors and relDistances.

Saving the relative travelled distance allows us to compare experiments with a non-equivalent total travelled distance. We will also use the relative distances to filter parts of the trajectories. More specifically, we will filter out the errors corresponding to small displacements because, in their case, the ICP noise can have a greater magnitude than the travelled distance. An example of this is when the robot is static, and Equation 4.3 only returns a small random noise. When we divide by this number, the errors are, clearly, colossal. Higher travelled distance values correspond to larger $\Delta$. These can be inaccurate since they penalise more angular errors happening near the beginning of the trajectory than near the end.

**Algorithmus 3** Trajectory evaluation

```
Require: $\mathbf{Q} = \{\mathbf{q}_{t_1} \dots \mathbf{q}_{t_n}\}$          ▷ Reference trajectory
Require: $\mathbf{P} = \{\mathbf{p}_{t_1} \dots \mathbf{p}_{t_n}\}$          ▷ Estimate trajectory
Require: allPairs                                  ▷ Boolean
 1: $\Delta = 1$
 2: errors $\leftarrow \emptyset$, relDistances $\leftarrow \emptyset$
 3: $d_l \leftarrow d(\mathbf{Q}, 0, l)$
 4: while $\Delta < n$ do
 5:     if allPairs then $s \leftarrow 1$ else $s \leftarrow \Delta$ end if
 6:     while $i \leq n - \Delta$ do
 7:         $e \leftarrow e(\mathbf{P}, \mathbf{Q}, t_i, \Delta)$
 8:         $d \leftarrow d(\mathbf{Q}, t_i, \Delta)$
 9:         APPEND(errors, $e$)
10:         APPEND(relDistances, $d/d_l$)
11:         $i \leftarrow i + s$
12:     end while
13:     $\Delta \leftarrow \Delta + 1$
14: end while
15: return errors, relDistances
```

We will now introduce several functions $e(\cdot)$.

**Definition 4.2.1** Relative point distance error (RPDE). The Relative point distance error

$$\mathrm{RPDE}(\mathbf{P},\mathbf{Q},t_i,\Delta) = \frac{\|\mathbf{q}_{t_{i+\Delta}} - (\mathbf{p}_{t_{i+\Delta}} - \mathbf{p}_{t_i} + \mathbf{q}_{t_i})\|_2}{d(\mathbf{Q},t_i,\Delta)} \quad , \tag{4.4}$$

is the point-to-point distance between the estimate and the reference positions in time $t_{i+\Delta}$. The distance is divided by the distance travelled by the robot in the reference trajectory between time $t_i$ and $t_{i+\Delta}$.

The local estimate and reference trajectories are aligned by subtracting $\mathbf{p}_{t_i}$ and adding $\mathbf{q}_{t_i}$ to the estimate. This ensures that we measure the drift appearing only between $t_i$ and $t_{i+\Delta}$, because in $t_i$ we always start with a zero drift. The same denominator will also be used for the other two error metrics. While the evident advantage of the RPDE is that is measures error in all three coordinates, it requires the two trajectories to be aligned. The translation effected in Equation 4.4 aligns the trajectories spatially, and they need to be also aligned in rotation, which we achieved in Equation 4.2.

Contrary to RPDE, the following two error metrics shrink the 3D trajectories into one-dimensional vectors and thus do not need any alignment as long as the two trajectories are synchronised in time.

**Definition 4.2.2** Relative travelled distance error (RTDE). The Relative travelled distance error

$$\mathrm{RTDE}(\mathbf{P},\mathbf{Q},t_i,\Delta) = \frac{d(\mathbf{P},t_i,\Delta) - d(\mathbf{Q},t_i,\Delta)}{d(\mathbf{Q},t_i,\Delta)} \quad , \tag{4.5}$$

is the difference between the distances the robot travelled, recorded in the estimate and the reference trajectories, relative to the reference travelled distance.

The RTDE works on travelled distances, which are always positive. Consequently, there is no difference between a trajectory composed only of a random movement around a fixed point and an actual trajectory, if their length is similar. This property of the error metric is especially unsuitable for algorithms like the ICP, which return outputs subject to noise. This noise is often modelled as the white noise, with mean at the true position. Therefore, the travelled distance increases even when the robot stays in place. However, the metric differentiates between going straight from one point to another and moving there in a more complicated way. A tangled trajectory is, naturally, longer. This can save some computation time in the evaluation by increasing the value of the $\Delta$ parameter in Algorithm 3.

**Definition 4.2.3** Relative explored distance error (REDE). The Relative explored distance error

$$\mathrm{REDE}(\mathbf{P},\mathbf{Q},t_i,\Delta) = \frac{\|\mathbf{p}_{t_{i+\Delta}} - \mathbf{p}_{t_i}\| - \|\mathbf{q}_{t_{i+\Delta}} - \mathbf{q}_{t_i}\|}{d(\mathbf{Q},t_i,\Delta)} \quad , \tag{4.6}$$

takes the distance between two points at $\Delta$ steps from each other in the estimate and reference trajectories and computes their difference. The difference is, once again, relative to the ground truth travelled distance.

The REDE operates on positions at the beginning and the end of a time window but does not consider what happens in between. The explored distance can be interpreted as the radius of a circle to which boundary we got over the time interval $\Delta$. This approach is better than the RTDE to quantify the error in a trajectory with underestimated or overestimated distance. However, one important assumption needs to be met for this to work. The experiment must be designed such that the error in one direction is much higher than in the other two axes. This is precisely the case in our experiment, where we expect the drift to appear in the direction of the x-axis. The error in the other two axes is relatively small since the ICP algorithm is well constrained by the walls, floor, and ceiling.

Figure 4.9 shows a visual explanation of the three error metrics. Both the RTDE and REDE return a negative value if the distance is underestimated, e.g., when the travelled or the explored distance is shorter than it is supposed to be. The RPDE returns only positive values.

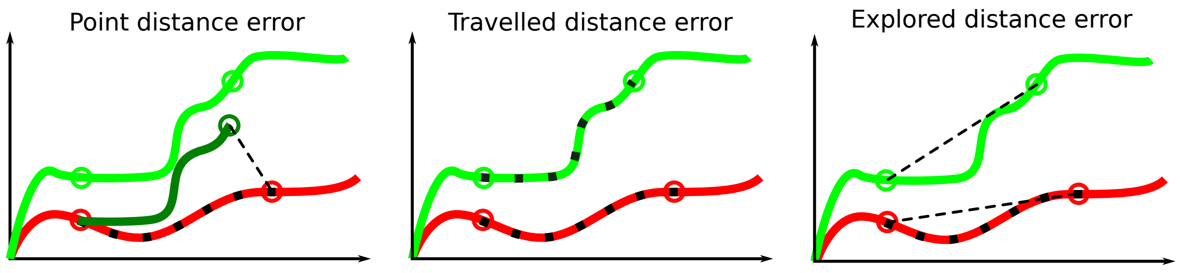


**Figure 4.9:** Visualisation of the presented error metrics. We measure different distances between positions (circles) in the estimate (light green) and the reference (red) trajectories. All errors are relative to the distance travelled by the reference, denoted in black dots over the reference. Left: Relative point distance error, with the actual distance highlighted in a black dashed line. Middle: Relative travelled distance errorF is the difference between the two travelled distances, in black over the trajectories. Right: Relative explored distance error, with the actual distances highlighted in black dashed lines.

With this setup, we processed each of the six sampling methods and the two controls 20 times. The only exception was the Spectral decomposition filter, which turned out to be too time intensive. Therefore, we only report ten executions for this filter. In the next chapter, we will take a look at and discuss the results.

# Chapter 5
# Results and Discussion

This chapter concludes the experiments by presenting both qualitative and quantitative results. In the first part, we explain the difficulties of seeking a fair comparison of different sampling methods. Next, we present the results, employing the three error metrics defined in the previous chapter and discussing their advantages and disadvantages for the comparison of point cloud sampling methods in an underconstrained environment. We report on the localisation performance of the particular methods and offer the final maps of the conducted experiments. Finally, we also present a computation time analysis from the perspective of both the localisation and mapping for each of the methods and examine their real-time capabilities.

The difficulties in direct comparing methods for point cloud subsampling in the localisation task have multiple origins. The first comes from the fact that various sampling methods work with a different number of points. As we saw in the introduction of various filtering techniques in Section 3.3.1 and its summary in Table 3.1, only two methods selected for the final evaluation are controllable, meaning that only for these two methods we can control the number of their output points and the CR. Even though we cannot achieve the exact required CR for the other methods, we could still attempt to reach the closest CR to the desired CR by performing a search in their parameter space. An example of such a parameter is the `Octree Voxel`'s voxel size or the point density per m$^3$ in the case of the `Max Density`. However, such an extensive parameter search is clearly not practical as it is very time intensive. Undoubtedly, a search in the parameter space of a sampling filter to get a particular output is distant from how the filters are commonly used. Furthermore, the search space may not contain the required solution at all, as in the case of `Octree Voxel`, where the space subdivision may not allow some points to be preserved. Therefore, we accept that the filters undergo different CR at different parts of the experiments. We see that it is impossible to ensure an equal number of output points for all methods. On that account, we sampled the parameter space of each filter to cover the CR space evenly from 0 % to 100 %. The parameter space of each filter was sampled 40 times except for the SpDF. Since the SpDF is highly computation time demanding, we limited the parameter space to only 27 samples, ending with the CR of 85 %. For lower CR, the map update duration took more than 150 s, which we set as the limit value. The used parameters and their ranges are presented in Table 5.1.

| Method | Parameter | Range | Count |
|---|---|---|---|
| `Covariance` | compression ratio [%] | [99, 1] | 40 |
| `Max Density` | maximal number of points by m$^3$ | [2, 50 k] | 40 |
| `Global Max Density` | maximal distance between two points [m] | [0.6, 0.0025] | 40 |
| `Octree Voxel` | size of a leaf cell [m] | [0.6, 0.0025] | 40 |
| `Random Informed` | compression ratio [%] | [99, 1] | 40 |
| `SpDF` | radius of uniformity [m] | [1.5, 0.526] | 27 |

**Table 5.1:** Ranges of parameters of the sampling methods used for the evaluation.

Figure 5.1 depicts a comparison of the CRs of the sampling methods as a function of the time since the beginning of an experiment. The reported compression ratio is a median of 20 runs executed for each filter. The filter parameters used here were selected from the values in Table 5.1 with the intention to have the CR with minimal variations between the methods. We immediately see that except for the two controllable methods, `Covariance` and `Random Informed`, the CR is not constant. The `Octree Voxel` shows the least stable performance, with values near 80 % until 35 s and exceeding 90 % near 120 s. The `SpDF` and `Octree Voxel` oscillate around 88 %. Finally, the `Max Density` eventually converges to a stable value, but it differs between the three experiments: *constrained*, *empty* and *10 cones*. Hence, another essential factor affecting the compression ratio is the complexity of the environment. Indeed, we can see that although the `Max Density` filter applied in the *constrained*, *empty* and *10 cones* environment used the same parameter of 700 points per $m^3$, the resulting curves are different. The compression ratio of about 85 % corresponds to a point cloud with 100 000 points with the Velodyne VLP-16 LiDAR and the map update condition set to 1 s.

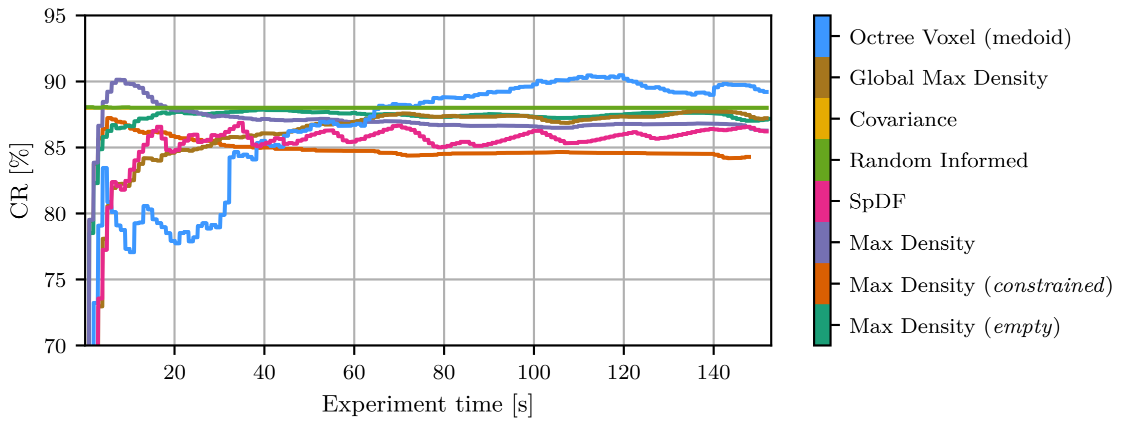


**Figure 5.1:** Compression ratios as a function of time. The lines denote medians of CRs computed over 20 runs with fixed filter parameters and different seeds. The controllable filters, `Covariance` and `Random Informed`, form straight lines. For other filters, the CR is not constant and is a function of the mapping and the localisation quality. The CR also differs based on the constrainess of the environment, as depicted with the three variants of the `Max Density`. The experiment where the data were recorded is the *10 cones*, if not specified in italic.

In the *10 cones* experiment, the environment complexity expresses locally in parts of the trajectory where the robot is close to the obstacles embodied by cones. The localisation error may be smaller if a cone is present in the reading point cloud. At the same time, the number of points in the map after merging such a reading may be different than if the LiDAR's beams reflect from the wall instead. The wall may be already well mapped with enough density, and a density or spatial-based filter would remove any additional information. The localisation error and the CR also influence each other. For example, if the ICP underestimates the robot's position, more points end in a densely populated map area and are, therefore, sampled with a higher rate. On the contrary, if the position is overestimated, more points are kept since they bring new information, and the CR is smaller. Despite these challenges, we believe that we will be able to draw some interesting conclusion from the comparison of the subsampling methods in the following sections.

## 5.1 Localisation error analysis

At first, we will investigate the two trajectories, gathered in the *constrained* and the *empty* experiment with the `Max Density` method. The selection of the sampling method is, in this case, arbitrary. We suppose that all methods should be constrained enough in the *constrained* environment. Similarly, all methods should fail in the *empty*, underconstrained environment.

At the end of the previous chapter, we mentioned that we would use relative values of the travelled distance returned by Equation 4.3 to filter parts of trajectories. The selected relative values range is depicted as the shaded area in Figure C.1. This range is a compromise between error stability and the number of data points. The range values are between 35 % and 45 %, corresponding to a distance travelled by the robot between 50 m and 65 m.

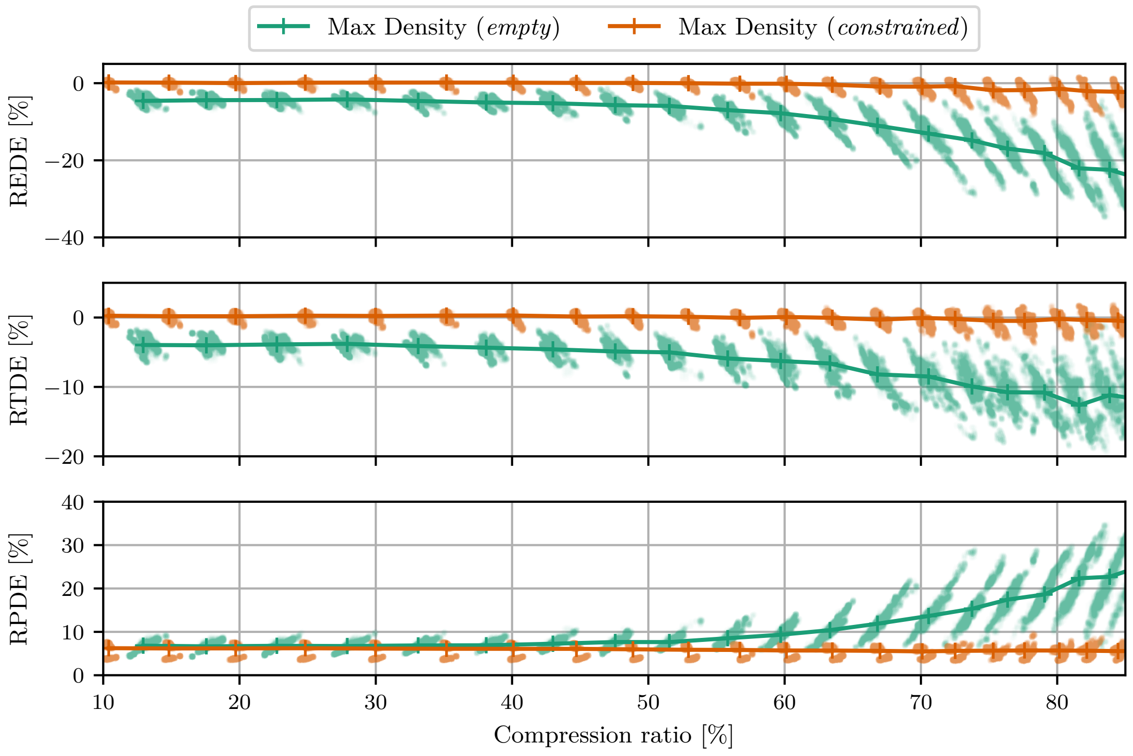


**Figure 5.2:** Errors as a function of CR between 0 % and 85 % for the *empty* and the *constrained* experiment. Lines represent medians of the raw scattered data, with markers at medians of CR. The errors are invariably higher for the *empty* experiment in an unconstrained environment. The errors grow rapidly for CR greater than 70 % for the *empty*, while the *constrained* shows a stable performance regardless the CR.

Figure 5.2 depicts the errors of these two experiments as a function of the Compression ratio. Each cluster of points corresponds to one parameter of the given filter. The x-coordinates of the lines come from the median of CR of the given cluster. The y-coordinates come from the median of errors of the given cluster. We see that for all three error metrics, the error is, as expected, higher for the empty experiment. Moreover, the error is growing with the growing CR. Interestingly, its nature also changes. While for lower CRs, the points form more evenly distributed circular clusters, with growing CR, their spread increases and becomes more elliptical. Locally, the CRs are correlated with the errors, so a higher CR leads to a higher error. For *empty*, the errors are stable up to 50 % and then start growing. For *constrained*, we can notice a slight jump in REDE at around 75 %. The error metrics show similar error values, with the RTDE being the lowest out of the three and the REDE and RPDE being quite similar, with the RPDE resembling the absolute value of the REDE. For the RPDE, it also appears that the data from the *constrained* experiment form two clusters for each parameter. These two clusters eventually connect into one with growing CR, around 80 %.

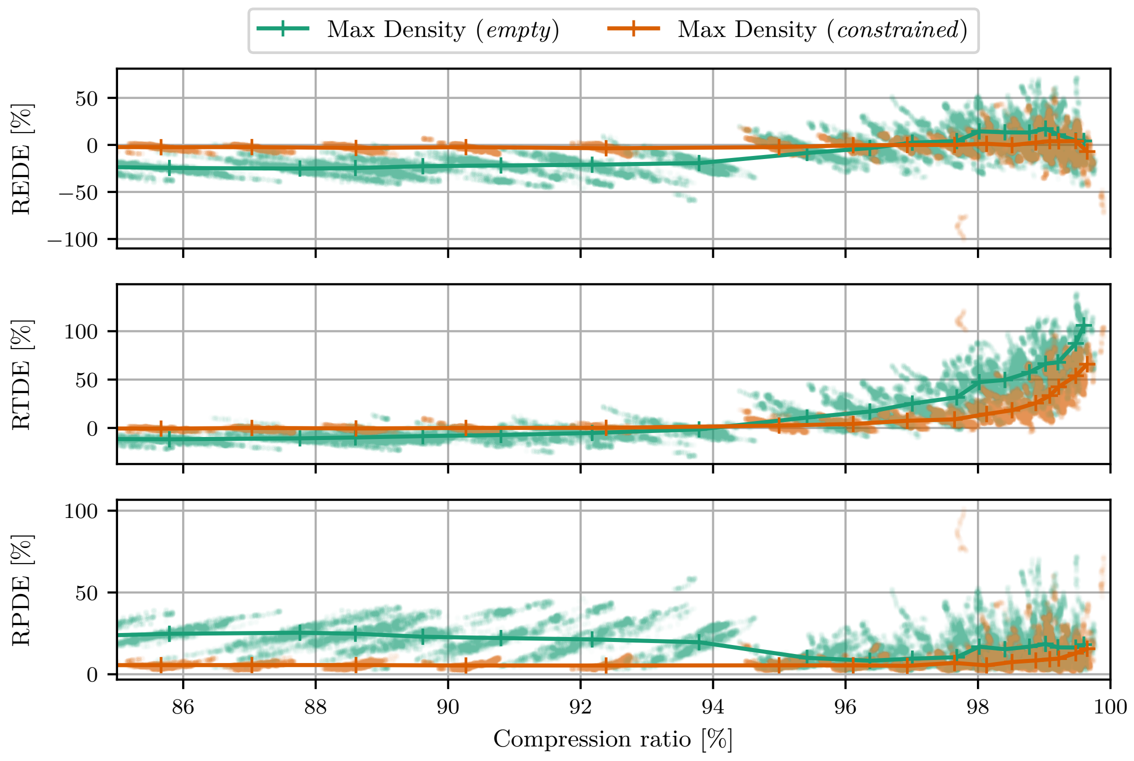


**Figure 5.3:** Errors as a function of CR between 85 % and 100 % for the *empty* and the *constrained* experiment. Lines represent medians of the raw dotted data, with markers at medians of CR. In this CR range, the robot positions are not underestimated anymore. The error values reflect this, being spaced in both the positive and the negative half-plains for the REDE, mostly positive for the RTDE and forming v-shapes for the RPDE.

The growing error changes its nature completely after the CR reaches 85 %, which we demonstrate separately in Figure 5.3. We see that the clusters, initially constituting flat formations, tend to straighten with the CR going to 100 %. Additionally, the magnitude of the error grows rapidly. We believe that these changes have two explanations. Firstly, we saw earlier that the value of the REDE and the RTDE were mostly negative. This corresponded to situations where the robot's position in the environment was underestimated. Possibly, the underestimated positions come from the fact that the robot is moving forward and only in one direction throughout the whole experiment. As the robot moves through the tunnel, the map is being built around it, but most points in the existing map lie behind it. The ICP is, therefore, more attracted to the points behind the robot, the position is underestimated, and the error is biased. With higher CRs, the number of points in the reading point cloud becomes closer to the number of points in the map, at least locally in the close surroundings of the robot. Consecutively, the error becomes unbiased and equally spread around the true position and reassembles the white noise. This random fluctuations around the actual position has different effects on each of our three error metrics. We can see that for REDE, the error at around 96 % becomes centred at zero and distributed evenly on both the negative and the positive half-plane. For the RPDE, the error starts to form v-shaped clusters. These clusters emerge because the error is an absolute value and, therefore, cannot be negative. At the same time, the RTDE becomes strictly positive, as more random movements mean higher travelled distance and, therefore, an overestimated travelled distance of the estimate. The individual clusters also become mixed as their variance in error and the CR increases. Generally, points over CR 85 % have higher error values, with some reaching even 100 %. For our relative distance, the errors of such a magnitude mean that for some iterations, the corresponding distance error reaches 50 m, i.e., over one-third of the total tunnel length. Due to the described characteristics of the errors for higher CRs, we will only discuss the *10 cones* experiment on the interval from 0 % to 85 %. The reader may find to the remaining 15 % in Appendix D.

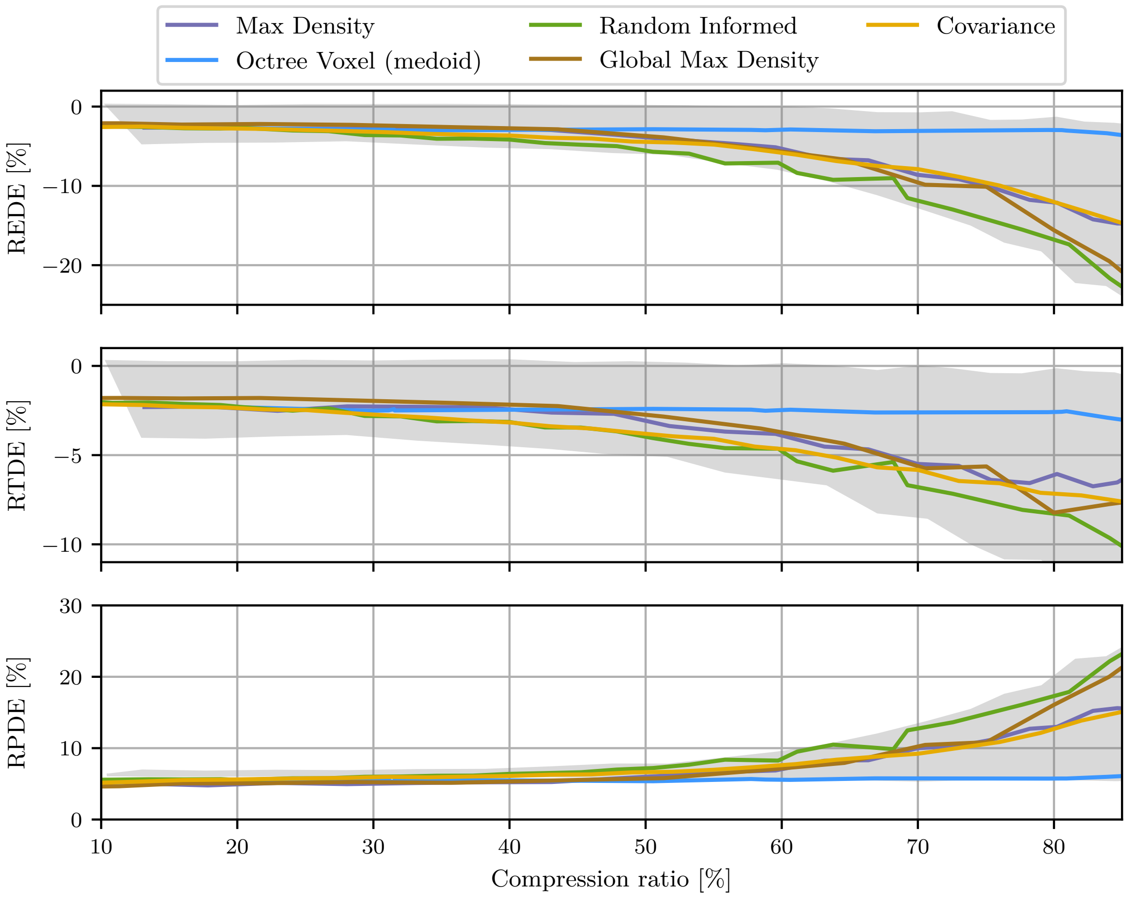


**Figure 5.4:** Errors as a function of the Compression ratio between 0 % and 85 % for different sampling filters. The shaded area represents the space between the medians of baselines, i.e., the *constrained* and the *empty* experiments, both sampled with the `Max Density` filter. The coloured curves represent medians of data from the *10 cones* experiment, processed by individual sampling methods. We see that the curves lie in the shaded area for most of the CR range of the three error metrics. The best performing filter is the `Octree Voxel` with cell sampling set to medoid. While the errors tend to increase around 60 % for the other filters, the `Octree Voxel` shows surprisingly stable performance despite high CR.

Figure 5.4 depicts errors as a function of the CR for other sampling methods applied to the *10 cones* dataset. The shaded area matches the medians of the raw data from the *constrained* and the *empty* experiments, which we described in the previous paragraphs. We can see that for most of the CR range, the coloured lines, representing the medians of the raw CR-error data of each method, lie in the shaded area. After the errors start growing around 60 %, we can already distinguish that the `Octree Voxel` shows a surprisingly stable performance. This is further highlighted for even higher CRs, with the errors rising only moderately after 80 %. Around the same CR, the remaining four methods divide into two clusters. The first one includes the `Covariance` and the `Max Density` filters. The errors for these two filters grow slowly and stay in the shaded area. Contrary, the methods from the second cluster, the `Random Informed` and the `Global Max Density`, go closer to the shaded area from the *constrained* and *empty* experiments. The `SpDF` was not included in this evaluation since in its case, no data were generated for CR lower than 85 %. In Appendix D, we present a more detailed analysis of each of the methods, consisting of figures of the raw data and error evaluation for CRs higher than 85 %.

Figure 5.5 depicts the statistical comparison of the sampling methods for CRs between 50 % and 60 %. The `Octree Voxel` performs clearly the best by a large margin, with the median error at 6.15 %, the lower quartile $Q_1 = 5.83$ %, and the upper quartile $Q_3 =$ 6.46 % of RPDE. This error represents only half a per cent increase over the *constrained* experiment with a median at 5.59 %, the lower and upper quartiles at $Q_1 = 3.87$ % and $Q_3 = 6.62$ %, respectively. The `Octree Voxel` filter is followed by the two density-based filters, `Max Density` and `Global Max Density` with medians close to each other at 6.25 % and 6.64 %. The controllable filters perform the worst, with the error for `Random Informed` reaching 7.56 % for RPDE, corresponding to 3 m of error in the distance over travelled 40 m. A similar comparison for higher CR is in Appendix D.

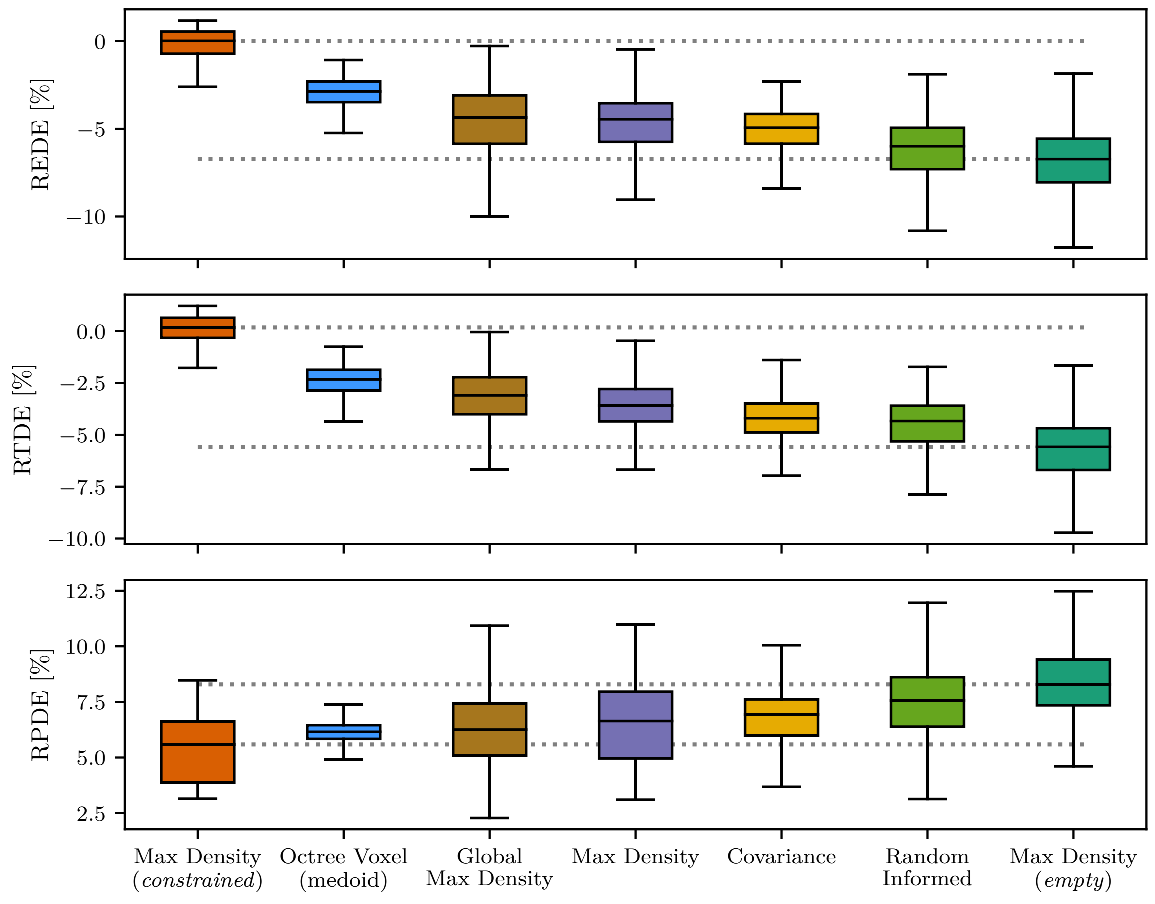


**Figure 5.5:** Statistical analysis of the REDE, RTDE and RPDE for CR between 50 % and 60 %, represented as box plots. Medians, together with the first $Q_1$ and the third quartile $Q_3$, are depicted as coloured boxes. Whiskers extend from the boxes by 1.5 IQR, where IQR is the inter-quartile range IQR $= Q_3 - Q1$. The horizontal dotted lines indicate the area between the medians of data from the *constrained* and the *empty* experiments, sampled with the `Max Density` filter. The spatial method, `Octree Voxel`, distinguishes as the best, with the medians of REDE, RTDE and RPDE at −2.87 %, −2.33 % and 6.15 %, respectively. The density-based `Max Density` and `Global Max Density` form a cluster with results in the second half of the *constrained-empty* range. They are followed by the controllable methods, with the `Covariance` outperforming the `Max Density`.

Figure 5.6 depicts a comparison of the map created with the `Octree Voxel` and the *constrained* and *empty* references. The robot was travelling from left to right. The black dots represent the travelled trajectories. We can see that all three maps bend towards the robot's left side. The bending is the principal source of the error for the *constrained* experiment, with an error of about 7 m in the y and z axes. For the `Octree Voxel`, the errors are a combination of the map bending and the underestimated ICP position. For the *empty* experiment, the error mostly comes from the underestimated robot position, as the map is the most straight one. The origin of the map bending is unclear. ItF can be caused by the LiDAR bias, as explained by Laconte *et al.* [11], or it can also come from a poor odometry estimate. The bending explains higher error values for RPDE compared to the other two metrics in Figure 5.5. The reader is invited to examine the maps produced by other sampling methods in Appendix E.

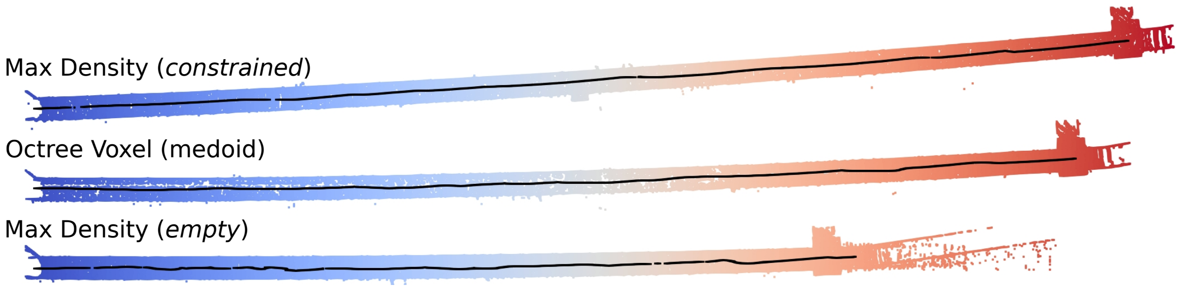


**Figure 5.6:** Comparison of point clouds created by the `Octree Voxel` and the baselines. The colour corresponds to the x-axis distance. All three maps are bent despite the actual tunnel being straight. Top and bottom maps were created with the `Max Density` filter, applied to data from *constrained* and *empty* experiments. The middle point cloud was created with the `Octree Voxel` with medoid as the cell sampling method. The middle map contains 91 692 points.

The results showed that the spatial `Octree Voxel` is superior to the two density-based methods, with the controllable methods not being competitive for higher CRs. An interesting behaviour of the `Octree Voxel` is its stability, which it holds despite a growing CR. We can reliably exclude the explanation that the `Octree Voxel` blindly follows the odometry prior, as in that case we should see the effects of the additional random perturbations on the RTDE. We reason that the spatial division is highly effective in the structured tunnel environment, covering the walls, the floor and the ceiling evenly with points. An interesting analysis would be to compare the different available sampling strategies inside the `Octree Voxel`. Maybe the more computationally expensive, non-image inclusive centroid operation would prove even more performing. On the other hand, the random version may suffice and may add gains in computational speed. Interestingly, even though the random filter proved high-performing and able to compete with more advanced filters in the case of a single reading-to-reference registration [20], our `Random Informed` loses by a significant margin in the continuous localisation and mapping task. Its bad performance may come from the fact that it does not, in any way, prefer points reflected from cones. Instead, most points it keeps come from the planar surfaces of walls, the floor and the ceiling. The density for the `Max Density` filter is computed on a sphere, assuming a different distribution of points than what we observe on the mainly planar surfaces of the tunnel. The `Covariance` is not resistant to unevenly dense point clouds, and we assign this to be the primary source of its error. The `Global Max Density` suffers from its original design and implementation because the distance condition for insertion is only calculated between points in the reading and the map, not between points in the reading itself. The implementation issues could be overcome by adding an insert operation to the employed kd-tree library, allowing us to perform an NNSearch also with points in the reading. This would slow the NNSearch down from the theoretical $O(nlog(m))$. However, since the number of points in the reading $n \ll m$, the effects would be negligible. After adding the insertion, a re-balancing operation could replace the construction of a new kd-tree, which is currently being used. The design problem of the `Global Max Density` comes from the fact that the first point added to the map is always kept, no matter how well it represents points in its neighbourhood. This approach is inferior to the `Octree Voxel`, where each cell was represented by a medoid, or the `Max Density`, where the points are discarded randomly to keep the density uniform.

## 5.2 Computation time analysis

Now that we know how the different filters perform in terms of the localisation, we will briefly discuss their computational time requirements. Five samples with the same seeds between different sampling methods were used for this analysis. The SLAM was conducted on a computer with the Intel® Core$^{TM}$ i7-6850K CPU 3.60GHz with 12 cores and 16 GB of memory, running Ubuntu 18.04. Although the results would differ with a different processing unit, the trend and comparison between methods persist.

Figure 5.7 depicts the duration of the map update phase, i.e., the operation of the filtering and the reading merging to the map as the function of time of the experiment. In order to allow for a similar number of points in the map produced by different sampling methods, the data were generated using the same parameters as in Figure 5.1. The real-time capabilities of our system lie under the 100 ms bound, highlighted in grey. The 100 ms bound comes from the map update rate being set to 1 Hz and the deployed LiDAR producing data at 10 Hz. Once per second, there is a need to do both the ICP-based localisation and to merge the reading into the map and to filter it. For all sampling methods, we see the time rises linearly with a growing number of points in the map. There is one cluster of real-time methods and two isolated lines. The best performing filters in terms of speed are the two controllable filters, with `Random Informed` ahead of `Covariance`. The mapping operation of the `Random Informed` is under 1 ms in the first 10 seconds. As mentioned earlier, we apply a bounding box filter to remove the constraining wall located near the map origin. At the beginning of the mapping, a great part of all points in the reading merged into the map are removed with this bounding box filter. There are not many points left for the kd-tree construction, and the map is not filtered in the case of `Random Informed`, hence the fast execution. The `Random Informed` filter can be regarded as the baseline of how much time it takes to merge points into the map and construct the kd-tree. The third filter, competing with `Covariance` at the beginning and eventually taking longer is `Global Max Density`. This filter does not perform any operation on all points in the reading, only the tiny portion that is added to the map. `Octree Voxel`, the best performing filter in terms of the errors, maintained real-time capabilities throughout the experiment, ending just below the 100 ms bound. This cannot be said about the two density-based filters, with `Max Density` reaching 300 ms per map update and `SpDF` exceeding 10 s, starting at 0.5 s.

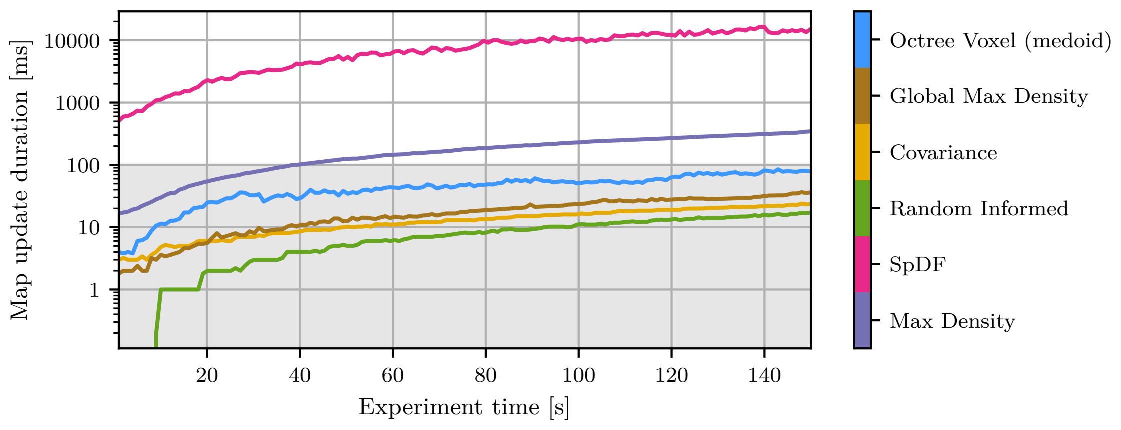


**Figure 5.7:** Map update duration of different sampling methods for CR between 85 % and 90 %. The shaded area indicates the theoretical limit of the real-time capabilities of our system. The lines represent the medians of time needed to merge the reading into the map and filter them. The medians are computed from five runs with different random seeds.

We will now shortly discuss the prospects of further improving the processing speed of the mapping operation, ignoring, for the moment, possible computation hardware improvements. The only increase in real-time capabilities would be at the expense of a lower localisation or mapping rate or higher compression ratio of either the sensor reading or the map. There is not much room to sample the map even further without sacrificing the localisation quality because of the change of error nature that occurs with CR>90 %. Further increasing the CR is risky for the reading, because the overlap between points in the two point clouds can become too small. It is also important to remind ourselves of how was the CR used for Figure 5.7 acquired. The CRs were computed with the formula defined in Section 3.3.1, i.e., only readings at time instants that had satisfied the map update condition had been taken into account in the denominator summation. If we use the number of points produced by the LiDARs instead, the CR will become much higher, with values between 96 % and 98 %. Another influencing factor is the sensor itself, as the computation time becomes, of course, even higher with more advanced sensors. After all, the LiDAR we used only works with 16 beams. With the industry producing increasingly developed products, the need to sample will only rise. We cannot get any speedup from the map offloading either. For the offloading, the map is divided into cells with a cell size of $s = 20$ m. A cell is offloaded if its distance from the estimated robot position is more than $2r + s$ meters, where $r$ is the sensor range. In our case, the map would be offloaded after 240 m, meaning that all methods except for `Covariance` and `Random Informed` would lose their real-time capabilities if the experiment was twice as long. The analysis we conducted is, however, valid only in a simple environment, such as ours, where all points occupy a small area around a straight line travelled by the robot. In a more complex environment, e.g., a forest or a city, the robot would cover with points a planar space instead, and the number of points would grow quadratically. Similarly, for a robot exploring a multi-floor office building, the number of points would grow cubically, to a state where real-time capabilities would be impossible without any compromises on localisation or mapping rate

Finally, we can also examine the time requirements of the localisation part, i.e., the ICP using the initial transformation, the reference and the reading to find an optimal transformation that transforms the reading into the map. Figure 5.8 provides a statistical analysis of the localisation update duration. Theoretically, the time of the localisation duration reflects the quality of the map and the proximity of the initial estimate to the local minimum. Practically though, the values reported in Figure 5.8 are statistics of more than 1500 iterations and are subject to lots of noise. The median of all methods lies under 20 ms, with no whiskers over 100 ms. The `Octree Voxel` is the most stable one, with a minimal localisation duration slightly over 10 ms.

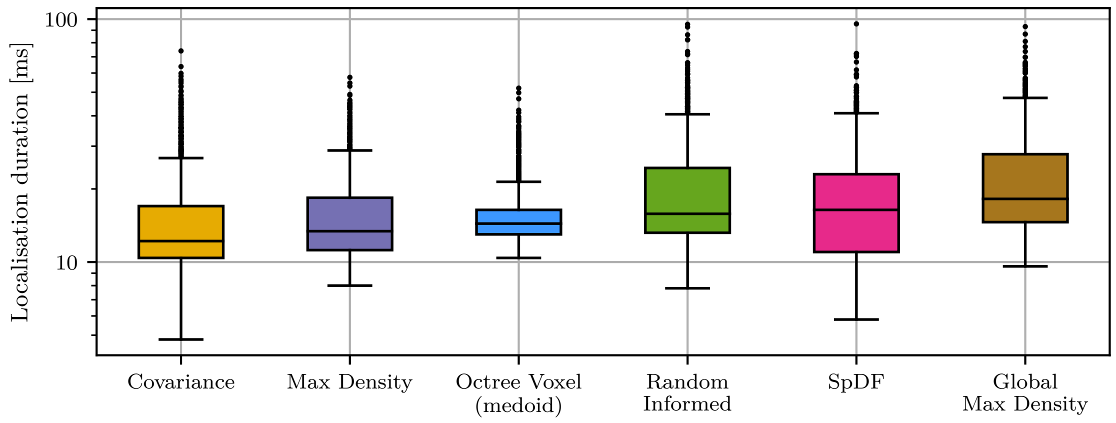


**Figure 5.8:** Statistic analysis of the localisation duration for different sampling methods, represented as box plots. Medians, together with the first $Q_1$ and the third quartile $Q_3$, are depicted as coloured boxes. Whiskers extend from the boxes by 1.5 IQR, where IQR is the inter-quartile range between the first and the third quartiles IQR $= Q_3 - Q1$. The theoretical real-time limit for localisation of our system lies below 100 ms.

# Chapter 6
# Conclusion

This thesis presented an evaluation of point cloud subsampling methods in an unconstrained environment, subject to thin structures. Unlike most subsampling method evaluations present in the literature, we did not focus on assessing a single reading-to-reference point cloud registration. Instead, we analysed the methods in the context of 3D LiDAR-based Simultaneous localisation and mapping. The motivations originated not only from the need to reduce the number of points in today's point clouds to achieve better memory and computation efficiency but primarily to investigate the effects of downsampling methods in a human-made environment with sparse and tiny spatial features.

In this work, we first established the underlining theory of point cloud subsampling in the context of 3D localisation and mapping. Then, we introduced several filters, explained their properties and behaviour and provided a taxonomy based on the proposed theory. To better understand the limitations of the subsampling filters, we explored the representation of thin structures in 3D point clouds using three LiDAR sensors and a surveying total station. The findings were later used to design experiments, which evaluated the quality of environment representation with the different methods. A data processing pipeline to synchronise and evaluate trajectories from three sources, the ground truth, the odometry and the estimate, was outlined. At its final stage, this pipeline contained a localisation evaluation, for which we proposed three error metrics, two of which do not require any spatial alignments of the trajectories. Finally, we used these error metrics to evaluate and compare the results of SLAM performed with six distinct subsampling strategies. The results also included a computation time analysis of both the mapping and the localisation components. The discussion concluded with the only representative of spatial methods, the `Octree Voxel`, overcoming the other methods in all three error metrics while retaining real-time capabilities in the experimental setup.

Several important lessons were learned during the course of this thesis.

- We found that contemporary LiDARs cannot reliably represent objects with a diameter of less than 10 cm without an extensive calibration. Furthermore, some sensors may contain advanced software, confusing thin structures with noise and filtering it from the scan. Anyone interested in the 3D representation of thin structures should bear these limitations in mind and reserve adequate time for addressing them.

- Until this work, the employed surveying total stations system was only tested on the exterior. Compared to experiments in the open air, the total stations' configuration needed to adapt to different conditions of a narrow environment of the tunnels to deliver the same accuracy of the ground truth trajectory.

- The experiment execution also required a different approach than outside. A person following and operating the robot could block the sight of view between the total stations and the robot or supplement additional constraints to the map point cloud. Remote control with a good range and a camera feeding visual data of the robot's surroundings to the operator was selected as a preferable option for the robot operation.

- Identifying a fair subsampling evaluation metric in the context of SLAM had been complicated, and a new protocol had to be proposed. In a single reading-to-reference scenario, a sole distance value sufficiently describes the quality of the resulting transformation. On the other hand, the proposed evaluation protocol capitalised on sampling each filter's parameter space to ensure even coverage of the Compression ratio space. However, the forfeit for this approach was the correlation between the error and the CR.

In the future, the evaluation could be extended to more data. In fact, the presented three datasets of about 150 m each represent only a part of the conducted experiments and recorded data in the underconstrained environment. Firstly, together with the one-way robot journeys reported in this work, we also recorded the return journeys, doubling the total distance the robot travelled. We also gathered 1400 m of further data in the tunnels, which were not used in this work for time reasons. It would also be interesting to evaluate the sampling methods in a standard constrained environment and compare the results to verify that, theoretically, the methods should perform similarly in such a scenario. The reported results show that even though the `Octree Voxel` outperforms other strategies, there still exists an interesting area for improvement. With a different local sampling in octree cells, we could achieve a faster execution or even smaller localisation errors. Moreover, we noticed that even though the final maps often contain correctly reconstructed cones, they did not suffice for the ICP to converge to the correct solution. The author of this work is ambitious to capitalise on the findings presented hereF and submit the results, covering the extended dataset, to a scientific conference. Another future direction could thus concentrate on sampling both the map and the reading and analysing the ICP error function. Finally, we saw that the map of the tunnel bends, no matter the environment's complexity. While the reasons and explanations are still not fully revealed, tunnel bending represents a considerable difficulty for deploying robots with an ICP-based SLAM in any tunnel-like environment.

# MASTER‘S THESIS ASSIGNMENT

## I. Personal and study details

| | | | |
|---|---|---|---|
| Student's name: | **Boxan Matěj** | Personal ID number: | **469915** |
| Faculty / Institute: | **Faculty of Electrical Engineering** | | |
| Department / Institute: | **Department of Cybernetics** | | |
| Study program: | **Cybernetics and Robotics** | | |
| Branch of study: | **Cybernetics and Robotics** | | |

## II. Master’s thesis details

Master’s thesis title in English:

**3D Point Clouds Reconstruction of Environment Subject to Thin Structures**

Master’s thesis title in Czech:

**Rekonstrukce trojrozměrných mraků bodů v prostředí s tenkými objekty**

Guidelines:

• Get familiar with lidar sensors (noise and limitation), 3D registration pipeline using libpointmatcher (open source library), and with implemented algorithms for point cloud manipulation;
• Investigate the impact of thin structures in current point cloud registration algorithms relying on point cloud subsampling methods to accelerate the computation;
• Propose solutions for limitations observed;
• Develop an experimental setup (software) allowing the quantification of localization error to evaluate the quality of the proposed solution;
• Write the results in the format of a conference paper.

Bibliography / sources:

[1] Mathieu Labussière, Johann Laconte, and François Pomerleau. “Geometry Preserving Sampling Method Based on Spectral Decomposition for Large-Scale Environments”. In: Frontiers in Robotics and AI7(Sept. 2020). ISSN: 22969144. doi: 10.3389/frobt.2020.572054.
[2] Charles R. Qi et al. “PointNet: Deep learning on point sets for 3D classification and segmentation”. In:vol. 2017-January. Institute of Electrical and Electronics Engineers Inc., Nov. 2017, pp. 77–85.isbn:9781538604571.doi:10.1109/CVPR.2017.16.url:https://openaccess.thecvf.com/content_cvpr_2017/papers/Qi_PointNet_Deep_Learning_CVPR_2017_paper.pdf
[3] K.H. Lee, H. Woo, and T. Suk. “Data Reduction Methods for Reverse Engineering”. In:The InternationalJournal of Advanced Manufacturing Technology17 (10 May 2001), pp. 735–743.issn: 0268-3768.doi:10.1007/s001700170119.url:http://link.springer.com/10.1007/s001700170119

Name and workplace of master’s thesis supervisor:

**Att. Prof. François Pomerleau   Université Laval, Québec, Canada**

Name and workplace of second master’s thesis supervisor or consultant:

**Ing. Vladimír Smutný, Ph.D.   Robotic Perception  CIIRC**

Date of master’s thesis assignment: **20.01.2022**   Deadline for master's thesis submission: **15.08.2022**

Assignment valid until: **30.09.2023**

| | | |
|---|---|---|
| Att. Prof. François Pomerleau | prof. Ing. Tomáš Svoboda, Ph.D. | prof. Mgr. Petr Páta, Ph.D. |
| Supervisor’s signature | Head of department’s signature | Dean’s signature |

## III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

| | |
|---|---|
| . | |
| Date of assignment receipt | Student’s signature |

# Appendix B

# Map update delay condition analysis

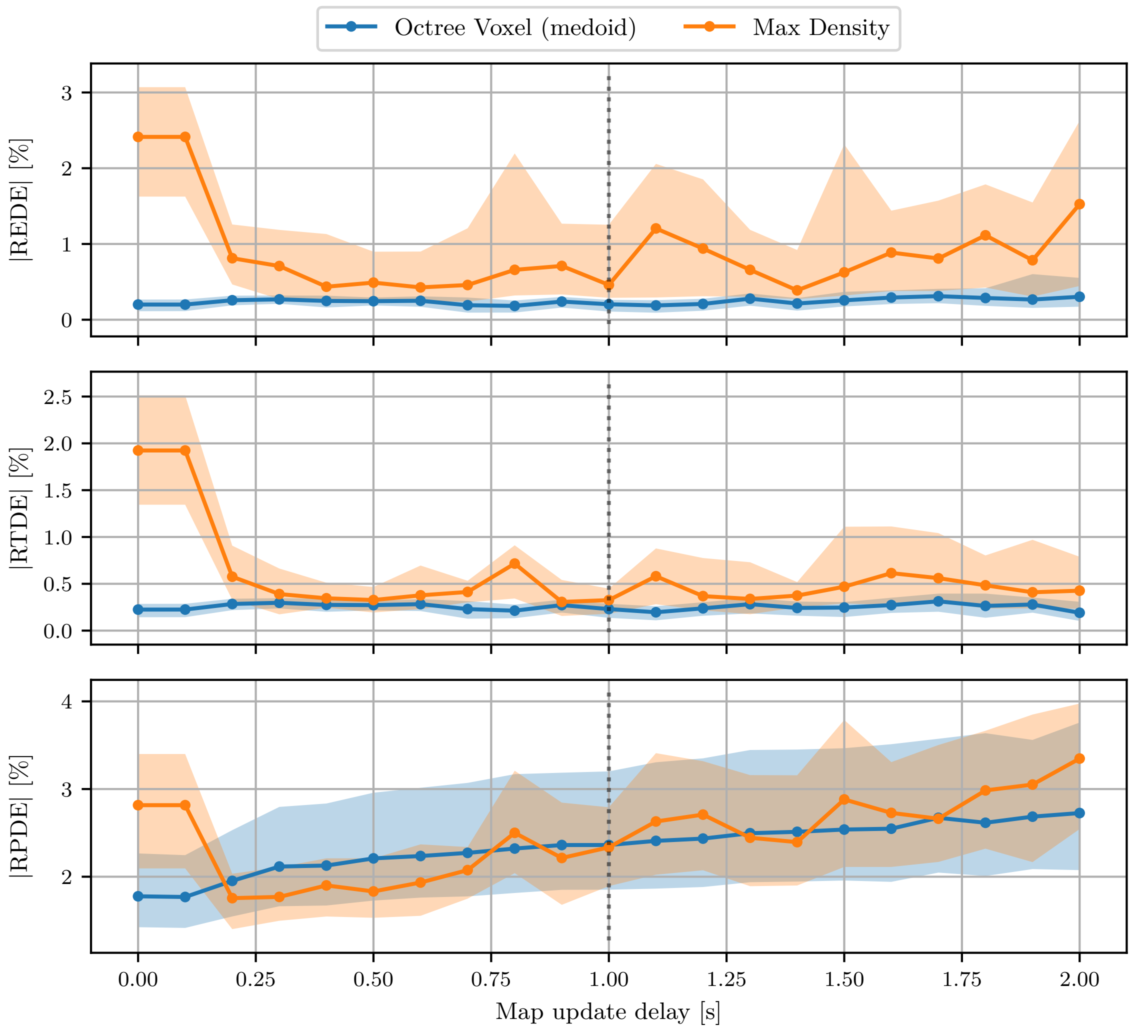


**Figure B.1:** Analysis of the map update condition shows the errors as a function of the map update delay. The x-axis value determines the delay between two consecutive map updates. The delay is evaluated on absolute values of three error metrics and two sampling filters, the `Max Density` and the `Octree Voxel`, with a medoid used for sampling inside individual voxels. The lines denote medians, and the shaded areas illustrate the first and third quartiles $Q_1$ and $Q_3$. The *constrained* experiment was used for this evaluation. After an initial decrease of `Max Density`, the errors fluctuate between 0.0 % and 1.5 % for the REDE and RTDE while slowly rising for the RPDE. The `Octree Voxel` shows a far more stable course, with the REDE and RTDE under 0.5 % and slowly rising for the RPDE. The median and quartile statistics were calculated over ten mapping runs. The final map update frequency (in dotted grey), set to 1 s, is a compromise between mapping and localisation quality and processing time.

# Appendix C

# Range of the relative travelled distances analysis

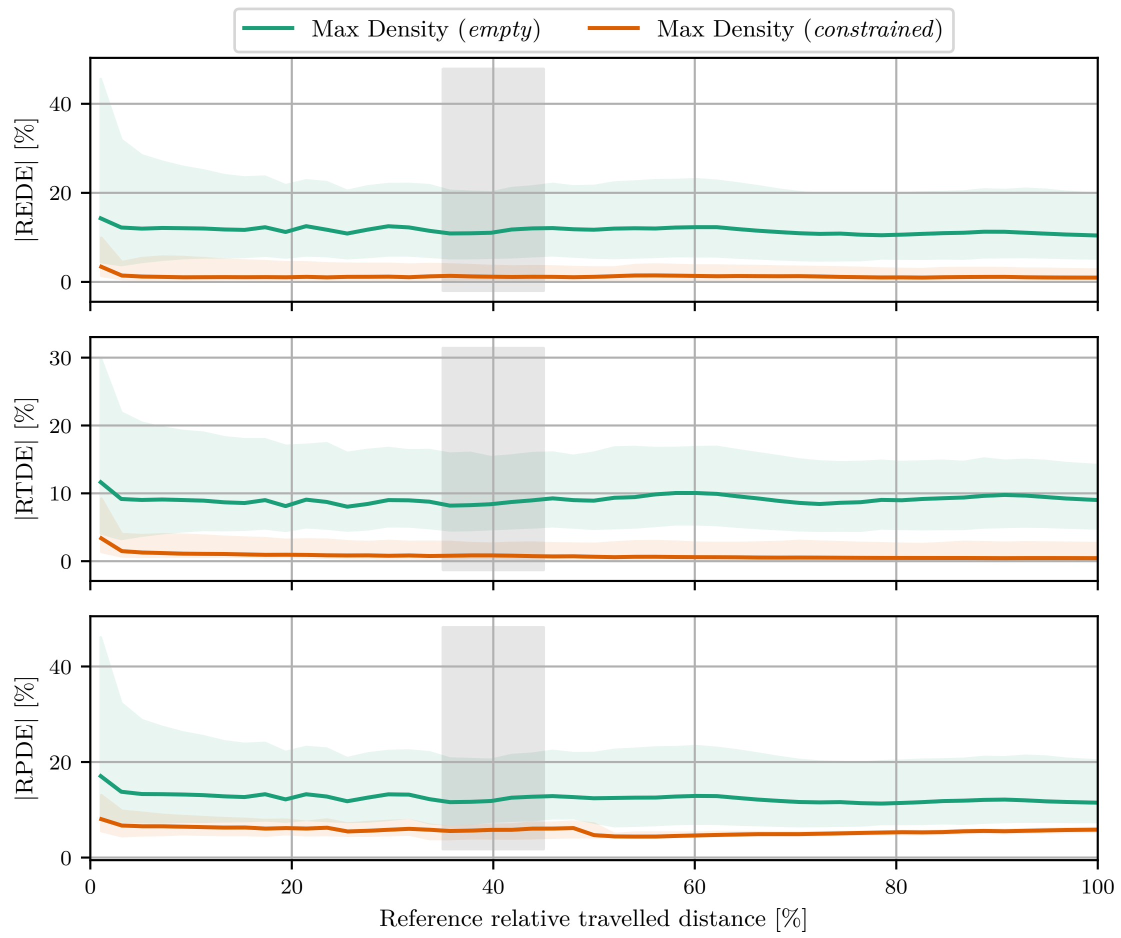


**Figure C.1:** Analysis of the relative travelled distance used for evaluation shows the errors as a function of the relative values returned from Equation 4.3. The x-axis value determines the travelled distance of the reference, ground truth trajectory, relative to its total travelled distance. The distances matching these relative values range from millimetres to the total tunnel length of approximately 150 m. The analysis is given as the absolute values of three error metrics for the *constrained* and *empty* experimental environments. For both experiments, the `Max Density` sampling filter was used. The lines denote medians and the shaded areas indicate the first and third quartiles $Q_1$ and $Q_3$. The statistics were calculated over 20 mapping runs for 40 `Max Density` parameters, covering the CR space evenly from 10 % to 100 %. We can clearly see that the errors stabilise after an initial drop in error values. The starting higher error values are caused by the situations where the robot did not move or moved only very slowly. The ICP error was more significant than the reference travelled distance, hence the high error values. The shaded area indicates the chosen values between 35.0 % and 45.0 %. The chosen values are a trade-off between the error value stability and data points quantity.

# Appendix D
# Quantitative results

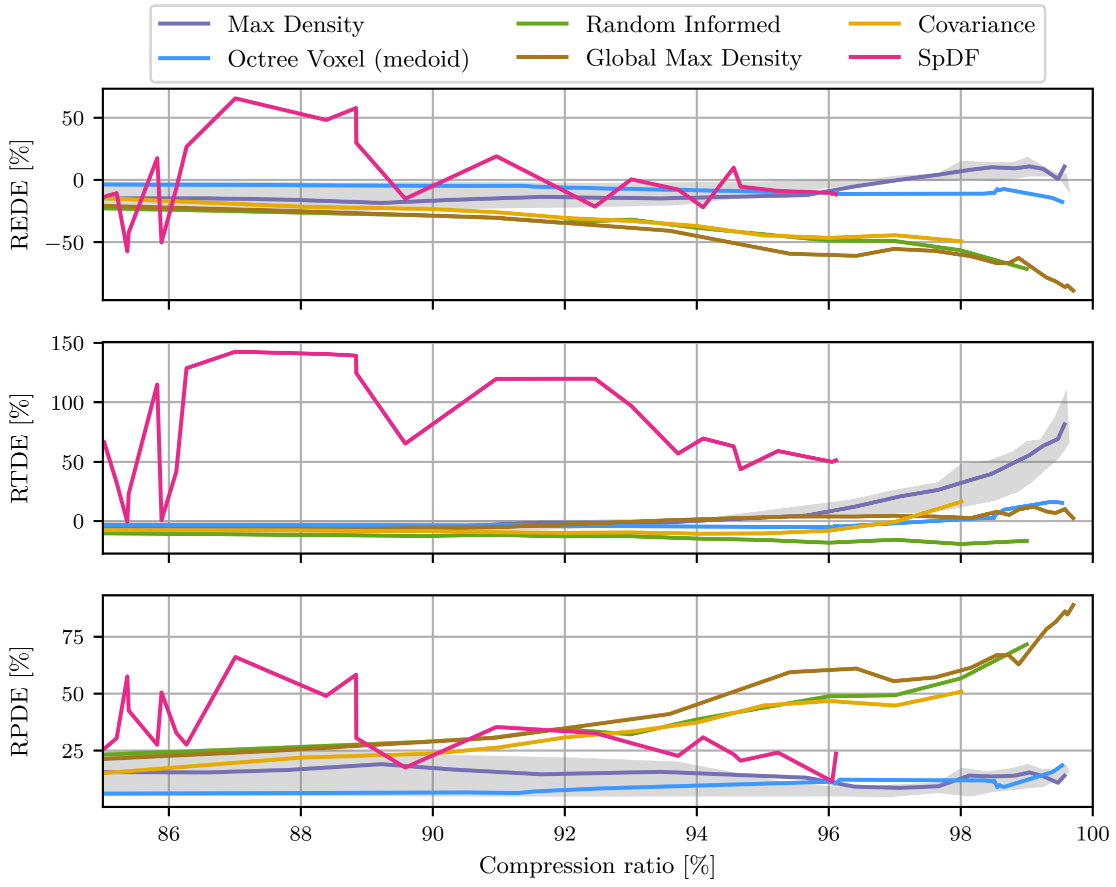


**Figure D.1:** Errors as a function of the Compression ratio between 85 % and 100 % for different sampling filters. The shaded area represents the space between the medians of baselines, i.e., the *constrained* and the *empty* experiments sampled with the `Max Density` filter. The coloured curves represent medians of data from the *10 cones* experiment, processed by individual sampling methods. For the REDE and RPDE, we can notice that three sampling methods, the `Random Informed`, `Covariance` and `Global Max Density` form a cluster. These methods cause an underestimation of the robot's position in the tunnel, provoking a negative REDE. The RPDE's magnitude is greater because it also reflects the tunnel bending. The `Max Density` copies the baseline shaded areas, proving that the method works steadily no matter the environment's constrainess. The `Octree Voxel` is the best performing method even for a higher CR range, with a sudden increase only after 98 %. On the contrary, the `SpDF` is the least stable sampling method, with the error decreasing with a growing CR. The missing data for CR > 96 % for `SpDF` is caused by the `libpointmatcher` returning a convergence error caused by insufficient overlap between the map and the reading.

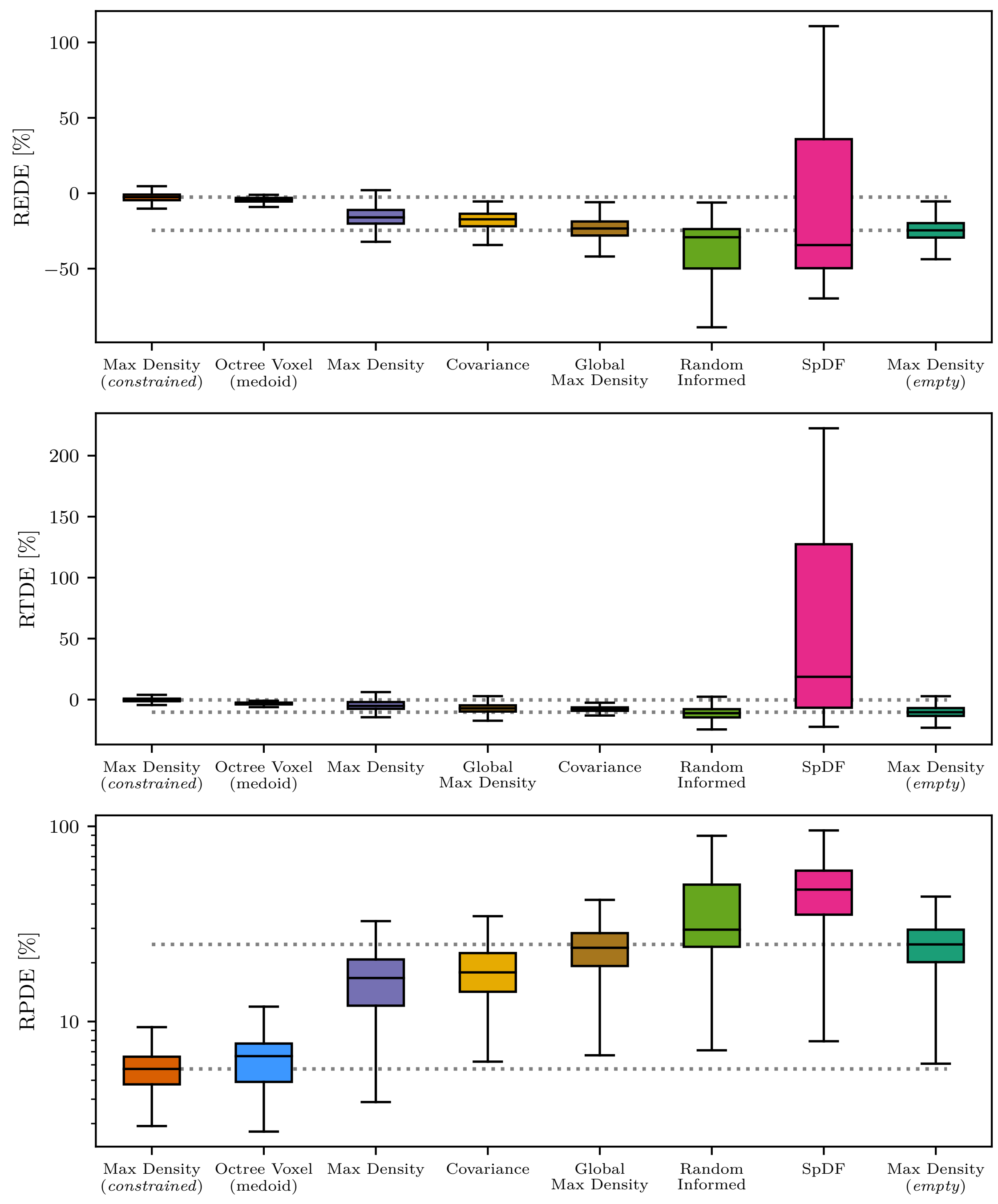


**Figure D.2:** Statistical analysis of the REDE, RTDE and RPDE for CR between 85 % and 90 %, represented as box plots. Medians, together with the first $Q_1$ and the third quartile $Q_3$, are depicted as coloured boxes. Whiskers extend from the boxes by 1.5 IQR, where IQR is the inter-quartile range $\mathrm{IQR} = Q_3 - Q1$. The horizontal dotted lines indicate the area between the medians of data from the *constrained* and the *empty* experiments, sampled with the `Max Density` filter. The `Random Informed` is no longer in the zone defined by the baselines in this CR range. Together with `SpDF`, the quartiles for the REDE and RPDE exceed this zone by more than 30 %. The equally positive and negative values of REDE for `SpDF` show that the robot's position was both under and overestimated. The `Octree Voxel` shows the best performance, with the median of the RPDE less than 2.0 % from the *constrained* baseline.

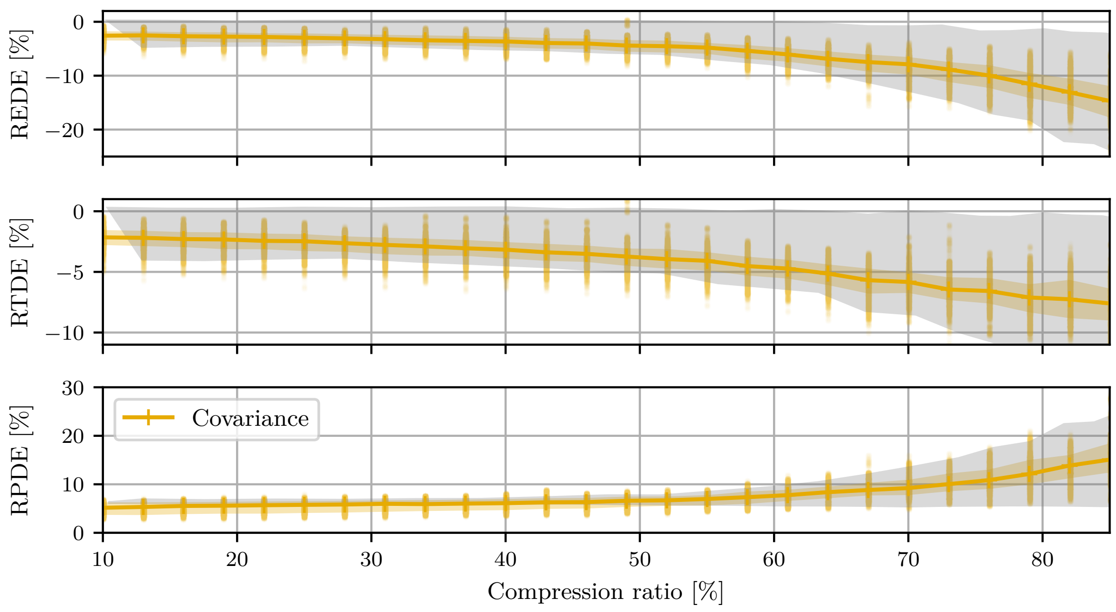


**Figure D.3:** Raw errors data as a function of the Compression ratio between 0 % and 85 % for `Covariance`. As a controllable method, the `Covariance` keeps all the scattered raw data values on vertical lines defined by the CR. The yellow lines represent the raw error data medians. The errors start growing after 50 % and until 85 % are in the shaded area representing the space between the medians of baselines, i.e., the *constrained* and the *empty* experiments sampled with the `Max Density` filter.

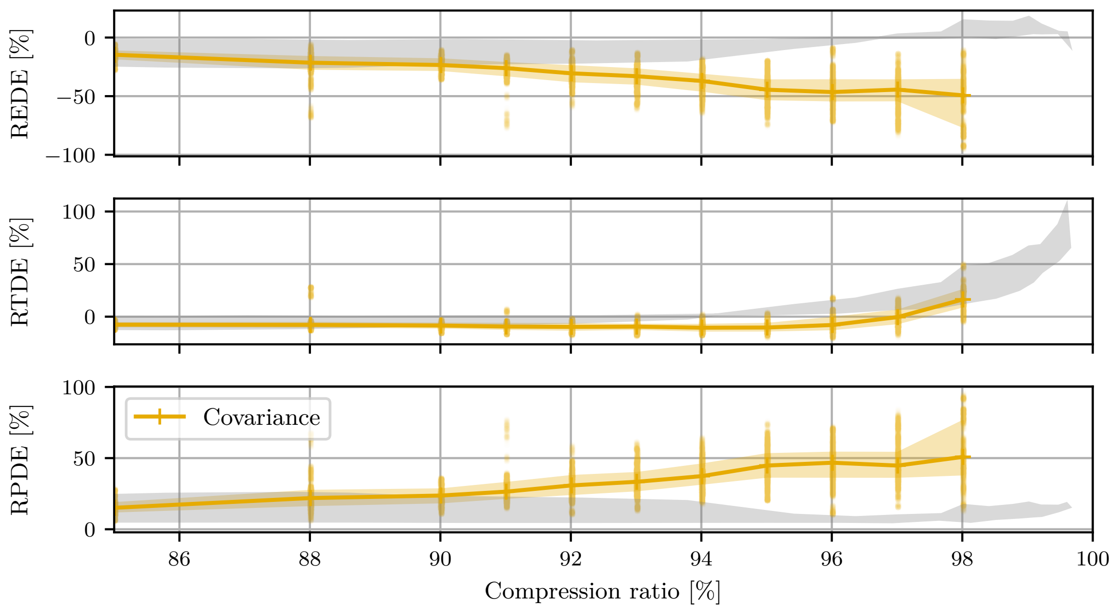


**Figure D.4:** Raw errors data as a function of the Compression ratio between 85 % and 100 % for `Covariance`. As a controllable method, the `Covariance` keeps all the scattered raw data values on vertical lines defined by the CR. The yellow lines represent the raw error data medians. The errors start to quit the shaded area representing the space between the medians of baselines, i.e., the *constrained* and the *empty* experiments sampled with the `Max Density` filter, at about 90 %. The missing data for CR = 99 % is caused by the `libpointmatcher` returning a convergence error caused by insufficient overlap between the map and the reading.

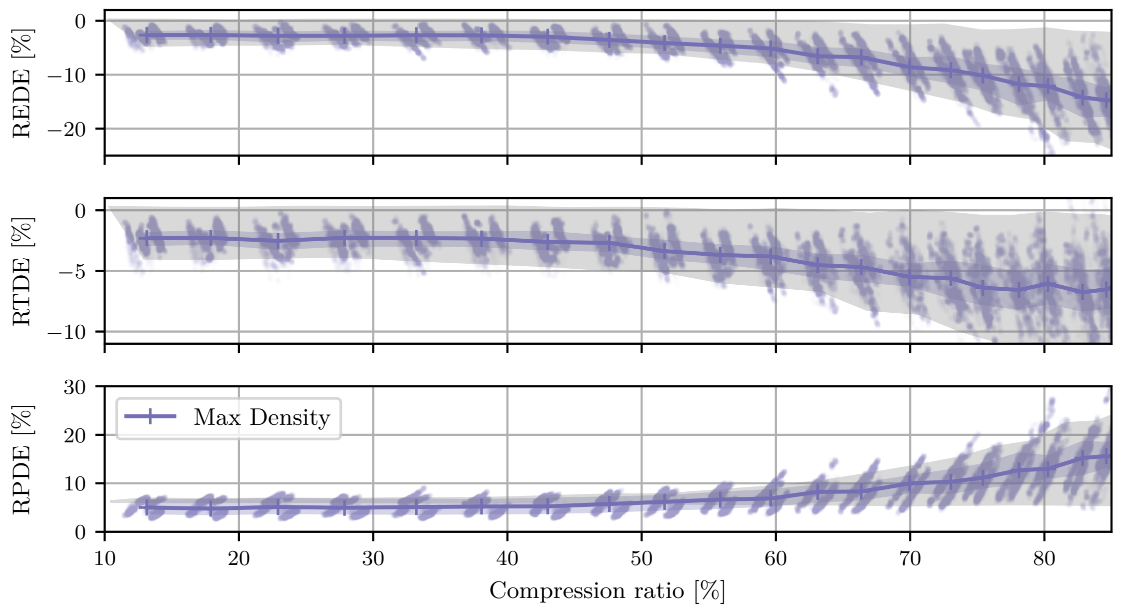


**Figure D.5:** Raw errors data as a function of the Compression ratio between 0 % and 85 % for `Max Density`. The purple lines represent the raw error data medians, with the markers indicating the median CR for a single filter parameter value. Interestingly, we can notice that the scattered raw data form three clusters for each parameter value, around a fixed CR. Our explanation of this phenomenon is that the errors and the CR are correlated and are a function of the robot's observation of the environment. The errors start slowly growing after 50 % and until 85 % are in the shaded area representing the space between the medians of baselines, i.e., the *constrained* and the *empty* experiments sampled with the same filter.

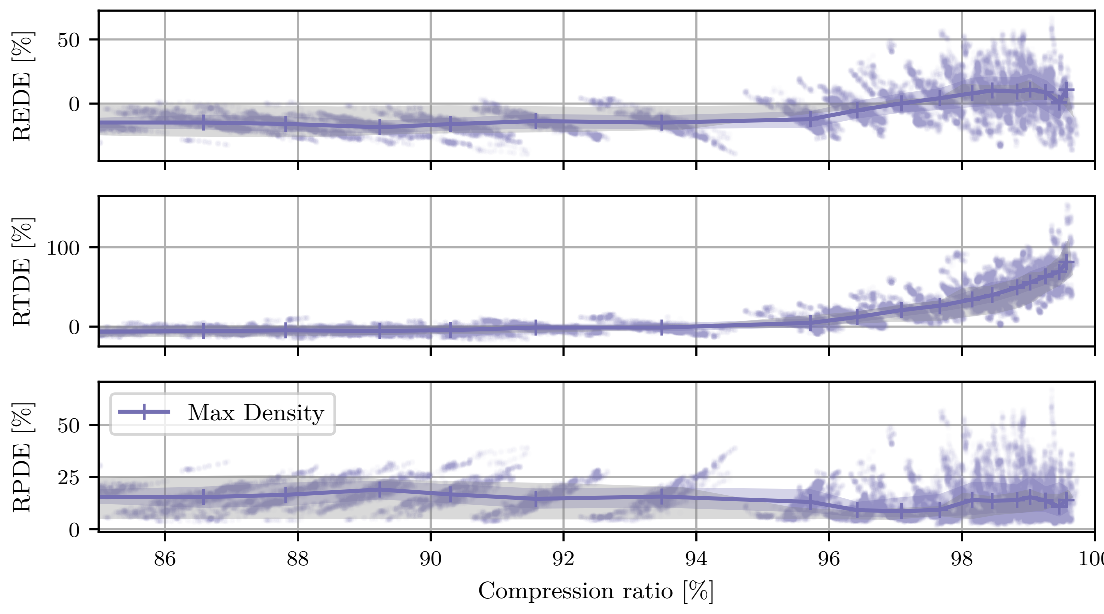


**Figure D.6:** Raw errors data as a function of the Compression ratio between 85 % and 100 % for `Max Density`. The purple lines represent the raw error data medians, with the markers indicating the median CR for a single filter parameter value. The errors' magnitude increases as well as the variance in the x-axis. The values are, nevertheless, still in the shaded area representing the space between the medians of baselines, i.e., the *constrained* and the *empty* experiments sampled with the same filter.

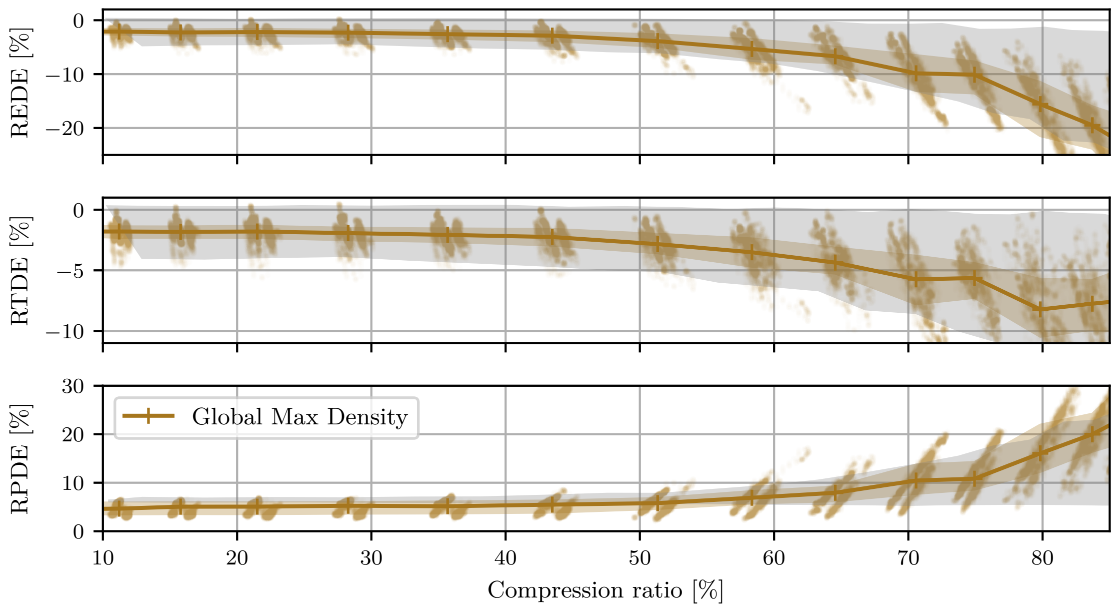


**Figure D.7:** Raw errors data as a function of the Compression ratio between 0 % and 85 % for `Global Max Density`. The brown lines represent the raw error data medians, with the markers indicating the median CR for a single filter parameter value. As for the `Max Density`, we can notice that the scattered raw data form three clusters for each parameter value, around a fixed CR. The errors start growing after 50 % and until 85 % are in the shaded area representing the space between the medians of baselines, i.e., the *constrained* and the *empty* experiments sampled with the `Max Density` filter.

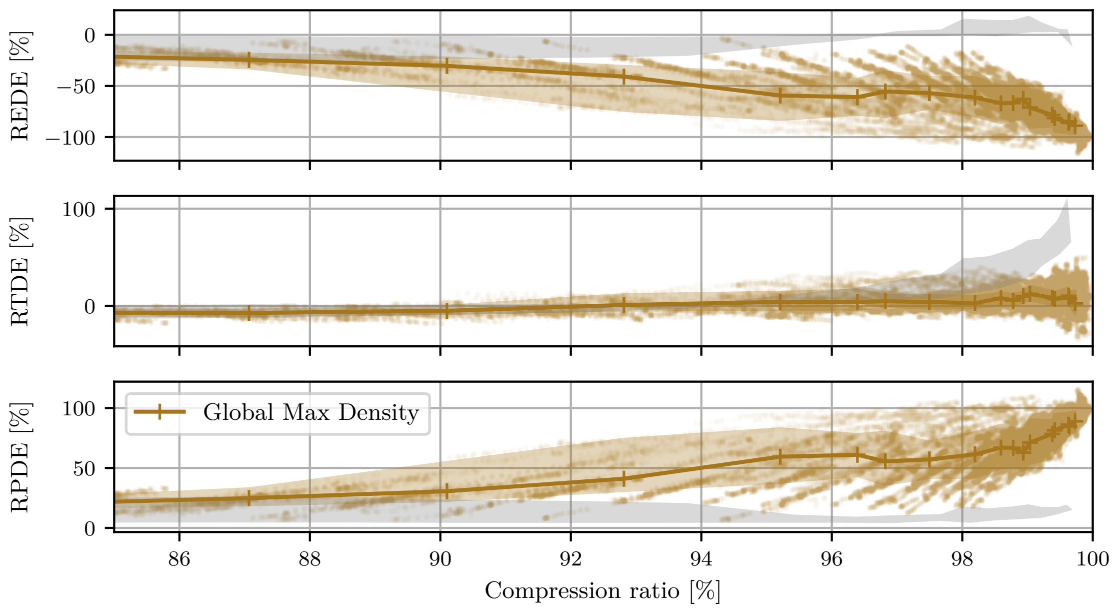


**Figure D.8:** Raw errors data as a function of the Compression ratio between 85 % and 100 % for `Global Max Density`. The brown lines represent the raw error data medians, with the markers indicating the median CR for a single filter parameter value. The REDE and RPDE errors reach 100 % for CR approaching 100 %.

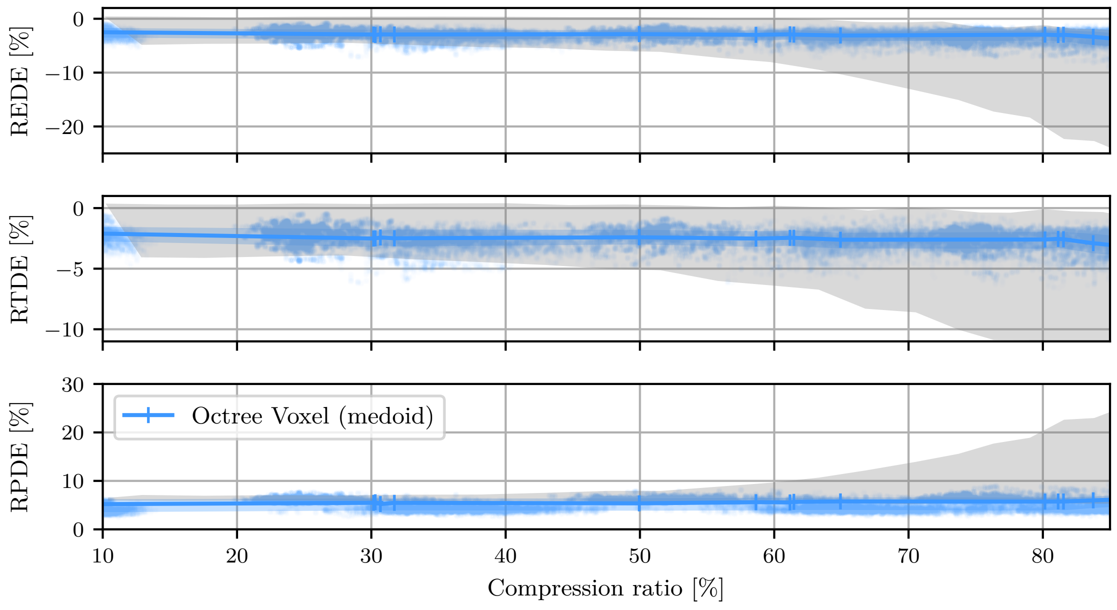


**Figure D.9:** Raw errors data as a function of the Compression ratio between 0 % and 85 % for `Octree Voxel`. The blue lines represent the raw error data medians, with the markers indicating the median CR for a single filter parameter value. We can notice that the raw data have the largest x-axis variance of all evaluated sampling methods. The clusters of the scattered raw data overlap between various parameter values. The missing data between 15 % and 20 % indicate a non-existing spatial subdivision that would produce CR in this range. The errors' values are stable and are located in the shaded area representing the space between the medians of baselines, i.e., the *constrained* and the *empty* experiments sampled with the `Max Density` filter.

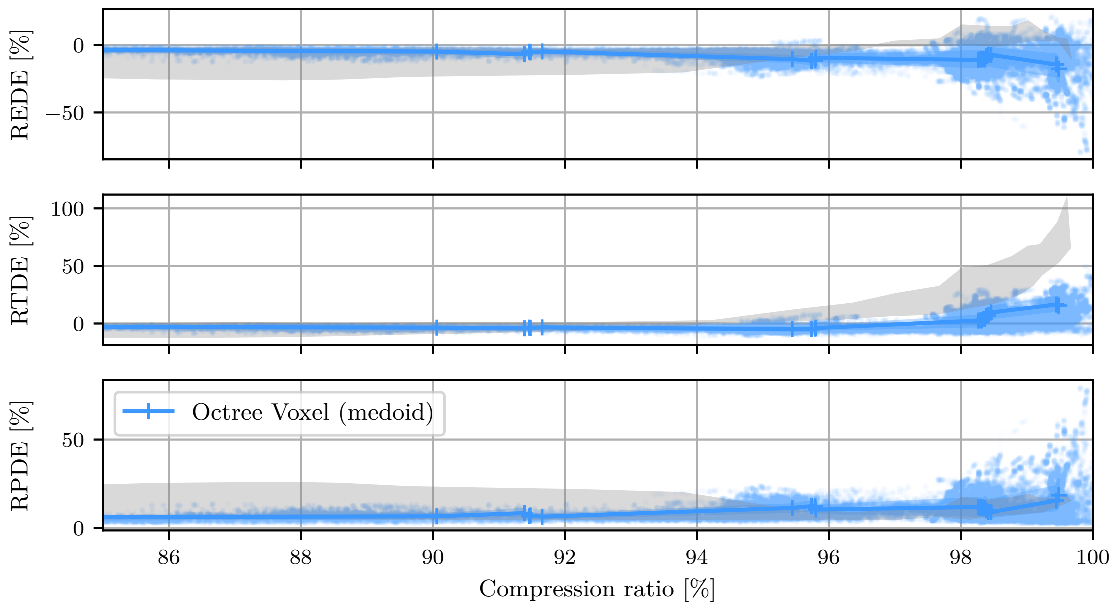


**Figure D.10:** Raw errors data as a function of the Compression ratio between 85 % and 100 % for `Octree Voxel`. The blue lines represent the raw error data medians, with the markers indicating the median CR for a single filter parameter value. The clusters of the scattered raw data overlap between various parameter values. The errors' values are stable and are located in the shaded area representing the space between the medians of baselines.

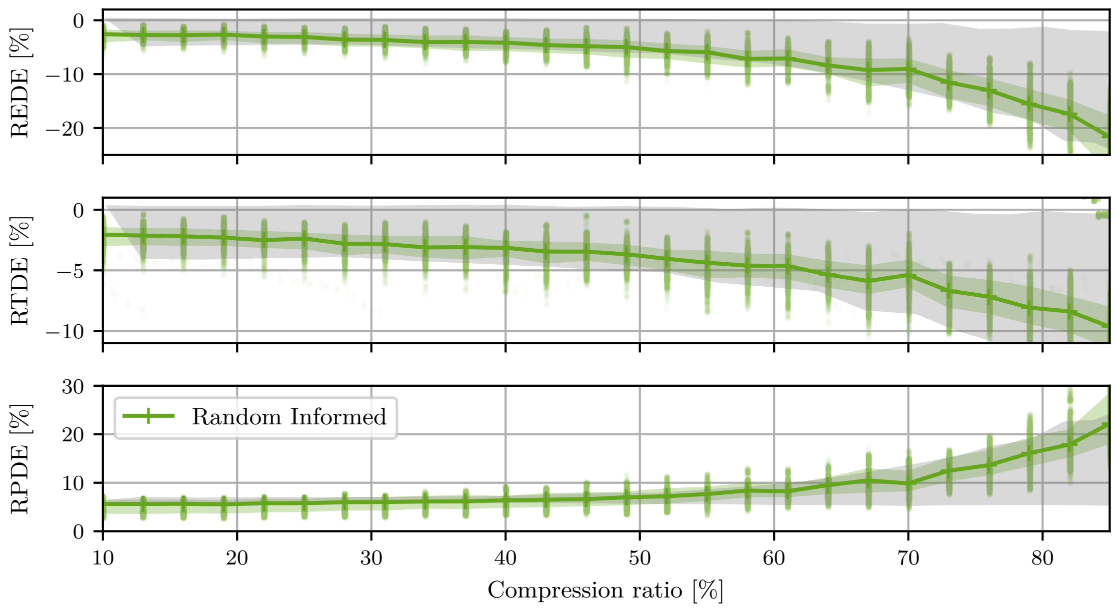


**Figure D.11:** Raw errors data as a function of the Compression ratio between 0 % and 85 % for `Random Informed`. The green lines represent the raw error data medians, with the markers indicating the median CR for a single filter parameter value. Similarly to `Covariance`, the `Random Informed` is also a controllable method, so it keeps all the scattered raw data values on vertical lines defined by the CR. The errors' values start growing after 50 % and are located in the shaded area representing the space between the medians of baselines, i.e., the *constrained* and the *empty* experiments sampled with the `Max Density` filter.

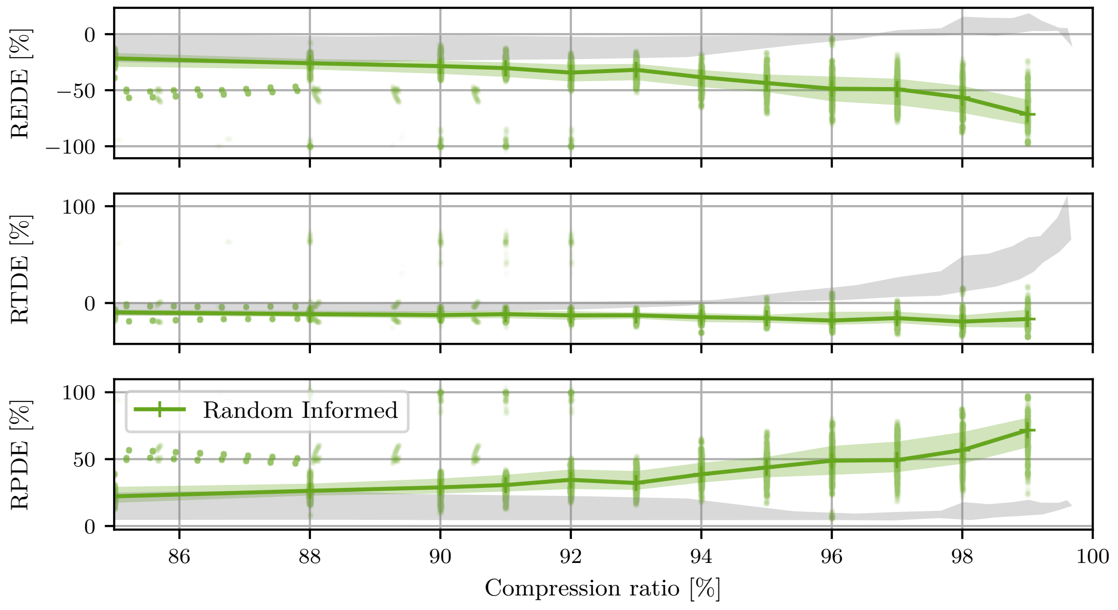


**Figure D.12:** Raw errors data as a function of the Compression ratio between 85 % and 100 % for `Random Informed`. The green lines represent the raw error data medians, with the markers indicating the median CR for a single filter parameter value. The errors quit the shaded area representing the space between the medians of baselines, e.g., the constrained and the empty experiments sampled with the `Max Density` filter, at about 89 %.

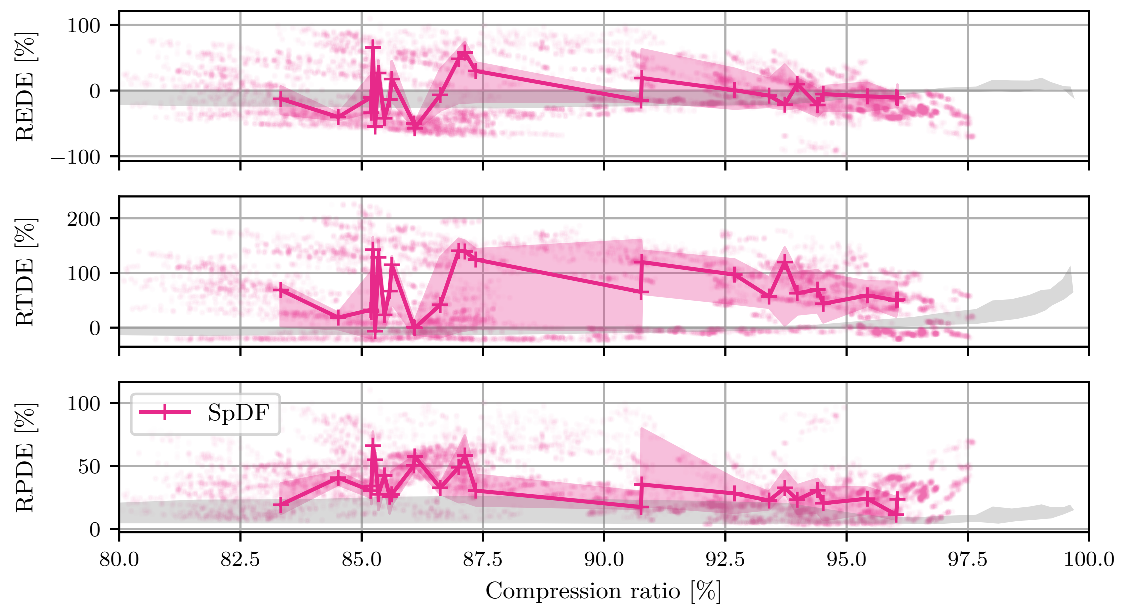


**Figure D.13:** Raw errors data as a function of the Compression ratio for `SpDF`. The pink lines represent the raw error data medians, with the markers indicating the median CR for a single filter parameter value. Although some raw data lie in the shaded area representing the space between the medians of baselines, the outlier scatter values force the medians out of this area. The `SpDF` is the least stable of all examined sampling methods, with errors differing significantly between iterations with a fixed parameter and a different seed. The method takes a single parameter to regulate densities of all three geometric primitives, i.e., surfaces, curves and junctions. We also tested a modified parameter taking a different density for a specific geometric primitive, but without observing any improvements.

# Appendix E
# Qualitative results

In this chapter, we offer a comparison of final maps of the tunnel created with the `Covariance`, `Max Density`, `Global Max Density`, `Random Informed` and `SpDF` sampling methods. The maps were generated with the parameters used in Figure 5.1 for Compression ratio comparison, with fixed seeds between different methods. All maps are presented with the two baselines, i.e., the *constrained* and *empty* experiments processed with the `Max Density` filter. The number of points in the baselines is 89 288 and 97 016, respectively.

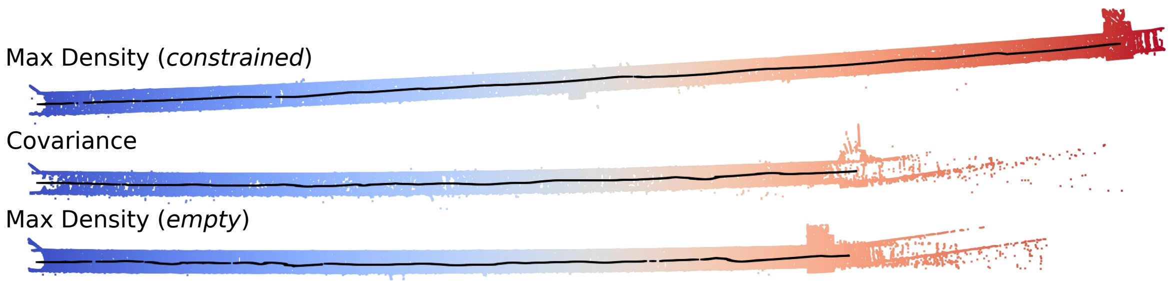


**Figure E.1:** Comparison of point clouds created by the `Covariance` and the baselines. The colour corresponds to the x-axis distance. All three maps are bent despite the actual tunnel being straight. Top and bottom maps were created with the `Max Density` filter, applied to data from *constrained* and *empty* experiments. The position during the mapping for the `Covariance` was considerably underestimated, reporting a final tunnel length almost as short as the *empty* experiment. The middle map contains 93 859 points.

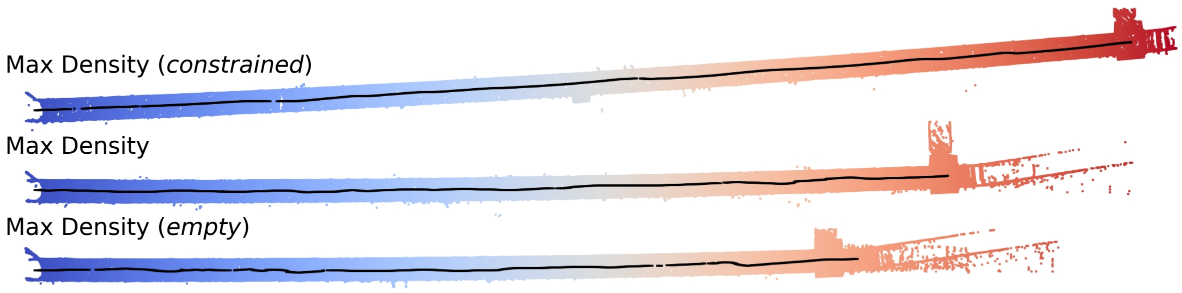


**Figure E.2:** Comparison of point clouds created by the `Max Density` and the baselines. The colour corresponds to the x-axis distance. All three maps are bent despite the actual tunnel being straight. Top and bottom maps were created with the `Max Density` filter and applied to data from *constrained* and *empty* experiments. The total tunnel length reconstructed in the `Max Density`'s final map lies between the lengths from the *constrained* and *empty* baselines. The middle map contains 105 846 points.

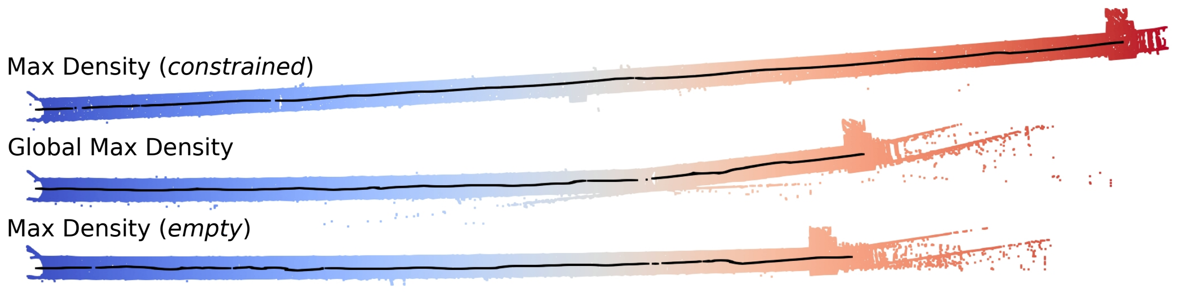


**Figure E.3:** Comparison of point clouds created by the `Global Max Density` and the baselines. The colour corresponds to the x-axis distance. All three maps are bent despite the actual tunnel being straight. Top and bottom maps were created with the `Max Density` filter and applied to data from *constrained* and *empty* experiments. The `Global Max Density` map goes straight until roughly two-thirds of the tunnel, where we can notice a sudden leap in the trajectory as the map breaks left. The middle map contains 107 212 points.

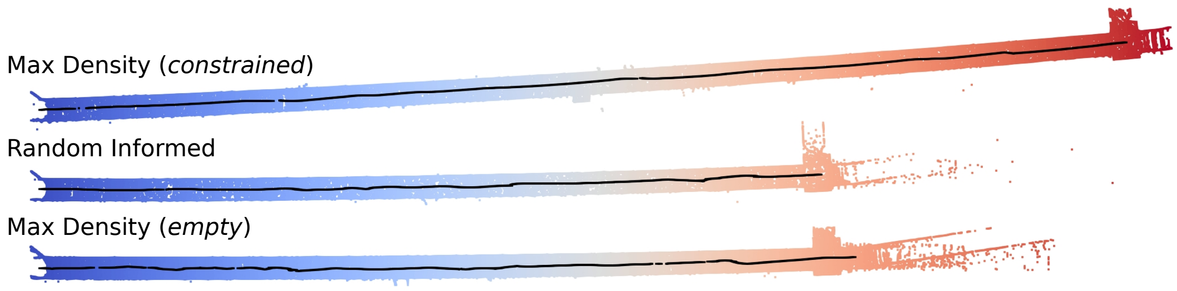


**Figure E.4:** Comparison of point clouds created by the `Octree Voxel` and the baselines. All three maps are bent despite the actual tunnel being straight. Top and bottom maps were created with the `Max Density` filter and applied to data from *constrained* and *empty* experiments. The `Random Informed` is even shorter than the *empty*, indicating an even strongly underestimated robot position throughout the trajectory. The middle map contains 93 856 points.

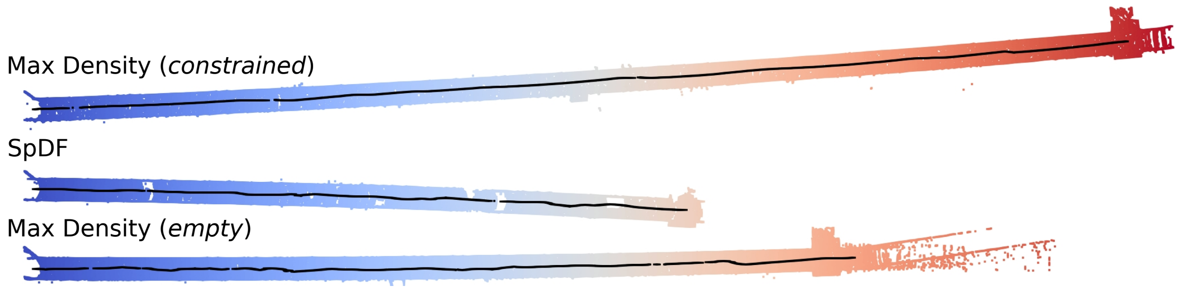


**Figure E.5:** Comparison of point clouds created by the `SpDF` and the baselines. All three maps are bent despite the actual tunnel being straight. Top and bottom maps were created with the `Max Density` filter and applied to data from *constrained* and *empty* experiments. The `SpDF` map is clearly shorter than two-thirds of the *constrained* tunnel length, with the total explored distance even shorter than *empty*. Unlike the other maps, it bends towards the right. The number of points in the middle map is 165 589. It may come to our attention that the map contains about 70 % more points than the baselines and other examined maps. While the parameter used by the `SpDF` filter to produce the map was the same as in Figure 5.1, that figure reported the CRs as the median of 20 runs. Since the `SpDF` is the least stable method both in terms of the errors and the CR, the inputs of the map in this figure lead to a high number of points being kept. Other maps produced with the same parameter contain less than 110 000 points, closer to the number of points we saw in other maps. A closer analysis of the three geometric primitives, i.e., surfaces, curves and junctions, showed that the environment contains hardly any curves and junctions compared to surfaces. The only points marked as curves are those in the corners between walls and ceiling and walls and floor. The junction points were either placed randomly within the tunnel or on the reflective cones. However, the reconstruction of cones was insufficient to avert the map bending and the low quality of the environment reconstruction.

# Appendix F
## Bibliography

[1] R. H. Helle and H. G. Lemu, “A case study on use of 3D scanning for reverse engineering and quality control”, *Materials Today: Proceedings*, vol. 45, pp. 5255–5262, Jan. 2021.

[2] A. Golovinskiy, V. G. Kim, and T. Funkhouser, “Shape-based recognition of 3D point clouds in urban environments”, in *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009, pp. 2154–2161.

[3] X. F. Han, J. S. Jin, M. J. Wang, W. Jiang, L. Gao, and L. Xiao, “A review of algorithms for filtering the 3D point cloud”, *Signal Processing: Image Communication*, vol. 57, pp. 103–112, Sep. 2017.

[4] N. Gelfand, L. Ikemoto, S. Rusinkiewicz, and M. Levoy, “Geometrically stable sampling for the ICP algorithm”, in *Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings.*, 2003, pp. 260–267.

[5] R. A. Jarvis, “A Laser Time-of-Flight Range Scanner for Robotic Vision”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, pp. 505–512, 5 Sep. 1983.

[6] G. Beheim and K. Fritsch, “Range finding using frequency-modulated laser diode”, *Applied Optics*, vol. 25, p. 1439, 9 May 1986.

[7] C. Suchocki, M. Damięcka-Suchocka, J. Katzer, J. Janicka, J. Rapiński, and P. Stałowska, “Remote Detection of Moisture and Bio-Deterioration of Building Walls by Time-Of-Flight and Phase-Shift Terrestrial Laser Scanners”, *Remote Sensing*, vol. 12, p. 1708, 11 May 2020.

[8] S. Kruapech and J. Widjaja, “Laser range finder using Gaussian beam range equation”, *Optics & Laser Technology*, vol. 42, pp. 749–754, 5 Jul. 2010.

[9] *Puck lidar sensor, high-value surround Lidar, Available at `https://velodynelidar.com/products/puck/`*, Jun. 2022.

[10] S.-P. Deschênes, D. Baril, V. Kubelka, P. Giguère, and F. Pomerleau, “Lidar Scan Registration Robust to Extreme Motions”, *2021 18th Conference on Robots and Vision (CRV)*, pp. 17–24, May 2021.

[11] J. Laconte, S.-P. Deschênes, M. Labussière, and F. Pomerleau, “Lidar Measurement Bias Estimation via Return Waveform Modelling in a Context of 3D Mapping”, May 2019, pp. 8100–8106.

[12] X. Huang, G. Mei, J. Zhang, and R. Abbas, “A comprehensive survey on point cloud registration”, *arXiv:1409.0876*, Mar. 2021.

[13] P. Besl and N. D. McKay, “A method for registration of 3-D shapes”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 239–256, 2 Feb. 1992.

[14] Y. Chen and G. Medioni, “Object modeling by registration of multiple range images”, *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2724–2729, 1991.

[15] K. S. Arun, T. S. Huang, and S. D. Blostein, “Least-squares fitting of two 3-d point sets”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, pp. 698–700, 5 Sep. 1987.

[16] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, “Comparing ICP variants on real-world data sets: Open-source library and experimental protocol”, *Autonomous Robots*, vol. 34, pp. 133–148, 3 Apr. 2013.

[17] A. Segal, D. Haehnel, and S. Thrun, “Generalized-ICP”, in *Robotics: Science and Systems V*, Jun. 2009.

[18] V. Kubelka, M. Vaidis, and F. Pomerleau, “Gravity-constrained point cloud registration”, Mar. 2022, *arXiv:2203.13799.*

[19] J. L. Bentley, “Multidimensional binary search trees used for associative searching”, *Communications of the ACM*, vol. 18, pp. 509–517, 9 Sep. 1975.

[20] M. Labussière, J. Laconte, and F. Pomerleau, “Geometry Preserving Sampling Method Based on Spectral Decomposition for Large-Scale Environments”, *Frontiers in Robotics and AI*, vol. 7, Sep. 2020.

[21] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, “3D ShapeNets: A deep representation for volumetric shapes”, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 07-12-June-2015, Jun. 2015, pp. 1912–1920.

[22] E. Nezhadarya, E. Taghavi, R. Razani, B. Liu, and J. Luo, “Adaptive Hierarchical Down-Sampling for Point Cloud Classification”, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020, pp. 12 953–12 961.

[23] P. Cignoni, C. Montani, and R. Scopigno, “A comparison of mesh simplification algorithms”, *Computers and Graphics (Pergamon)*, vol. 22, 1 1998.

[24] F. Uccheddu, M. Servi, R. Furferi, and L. Governi, “Comparison of mesh simplification tools in a 3D watermarking framework”, in *Smart Innovation, Systems and Technologies*, vol. 76, Springer Science and Business Media, 2018, pp. 60–69.

[25] L. Asgharian and H. Ebrahimnezhad, “How many sample points are sufficient for 3D model surface representation and accurate mesh simplification?”, *Multimedia Tools and Applications*, vol. 79, pp. 29 595–29 620, 39-40 Oct. 2020.

[26] M. Li and L. Nan, “Feature-preserving 3D mesh simplification for urban buildings”, *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 173, pp. 135–150, Mar. 2021.

[27] M. Pauly and M. Gross, “Spectral Processing of Point-Sampled Geometry”, in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’01, New York, NY, USA: Association for Computing Machinery, 2001, 379–386.

[28] W. Błaszczak-Bąk, J. Janicka, C. Suchocki, A. Masiero, and A. Sobieraj-Żłobińska, “Down-Sampling of Large LiDAR Dataset in the Context of Off-Road Objects Extraction”, *Geosciences*, vol. 10, p. 219, 6 Jun. 2020.

[29] W. Blaszczak-Bak, “New Optimum Dataset method in LiDAR processing”, *Acta Geodynamica et Geomaterialia*, vol. 13, pp. 381–388, 4 Jun. 2016.

[30] C. Moenning and N. A. Dodgson, “Fast Marching farthest point sampling”, University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-562, Apr. 2003.

[31] T. F. Gonzalez, “Clustering to minimize the maximum intercluster distance”, *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985.

[32] J. Qi, W. Hu, and Z. Guo, “Feature Preserving and Uniformity-controllable Point Cloud Simplification on Graph”, *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 284–289, Dec. 2018.

[33] J. Nubert, S. Khattak, and M. Hutter, “Self-supervised Learning of LiDAR Odometry for Robotic Applications”, in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, May 2021, pp. 9601–9607.

[34] D. Maturana and S. Scherer, “VoxNet: A 3D Convolutional Neural Network for real-time object recognition”, in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2015-December, Sep. 2015, pp. 922–928.

[35] R Martin, I. Stroud, and A. Marshall, “Data reduction for reverse engineering”, *RECCAD, Deliverable Document 1 COPERNICUS project, No 1068*, p. 111, 1997.

[36] K. Lee, H. Woo, and T. Suk, “Data Reduction Methods for Reverse Engineering”, *The International Journal of Advanced Manufacturing Technology*, vol. 17, pp. 735–743, 10 May 2001.

[37] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: an efficient probabilistic 3D mapping framework based on octrees”, *Autonomous Robots*, vol. 34, pp. 189–206, 3 Apr. 2013.

[38] H. Houshiar, D. Borrmann, J. Elseberg, and A. Nüchter, “Panorama based point cloud reduction and registration”, in *2013 16th International Conference on Advanced Robotics, ICAR 2013*, 2013.

[39] D. Meagher, “Geometric modeling using octree encoding”, *Computer Graphics and Image Processing*, vol. 19, pp. 129–147, 2 Jun. 1982.

[40] R. Schnabel and R. Klein, “Octree-based point-cloud compression”, in *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*, Boston, Massachusetts: Eurographics Association, 2006, 111–121.

[41] J. Elseberg, D. Borrmann, and A. Nüchter, “One billion points in the cloud – an octree for efficient processing of 3D laser scans”, *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 76, pp. 76–88, Feb. 2013.

[42] S. Rusinkiewicz and M. Levoy, “Efficient variants of the ICP algorithm”, *Proceedings of International Conference on 3-D Digital Imaging and Modeling, 3DIM*, pp. 145–152, 2001.

[43] T.-H. Kwok and K. Tang, “Improvements to the Iterative Closest Point Algorithm for Shape Registration in Manufacturing”, *Journal of Manufacturing Science and Engineering*, vol. 138, 1 Jan. 2016.

[44] T.-H. Kwok, “DNSS: Dual-Normal-Space Sampling for 3-D ICP Registration”, *IEEE Transactions on Automation Science and Engineering*, vol. 16, pp. 241–252, 1 Jan. 2019.

[45] C. Weber, S. Hahmann, and H. Hagen, “Sharp Feature Detection in Point Clouds”, in *2010 Shape Modeling International Conference*, 2010, pp. 175–186.

[46] C. Weber and S. Hahmann, "Methods for feature detection in point clouds", in *OpenAccess Series in Informatics*, vol. 19, 2011, pp. 90–99.

[47] S. Chen, D. Tian, C. Feng, A. Vetro, and J. Kovačević, "Fast resampling of three-dimensional point clouds via graphs", *IEEE Transactions on Signal Processing*, vol. 66, pp. 666–681, 3 Feb. 2018.

[48] Z. Wu, Y. Zeng, D. S. Li, J. Liu, and L. Feng, "High-volume point cloud data simplification based on decomposed graph filtering", *Automation in Construction*, vol. 129, Sep. 2021.

[49] Y. Ma, X. Zhu, S. Zhang, R. Yang, W. Wang, and D. Manocha, "TrafficPredict: Trajectory prediction for heterogeneous traffic-agents", in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[50] K. Zhang, S. Qiao, X. Wang, Y. Yang, and Y. Zhang, "Feature-preserved point cloud simplification based on natural quadric shape models", *Applied Sciences (Switzerland)*, vol. 9, 10 May 2019.

[51] G. Medioni, C.-K. Tang, and M.-S. Lee, "Tensor Voting: Theory and Applications", *Reconnaissance des formes et Intelligence Artificielle (RFIA)*, Jan. 2000.

[52] M. Gong, Z. Zhang, and D. Zeng, "A new simplification algorithm for scattered point clouds with feature preservation", *Symmetry*, vol. 13, pp. 1–15, 3 Mar. 2021.

[53] S. Yuan, S. Zhu, D. S. Li, W. Luo, Z. Y. Yu, and L. W. Yuan, "Feature preserving multiresolution subdivision and simplification of point clouds: A conformal geometric algebra approach", in *Mathematical Methods in the Applied Sciences*, vol. 41, John Wiley and Sons Ltd, Jul. 2018, pp. 4074–4087.

[54] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation", in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Institute of Electrical and Electronics Engineers Inc., Nov. 2017, pp. 77–85.

[55] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", *Advances in Neural Information Processing Systems*, vol. 2017-December, Jun. 2017.

[56] Y. Shen, C. Feng, Y. Yang, and D. Tian, "Mining Point Cloud Local Structures by Kernel Correlation and Graph Pooling", in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Dec. 2018, pp. 4548–4557.

[57] Y. He and C. H. Lee, "An Improved ICP Registration Algorithm by Combining PointNet++ and ICP Algorithm", in *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*, Apr. 2020, pp. 741–745.

[58] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Niessner, "ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes", in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2017-January, Jul. 2017, pp. 2432–2443.

[59] A Geiger, P Lenz, C Stiller, and R Urtasun, "Vision meets robotics: The KITTI dataset", *The International Journal of Robotics Research*, vol. 32, pp. 1231–1237, 11 Sep. 2013.

[60] I. Lang, A. Manor, and S. Avidan, "SampleNet: Differentiable Point Cloud Sampling", in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 7575–7585.

[61] Y. He, X. Ren, D. Tang, Y. Zhang, X. Xue, and Y. Fu, "Density-preserving Deep Point Cloud Compression", Apr. 2022, *arXiv:2204.12684*.

[62] S. A. Bello, S. Yu, and C. Wang, "Review: deep learning on 3D point clouds", Jan. 2020, *arXiv:2001.06280*.

[63] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep Learning for 3D Point Clouds: A Survey", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, pp. 4338–4364, 12 Dec. 2021.

[64] D. Fernandes, A. Silva, R. Névoa, C. Simões, D. Gonzalez, M. Guevara, P. Novais, J. Monteiro, and P. Melo-Pinto, "Point-cloud based 3D object detection and classification methods for self-driving applications: A survey and taxonomy", *Information Fusion*, vol. 68, pp. 161–191, Apr. 2021.

[65] M. Ye, S. Xu, and T. Cao, "Hvnet: Hybrid voxel network for lidar based 3d object detection", in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 1628–1637.

[66] L. Ma, Y. Li, J. Li, C. Wang, R. Wang, and M. A. Chapman, "Mobile laser scanned point-clouds for road object detection and extraction: A review", *Remote Sensing*, vol. 10, p. 1531, Sep. 2018.

[67] R. Zhong, J. Wei, W. Su, and Y. F. Chen, "A method for extracting trees from vehicle-borne laser scanning data", *Mathematical and Computer Modelling*, vol. 58, pp. 733–742, 3-4 Aug. 2013.

[68] J. Tremblay, M. Béland, R. Gagnon, F. Pomerleau, and P. Giguère, "Automatic three-dimensional mapping for tree diameter measurements in inventory operations", *Journal of Field Robotics*, vol. 37, pp. 1328–1346, 8 Dec. 2020.

[69] J. Tremblay, "Forest inventory with a lidar-equipped robot for difficult environments", *Available at `http://hdl.handle.net/20.500.11794/37634`*, M.S. thesis, Université Laval, 2019.

[70] Y. Long and D. Morris, "Lidar Essential Beam Model for Accurate Width Estimation of Thin Poles", in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 2281–2287.

[71] D. Landry, F. Pomerleau, and P. Giguère, "CELLO-3D: Estimating the Covariance of ICP in the Real World", in *2019 International Conference on Robotics and Automation (ICRA)*, vol. 2019-May, May 2019, pp. 8190–8196.

[72] M. Brossard, S. Bonnabel, and A. Barrau, "A New Approach to 3D ICP Covariance Estimation", *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 744–751, 2020.

[73] R. Wang, J. Bach, J. Macfarlane, and F. P. Ferrie, "A new upsampling method for mobile LiDAR data", in *2012 IEEE Workshop on the Applications of Computer Vision (WACV)*, Jan. 2012, pp. 17–24.

[74] L. Hyun-bin, E. su Kim, P. Rathnayaka, and P. Soon-Yong, "Low-Resolution LiDAR Upsampling Using Weighted Median Filter", in *Advances in Computer Science and Ubiquitous Computing*, Springer Singapore, 2021, pp. 213–220.

[75] G. Guy and G. Medioni, "Inference of surfaces, 3D curves, and junctions from sparse, noisy, 3D data", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 1265–1277, 11 1997.

[76] *C16, 16-Line Mechanical LiDAR*, *Available at* `http://www.lslidar.com/en/cx/103`, Jun. 2022.

[77] *RS-LiDAR-16, Powerful 16 laser-beam LiDAR*, *Available at* `https://www.robosense.ai/en/rslidar/RS-LiDAR-16`, Jun. 2022.

[78] *Trimble S7, Robotic Total Station, Available at* `https://geospatial.trimble.com/products-and-solutions/trimble-s7`, Jun. 2022.

[79] J. Tuley, N. Vandapel, and M. Hebert, "Analysis and Removal of Artifacts in 3-D LADAR Data", in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 2203–2210.

[80] J. Levinson and S. Thrun, "Unsupervised Calibration for Multi-beam Lasers", in *Experimental Robotics: The 12th International Symposium on Experimental Robotics*. 2014, vol. 79, pp. 179–193.

[81] F. Pomerleau, A. Breitenmoser, M. Liu, F. Colas, and R. Siegwart, "Noise characterization of depth sensors for surface inspections", in *2012 2nd International Conference on Applied Robotics for the Power Industry (CARPI)*, vol. 2012-January, 2012, pp. 16–21.

[82] *HD2 treaded ATR Tank Robot Platform*, *Available at* `https://www.superdroidrobots.com/robots/robotic-kits-platforms/tracked-robots/product=789`, Jul. 2022.

[83] Stanford Artificial Intelligence Laboratory et al., *Robotic Operating System*, *Available at* `https://www.ros.org`, version ROS Melodic Morenia, May 23, 2018.

[84] M. Vaidis, P. Giguère, F. Pomerleau, and V. Kubelka, "Accurate outdoor ground truth based on total stations", in *2021 18th Conference on Robots and Vision (CRV)*, 2021, pp. 1–8.

[85] S. O. H. Madgwick, "An efficient orientation filter for inertial and inertial/magnetic sensor arrays", University of Bristol (UK): Bristol, UK, Tech. Rep., 2010.

[86] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 376–380, 4 Apr. 1991.

[87] B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions", *Journal of the Optical Society of America A*, vol. 4, p. 629, 4 Apr. 1987.

[88] Z. Zhang and D. Scaramuzza, "A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry", in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018, pp. 7244–7251.

[89] D. Baril, S.-P. Deschênes, O. Gamache, M. Vaidis, D. LaRocque, J. Laconte, V. Kubelka, P. Giguère, and F. Pomerleau, "Kilometer-scale autonomous navigation in subarctic forests: challenges and lessons learned", *Field Robotics*, vol. 2, pp. 1628–1600, 2022.

[90] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems", in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 573–580.