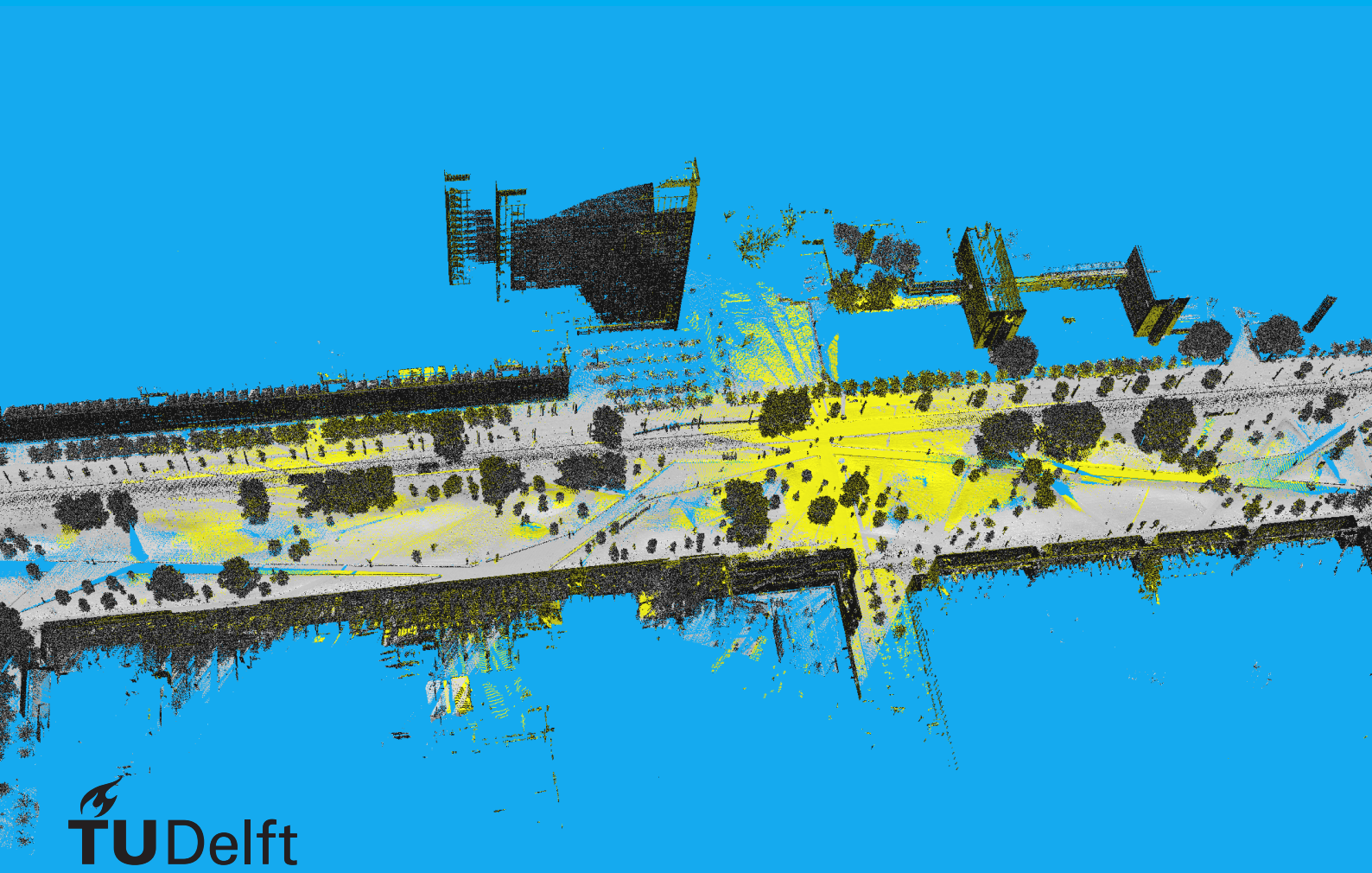
Deep Localization of Static Scans in Mobile Mapping Point Clouds

F. Meng

Geosciences & Remote Sensing
Delft University of Technology



TU Delft

Deep Localization

of Static Scans in Mobile Mapping Point Clouds

by

F. Meng

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on 18th November, 2020.

Student number:	4841204	
Project duration:	December 1, 2019 – October 30, 2020	
Thesis committee:	Dr. R. C. Lindenbergh,	Optical and Laser Remote Sensing, TU Delft, chair
	Dr. J. F. P. Kooij,	Intelligent Vehicles, TU Delft
	Dr. Y. Zang,	Optical and Laser Remote Sensing, TU Delft

This thesis is confidential and cannot be made public until November 13, 2020.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Localization is a problem of 'where we are'. Localization techniques help people understand their surrounding environment based on extracted position information in a geographic reference map. The development of global navigation satellite system (**GNSS**), light detection and ranging (**LiDAR**), computer vision (**CV**), etc., enables us to apply localization techniques for more specific tasks. Autonomous driving or robotics needs a reliable localization technique that both instantly retrieves an accurate positioning result and detailed information of the environment, which is challenging to realize in an urban environment. In our research, we propose a 3D point cloud localization framework in the urban environment to realize localization of terrestrial laser scanning (**TLS**) static scans in mobile laser scanning (**MLS**) point clouds. 3D point cloud localization consists of two steps: **place recognition** which aims to find the location of a **TLS** point cloud in a big **MLS** point cloud and **pose refinement** which accurately aligns **TLS** and corresponding **MLS** point clouds. In **place recognition**, about 100 cylinder-like objects with 2D coordinate information are retrieved per million points in a **TLS** scan or a **MLS** scene, taking nearly 15 seconds. 5 **TLS** static scans in the experiment are all successfully matched to corresponding **MLS** scenes by the Gaussian mixture model (**GMM**) probabilistic estimation based on extracted cylinder-like objects. **Results** indicate that it is possible to realize place recognition by designing an object feature descriptor instead of a fine feature descriptor, even though **TLS** and **MLS** point clouds are collected at different times (2016 and 2020). A cylinder-like object feature descriptor saves time, compared with taking more than 100 seconds per million points during point feature extraction. In **pose refinement**, point clouds of **TLS** scans and corresponding **MLS** scenes are resampled and inputted to a neural network which consists of feature extraction block (**FEB**), correspondence search block (**CSB**) and pose estimation block (**PEB**). The output of the neural network is tuned by the global refinement process to realize pose refinement. The accuracy (ϵ_r, ϵ_t), expressed as the mean rotation error ϵ_r in degrees (deg) and the mean translation error ϵ_t in meters (m), is (0.25 deg, 0.88 m) if the ground truth varies from (0, 0, 0) to (10 deg, 10 deg, 30 deg) in rotation and (0, 0, 0) to (30 m, 30 m, 10 m) in translation w.r.t. the (x, y, z) axis. The accuracy of a point-based registration neural network and traditional registration methods point-to-point iterative closest point (**ICP**), coherent point drift (**CPD**) is (~10 deg, ~10 m), (0.27 deg, 0.95 m) and (0.40 deg, 1.94 m), respectively. The run time of processing a million points for our neural network, a point-based neural network, point-to-point **ICP** and **CPD** is ~5 seconds, ~3 seconds, ~50 seconds and >100 seconds, respectively. **Results** prove that current point-based neural networks cannot work in an urban area, but our neural network achieves a more accurate result than some traditional registration methods. A neural network for pose refinement does not need prior information and is more efficient and more general than traditional registration methods as well. Overall, our research contributes to a general framework of 3D point cloud localization, incorporating both a traditional feature extraction method and a novel neural network. We hope this localization framework can be extended to other scenarios more than the urban environment, as well as other further applications e.g. urban city reconstruction.

Contents

1 Introduction

1.1. Localization

Localization is a technique helping people understand where they are in a strange environment, and humanity has longed to achieve it since a long time ago. In ancient days, the world was ambiguous to us. When people traveled for a long distance without a map, they would depend on some natural objects such as the sun and Polaris, the pole star, even trees, as well as make or remember some typical marks during the journey, to make sure they are on the right way and induce where they are, especially in the forest or the desert. However, sometimes those objects are invisible and the environment is always variable, so it is unreliable to only depend on experience. We need something stable that is helpful for localization. The magnetic compass is the earliest tool that human beings applied for navigation, it was widely used in ancient sailing, wars, surveying and mapping, etc. This great invention remarkably expands the scope of human activities, and accelerates the progress of human communication. As a result, some great ancient civilizations came into being and different people all over the world started to know each other.

In the era Age of Discovery and Renaissance, modern compass was applied by an amount of navigators, and at the same time there was great progress in modern science. Consequently, the earth was much more clear to us than ever before: the world was mapped, modern geography was founded, international trade was frequent. The fundamentals has been built for current localization.

In 19th century, James Clerk Maxwell derived the electric and magnetic wave equations, which was a breakthrough for modern physics. Thanks to that, many innovative techniques based on electromagnetic waves were invented in the next century, including those for modern localization, such as radar, Loran-C which is a hyperbolic radio navigation system, etc.

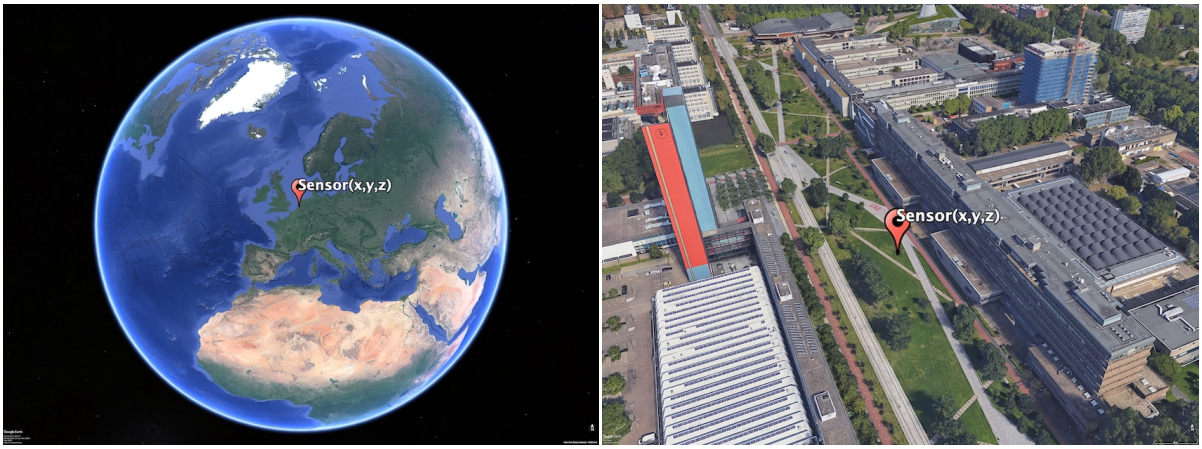
After the construction of global positioning system (**GPS**) in late 20th century, localization techniques have reached a new stage. In recent several decades many new techniques, such as near field communication (**NFC**), bluetooth, light detection and ranging (**LiDAR**), inertial navigation system (**INS**), have been applied to retrieve more details for specific tasks where global navigation satellite system (**GNSS**) cannot manage. There is a trend of system integration. Nowadays, people no longer worry that they would be lost in a strange environment, and many products are more and more accessible with simple user interface design.

1.1.1. Definition

Modern localization can be defined as the problem to find the geographic location of a sensor in a reference map. Localization is slightly different from positioning. As Fig. 1.1 shows, the result of localization is usually expressed by a relative position of the sensor in a well-known environment, while positioning gives the absolute coordinates of the sensor in a georeferenced coordinate system, such **WGS84** for **GPS**. Overall localization provides more semantic information than positioning, and it is usually applied for specific tasks but requires more detailed information of surroundings. Positioning can be applied at a wider scale, and it can be seen as the basic service for localization.

1.1.2. Applications

Localization techniques have been used in many fields. For example, **GNSS**, the most classical method for positioning, is the fundamental method of localization in different scenarios such as deformation monitor-



(a) (b)

Figure 1.1: (a) Positioning in a global coordinate system (b) Localization at TU Delft campus

ing for engineering [43] or volcanoes [74], transportation planning [45], etc. There are also novel applications, such as guide robots in museums where localization is achieved by some sensors [12], a pedestrian navigation of mobile augmented reality based on location for tourists [64], **NFC** which is a more direct way to quickly get localization information [21], indoor localization by **WIFI** [13] even the smart phone [63], etc. It almost covers every aspect of human's life.

1.2. GNSS Based Localization

GNSS is a generally used positioning technique. Based on time difference of signal transmission and Doppler Effect, position and velocity of the receiver are computed. Under a satellite constellation mainly consisting of a geosynchronous Earth orbit (**GEO**), medium Earth orbit (**MEO**), inclined geosynchronous satellite orbit (**IGSO**) satellites, we can realize worldwide services as long as we are in open area.

1.2.1. Development

Transit is the first navigation satellite system developed by the US navy, also called navy navigation satellite system (**NNSS**). Due to the fact that it was the first attempt, there are many shortcomings such as discontinuity and low accuracy and efficiency, but it provides the fundamental for the **GPS**. The **GPS** project was started in 1973, and completely built in 1993. It is more improved and reliable than Transit with satisfying positioning results, as it also provides service for timing. Initially the accessibility of this system is limited to civilians under the selective availability (**SA**) policy, but access restrictions were canceled in 2000 because of the development of the international situation and the sharply increasing demand in business, scientific research, etc.

There are three main other global navigation satellite systems: **GLONASS** by Russia, **BeiDou** by China and **Galileo** by the European. After developing for several decades, **GNSS** achieves high accuracy continuously at a relatively low cost, under the help of augmentation systems such as ground based the augmentation system (**GBAS**) and the Internet, by which people accurately know where they are through their map applications in mobile phones. In recent years, there is a trend of integration by combining **GNSS** with other systems, such as **INS** to solve the discontinuity in tunnels or other situations where the signal is interrupted.

1.2.2. Challenges

The information provided by **GNSS** is limited, and we need more except coordinates, to better understand our surrounding environment and the Earth. That is why we still do surveying and mapping, although **GNSS** provides the global coverage service. In other words, map is the other key to realize **GNSS** localization. Fortunately, the Earth is becoming more clear to us because of the development of surveying and mapping, especially the great progress in photogrammerty and measuring instruments. **GNSS** also provides surveying and mapping. As a result, **GNSS** has been widely applied in many different fields and is almost available to everyone.

Still there exist some circumstances where **GNSS** localization cannot work. For example, in a street canyon or underground park where the signal is interrupted because of skyscrapers or ground, we cannot depend on our map application in the smart phone. Some retrieved semantic information in a closed or narrow space cannot be made full use through **GNSS** because of the open environment limitation. For example, in a business block, there is a map describing the layout of shops, through **GNSS** we know how to reach those shops which interest us. If we are inside a big supermarket, we can only depend on some marks in the map, as **GNSS** is not available in an indoor area. Many applications start to offer indoor map for large shopping malls, but localization is still challenging within an indoor environment. Most importantly, autonomous driving and autonomous robotics are developing fast in recent years. It demands for a much more accurate positioning result and detailed geographic information for localization, which **GNSS** sometimes cannot achieve [38]. Cars or robots always have different types of sensors in addition to **GNSS** receivers. Another problem of **GNSS** for autonomous vehicles is that it is hard to monitor objects e.g. pedestrians in real time. If we want to retrieve a **GNSS** positioning result, a receiver is necessary. Although we have receivers in our smart phones, not everyone allow the applications access the positioning information all the time out of privacy and there are always some people who do not bring their phones. Integrating positioning information from different applications and different types of phones is a challenge as well. As a result, the robot can hardly make the correct decision only by **GNSS**, when it locates in a crowded environment.

To solve those problems, there are many methods proposed. For example, a tightly coupled **GNSS/INS** integration method [39] is designed to improve the robustness of the system with a low cost, but it also requires the vehicle runs under a satellite-observable environment most of the time. Some **GNSS**-free techniques have been developed e.g. self-localization based on visual odometry [11] which is a technique of extracting ranges and directions from camera for dead reckoning. In one word, different techniques compensate to make localization more robust and reliable, as well as generally used in different aspects of life and science.

1.3. Point Cloud

A point cloud is set of points describing the surface geometry, as Fig. 1.2 shows. 3D point clouds well represent the real world intuitively, and they are insensitive to lighting and seasonal changes [3], so a map generated by point clouds is stable and reliable if there are not large changes for certain areas. One of the contributions of point clouds is the reconstruction of the real world, such as digital visualization or preservation for ancient architectures, which helps people browse them online, as well as help their maintenance and protection. Point clouds provide some parameters to developers to make special effects or animations [2]. Overall, point clouds well connect the real world to the virtual world, thus a lot of applications are coming into being.

1.3.1. Laser Scanner

A laser scanner is an instrument for collecting 3D point clouds, with controlled deflection of laser beams [49]. A typical laser scanner contains a laser ranging unit, a opto-mechanical scanner, a control and processing unit. It is extensively used in entertainment industry for motion capture, virtual reality, etc. Compared to classical surveying and mapping, a laser scanner can collect a large amount of data at one time, and after classification and segmentation, the map has more details. The laser scanner extends the application of localization in a way, as we acquire some typical information in certain scenarios e.g. tree's distribution in the forest, car's flow in an urban area, layout of street lights along a road, all of which enables us to do more specific tasks out of different purposes.

1.3.2. Point Cloud based Localization

3D point clouds can also realize localization. Registration, an alignment method between different sets of point clouds, makes it possible to do the alignment between a local scan and a reference map if there exist overlaps between two pairs of point clouds, thus a rigid transformation matrix is retrieved, by which we would know the exact position of the scan if the reference map is accurate enough. One of the applications is simultaneous localization and mapping (**SLAM**) [6, 18] which is metric or appearance-based [32], while it is still a challenge to create a reliable reference map, since the reference map is generated by the robot itself from different fields of view. In general, mobile laser scanning (**MLS**) point clouds provide a relatively high resolution 3D map in a large scale [7], so it is a good reference map during registration. Iterative closest point (**ICP**) [10] is the standard method to do the registration for 3D point clouds, where correspondences are searched between two scans, and some prior information e.g. a coarse transformation matrix is needed before doing this method. Random sample consensus (**RANSAC**) [22] improves the robustness of point cloud

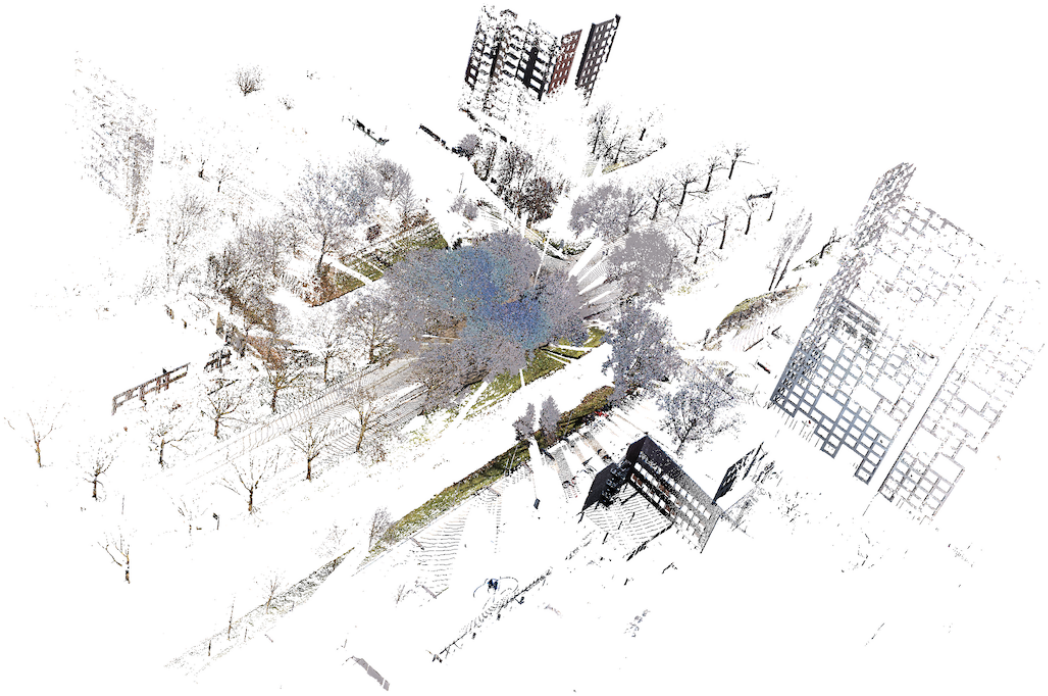


Figure 1.2: An urban point cloud shown in RGB color

localization by randomly select some points for checking outliers. The final accuracy is also affected by the quality of inputs and prior information.

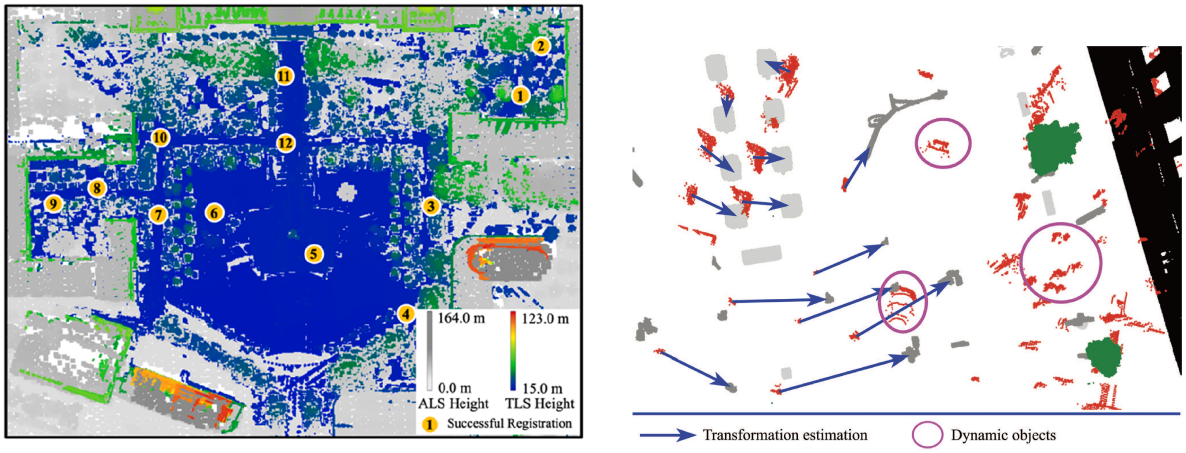
As for localization methods based on point clouds, a lot of have been proposed in different mapping systems. From terrestrial laser scanning (**TLS**) to airborne laser scanning (**ALS**), a novel skyline context descriptor has been designed based on height differences between a point and its neighbors, under the invariance of z-axis between these two systems [40], as Fig. 1.3 (a) shows. There is an innovative algorithm from **TLS** to **MLS** for large-scale registration that applies an unsupervised learning method with auto encoder-decoder in a neural network to reduce the dimension of candidate search [19]. A real time localization based on segmented **MLS** is also introduced, mainly for vehicles [56], as Fig. 1.3 (b) shows. Some other techniques, such as LocNet [90] based on **SLAM**, [87] with a stereo camera also proves the availability of point cloud localization. Hierarchical localization [69] solves localization based on 2D images with a coarse-to-fine method.

1.4. Deep Learning

Deep learning (**DL**) is a subclass of machine learning (**ML**), which came up in the 1970s. The key idea is to apply multiple layers, including many hidden layers within a neural network to extract features or estimate results we need, by giving high quality of data for training parameters. Initially, due to the limitation of computing power, more works were proposed only from the theoretical aspect, and in most cases the results were not as good as some well designed hand crafted classical methods. As the development of integrated circuit follows Moore's law which succeeds to predict the number of transistors in a dense integrated circuit (**IC**) doubles about every two years, the performance of computer improves remarkably, doubling every two years, and there is an explosion of information, all of which make it possible to apply **DL** with big data.

1.4.1. Development

Currently, there are three main neural networks designed: artificial neural network (**ANN**), convolutional neural network (**CNN**) and recurrent neural network (**RNN**). Artificial neural network is an early algorithm in comparison to other neural networks. Its design was inspired by human brain structure of nervous system. So it is a classical method in deep learning. Convolution neural network is a kind of network which is mainly designed for image analysis, and in recent years, the dimension of its application has been extended from 2D to 3D. Recurrent neural network is another neural network designed to do some sequential tasks such as handwriting recognition [25] and machine translation [80]. Generally speaking, different neural networks



(a) (b)

Figure 1.3: (a) Localization from **TLS** to **ALS** [40] (b) Localization based on a simple segmentation map for vehicles [56]

have consistent kernels where distinct features are extracted for different inputs during training, to make it possible to realize some tasks we design specifically.

1.4.2. Deep Learning for Point Cloud Processing

Similarly, there are also three main types of neural network for processing point clouds: voxel-based, image-based and point-based [61]. The main difference for these three networks is the input format. VoxNet [50] is the pioneer in voxel-based **CNN**, which has good performance but the training process is time consuming. It is also a challenge to assess neighborhood information for a voxel-based network. Image-based **CNN**s project 3D point clouds into a sequence of images to do shape recognition [78], object detection [58], registration [95], etc., but this kind of network has some problems such as loss of scene information. Point-based **CNN** is a prevalent design which firstly came up with PointNet [61]. As Fig. 1.4 shows, it directly works on 3D point clouds with several multi layer perceptrons (**MLP**), and has proved good performance in point cloud classification and segmentation [61, 62]. Later point-based **CNN**s for localization or registration inspired by PointNet have also been proposed, such as PPFNet [16] and PointNetLK [4]. The **CNN** framework for 3D point clouds performs reconstruction or completion tasks, such as road marking extraction, classification and completion [86]. It will develop fast with promising futures, since the huge amount of point cloud data can be easily collected by laser scanners or cameras, with detailed information captured, which enables us to define a more specific neural network for certain tasks.

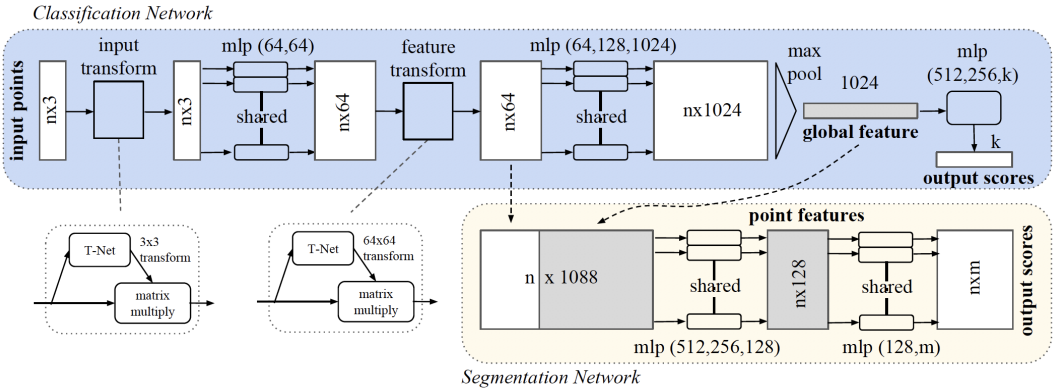


Figure 1.4: PointNet [61]: a point-based neural network for point cloud processing

1.5. Problem Statement & Challenges

Point cloud based localization is an extension of point cloud registration, and people have designed many methods to show the reliability of it. In deep learning, point cloud localization is still a new field. Some networks we mentioned have proven the possibility of point cloud registration or localization, and a well pre-trained neural network obviously decreases the complexity of design for registration and can be better than traditional registration methods under certain circumstances [4]. But it needs more testing, especially in a general scenario at a large scale, such as an urban area. Currently, most registration neural networks focus on datasets of single objects with relatively simple structure e.g. bunnies, chairs, as the source and the template, which is easy to visualize and check if something goes wrong. In a larger area, there would be thousands of objects with multi scales, in which some objects cannot be directly inputted such as buildings, cars with high density of points, and some are not regular with seasonal changes e.g. trees, leading to errors between the source and the template. DeepVCP [46] has designed a network for urban scenes, to extract key virtual point correspondences to realize registration, which is similar to **ICP**. But it highly depends on extracted key points and includes 3D **CNN** layers which is the same as those layers in voxel-based **CNN**s, so it is time consuming, and hard to define what is a key point in given point clouds.

In terms of our research, we want to extend the deep learning application of point clouds in a larger area, and check how deep learning improves localization in comparison to traditional registration methods. We focus on localization between static scans (**TLS**) and mobile mapping (**MLS**) point clouds in the urban environment, only with coordinates as input. Considering properties of **TLS** and **MLS** point clouds in such scenario, we divide the localization algorithm into two steps: **place recognition** and **pose refinement**.

1.5.1. Place Recognition

Since our **TLS** data is partially overlapping mobile mapping data, and usually **MLS** data covers a much larger area than **TLS**, we need to select the corresponding part from **MLS** data for a given **TLS** scan, which we call place recognition. As Fig. 1.5 shows, **MLS** data almost covers the whole campus but **TLS** data only covers a small area where we set a laser scanner in the center for data collection. Point density of **MLS** data is uniform, while **TLS** data has much higher point density around the center area than surroundings. **Place recognition** is defined as the problem to find the most similar scene of reference map (**MLS**) compared with the local scan (**TLS**). The output should be a corresponding pair (TLS_i, MLS_j). We want to realize it by extracting hand-crafted features, to see the contribution of classical methods. There are also some neural networks, e.g. PointNetVLAD [3] to do place recognition, but this deep learning method is better only in the situation where you need to recognize every part of mobile mapping, which is not our case. In addition, we want to prioritize the efficiency in this step because we do not need to retrieve an accurate registration result.

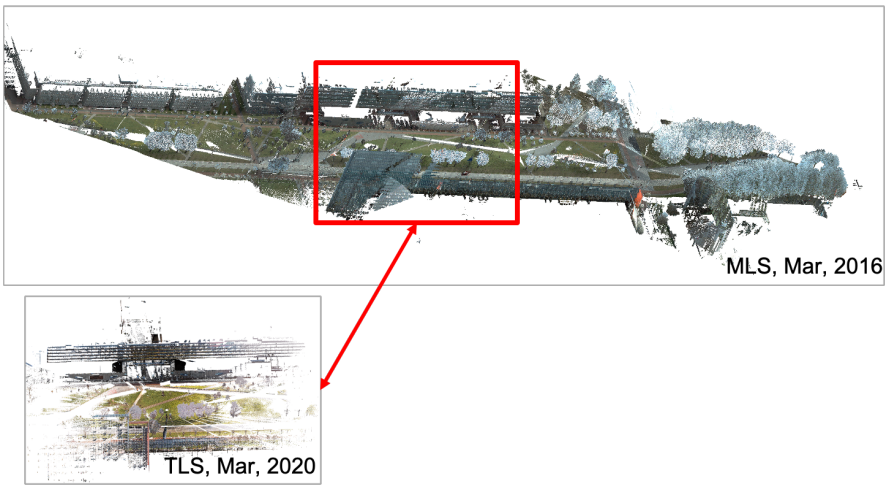


Figure 1.5: Place recognition example (The **TLS** point cloud is matched to the area within a red box in the **MLS** reference map)

1.5.2. Pose Refinement

After we get the **TLS** point clouds and the corresponding part of **MLS** point clouds as the source and the template, respectively, we will use them as input to further align **TLS** and **MLS** point clouds. As Fig. 1.6 shows,

pose refinement is defined as the problem to accurately estimate the transformation matrix between the source and the template. We want to realize it by a novel neural network, which extends the scope of neural network design for registration.

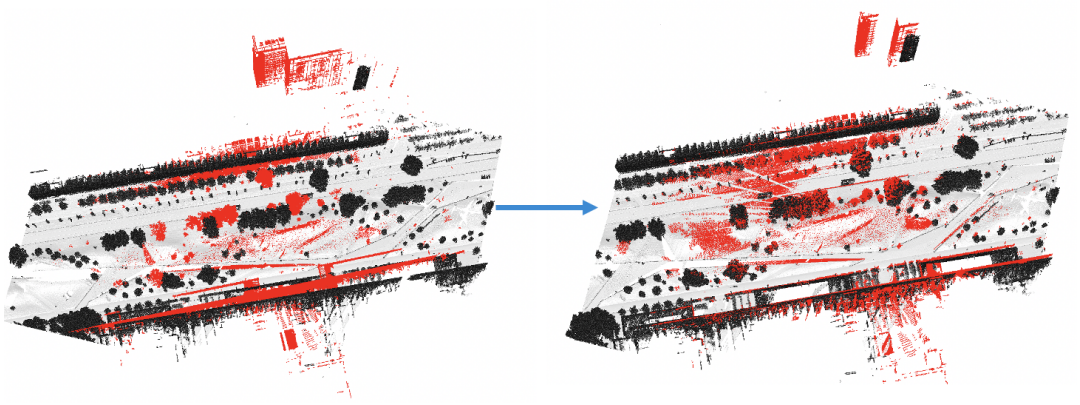


Figure 1.6: Pose refinement example (red represents **TLS** data and grey represents **MLS** data, a difference of ~3 degrees between **TLS** and **MLS** data in the left figure is corrected in the right figure)

1.5.3. Processing Challenges

There are some challenges to do point cloud localization at a relatively large scale. In place recognition, different from traditional segmentation or registration methods which design a fine feature descriptor to accurately achieve the result, we prioritize the efficiency for this step, because our goal is recognition. If a descriptor finds which part of the **MLS** point clouds is what we need, it is good enough. Dense and fine features are useful in the situation where two point clouds are close, but our point clouds are collected by different systems (**TLS** and **MLS**) at different times (2016 and 2020). The descriptor should be as simply designed as possible and can work in most cases. It is possible to design such a descriptor based on some common objects or clusters for feature extraction in the urban environment.

As for pose refinement, firstly, there are millions of points in the source or the template, so we cannot input them directly to the neural network. In general, each sample in deep learning on point cloud processing, also known as batch, has less than 1.0×10^5 points, which ensure the efficiency during training is not bad and the network is not too complicated. As a result, some preprocessing procedures are needed, specifically the source and the template should be downsampled and divided into many small parts as input. During this data compression and splitting process, we also need to think about how to properly process our data to totally or partially preserve some helpful features during training process. The input size should be the same for each batch in the neural network as well. Secondly, an extra problem needs to be solved following the preprocessing. We need to search corresponding pairs between the source and the template, as we split the whole scene to ensure the efficiency. Finally, in complicated scenes, we must consider the robustness and the accuracy aspect, especially when the source and the template are collected at different times. The design of a such kind of neural network is challenging but innovative, which should work well in the urban environment.

1.6. Research Objectives

The main question of our research is:

How to realize localization in the urban environment, by means of an innovative registration neural network and by traditional methods?

with some sub-questions:

1. How to organize the input data of large size?
2. What is the workflow of place recognition and pose refinement?
3. How to realize place recognition in a relatively simple way?

4. How to design our registration neural network and select proper hyperparameters during pose refinement?
5. What is the performance of place recognition and pose refinement?
6. What are improvements of our neural network in comparison with point-based neural networks and traditional methods?

1.7. Organizations of this Thesis

This chapter gave a brief introduction to the development of localization techniques, especially **GNSS**, point cloud localization and deep learning, as well as the purpose of our research. In Chapter 2 we will introduce some works which are closely related to our research. Chapter 3 contains descriptions of different data structures, as well as preprocessing methods before place recognition and pose refinement. In Chapter 4 we will introduce method for **place recognition**, and Chapter 5 for **pose refinement**. Chapter 6 gives results for both **place recognition** and **pose refinement**. Chapter 7 is related to conclusions and recommendations, and we will answer questions proposed in section 1.6 **research objectives**.

2

Point Cloud Localization Related Techniques

This chapter firstly introduces components and characteristics of terrestrial and mobile laser scanning system in Section 2.1. Then three methods to process point clouds aiming at scene understanding, classification & segmentation, registration, shape matching & object recognition are reviewed in Section 2.2. Section 2.3 and Section 2.4 discuss two point cloud processing methods that we will use in place recognition and pose refinement, feature descriptor and point-based neural network, respectively.

2.1. Point Cloud Acquisition

3D point cloud acquisition is offered performance using laser scanning systems or photogrammetry. A typical laser scanning system consists of three parts: a laser scanner, a geographic framework, and a control unit [85]. A laser scanner is an instrument which uses a laser beam to detect and measure the distance to an object. It uses scanning mirrors to steer the laser beam. A geographic framework provides a local or a global coordinate system to represent point clouds in 3D space. A control unit processes and records acquired data, to output various types of information e.g. 3D coordinates, station ID. A laser scanning system collects up to millions of points with high point density. The coverage depends on the range and varies per instrument. Apart from coordinate information, laser scanning systems also record intensity and color information [31]. Depending on the platform, there are different types of laser scanning systems. There are three systems that are closely related to urban environment: terrestrial laser scanning (**TLS**) system and mobile laser scanning (**MLS**) system and airborne laser scanning (**ALS**) system. In our research we concentrate on **TLS** and **MLS** systems.

2.1.1. Terrestrial Laser Scanning System

TLS systems collect 3D point clouds by installing a scanner on a stationary tripod, or by handheld scanners. The most obvious advantage of **TLS** systems is high point density and stable representation of the real world, especially under good measurement condition, it can realize consecutive scanning to get more than 10 scans per hour in an area. Occlusions have a large influence on **TLS** systems. Point clouds acquired by **TLS** systems are usually dense around the scanner but more sparse in its surroundings. Scanning an area from different fields of view is time consuming, so a large coverage is challenging for **TLS** systems. **TLS** systems are applied for data collection in unreachable or invisible areas for humans e.g. cave surveying [53], detailed scene understanding e.g. cultural heritage documentation [23], industrial surveying e.g. deformation monitoring [81], etc. The geographic framework is a local coordinate system where the scanner is the origin, and xOy plane is usually vertical to the local plumb-line.

2.1.2. Mobile Laser Scanning System

MLS is the most recently developed type of laser scanning system [82]. It is a moving laser scanning system to acquire 3D geo-referenced point clouds. Typically, a scanner is installed on the top of a vehicle together with global positioning system (**GPS**), inertial measurement unit (**IMU**) and cameras. A vehicle mounted **MLS** system efficiently collects point clouds in a large area e.g. a whole street block within an hour. Different

fields of view in a **MLS** system decrease the impact of occlusions. Various types of **MLS** systems are reviewed in [60]. **MLS** systems are used in many fields e.g. extraction of road markings [26], road inventory studies [59]. The geographic framework in **MLS** systems provides a global coordinate system, under the help of **GPS** and **IMU** to extract satellite positioning and orientation information. Point clouds acquired by **MLS** systems can be used as a reference map in point cloud localization since it covers a large area and provides detailed information. Well represented point clouds by **MLS** systems help the visualization of a 2D map to extend this 2D map to 3D, based on 3D model reconstruction [35].

2.2. Point Cloud Scene Understanding

Point cloud scene understanding is defined as how to output a specific result based on extracted features of point clouds to match the human defined standard, under different circumstances. A simple example is that during motion capture, many sensors are installed in different parts of the body of the actor. It precisely captures any slight motion or facial expression shown by the actor for digital motion or facial expression interpolation on computers, based on acquired point clouds from sensors. The output of the point clouds is classified as different components of the body automatically if we give each sensor an unique ID, and thus the sensor builds a one-to-one mapping between point clouds and body parts. It is easy to realize and we can get many details around important parts such as face and joints, to enable us vividly show the action of the virtual character in films or video games. While this technique is very expensive and time consuming; its application depends a lot on the investor. Besides, for a single person we can achieve motion capture easily by installing several tens of sensors, while it would be troublesome if there are thousands of different characters. That is also one reason why motion capture must be applied selectively. As for point clouds in a large or complicated area, they cover a lot of information, so many algorithms are designed to finish specific tasks. Based on the purpose they are dived into three types: classification & segmentation, registration, shape matching & object recognition.

2.2.1. Classification & Segmentation

Classification and segmentation are closely related to each other. Traditionally, classification refers to label a given data to give a class for certain set of point clouds which contain many objects with similar attributes. It is generally used when we need to quickly extract groups of data for later processing e.g. training, detection, segmentation, etc. Segmentation is the process of partitioning point clouds into multiple segments, to extract more details for different parts of point clouds. Because of that, algorithms are designed more specifically based on common attributes for segments, thus the classification process is necessary usually. For example, assume a dataset of **MLS** is given, covering an urban area. After classification, we will be able to classify the roads, buildings, vegetation, etc., by supervised learning. Then for each classification result, we retrieve more information by applying segmentation algorithms, such as facades on buildings, cars and street lights on roads, etc.

Related Work

A point feature is the representation of a point in another mathematical space created by a defined computation method. Feature extraction based methods have a general way for classification, such as a multi-scale feature based classification [84, 94] which retrieves features under different resampling cells, to form a hierarchical framework for classification. After extracting features a classifier will be constructed. In terms of point-based deep learning methods, there are slight differences compared with traditional feature extraction methods. The target is usually a single object, as PointNet [61] and PointNet++ [62] show and some other networks e.g. 3DMFV [9], thus it is more like classification on single segments. There are also networks for classification at a larger scale, such as [27] which is voxel-based. Because of the high efficiency of point-based convolutional neural networks, segmentation can be done conveniently as another output in the same framework with classification [61]. Apart from feature extraction methods, there is a lot of research about segmentation such as histogram-based methods [79], edge detection [97] in computer vision, and some of them have been applied or extended in 3D space. In a word, classification and segmentation are becoming more integrated, and we can easily adapt our output based on the content of input.

2.2.2. Registration

Registration aims to find a transformation matrix from source point clouds to template point clouds, to align two different coordinate systems, thus a relative position is retrieved for the source in the template. Specif-

ically, the rigid transformation matrix which we will use in our research is shown in Eq. (2.1) where $\mathbf{R}$, $\mathbf{T}$ are rotation matrix and translation vector; $\mathbf{0}_3$ is a 1×3 vector with all 0 elements; θ_x, θ_y, θ_z are rotation angles, and δ_x, δ_y, δ_z are translations w.r.t. the (x, y, z) axis, respectively; $\mathbf{\Phi}$ is the final rigid transformation matrix. In our research, source point clouds are our **TLS** static scans and template point clouds are **MLS** data collected by a specific vehicle. A prerequisite for point cloud registration is that there is overlap between the source and the template, and the overlap degree should not be low. The processing of registration really depends on the quality of the input, and usually the more points, the longer run time. Registration is the fundamental technique for point cloud localization. In other words, point cloud localization does an extra step of data organization to first make sure the source and the template close as much as possible, then point cloud registration is applied.

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_x) & sin(\theta_x) \\ 0 & -sin(\theta_x) & cos(\theta_x) \end{bmatrix} \quad \mathbf{R}_y = \begin{bmatrix} cos(\theta_y) & 0 & -sin(\theta_y) \\ 0 & 1 & 0 \\ sin(\theta_y) & 0 & cos(\theta_y) \end{bmatrix} \quad \mathbf{R}_z = \begin{bmatrix} cos(\theta_z) & -sin(\theta_z) & 0 \\ sin(\theta_z) & cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$\mathbf{R} = \mathbf{R}_z \cdot \mathbf{R}_y \cdot \mathbf{R}_x \quad \mathbf{T} = \begin{bmatrix} \delta_x & \delta_y & \delta_z \end{bmatrix}^T \quad \mathbf{\Phi} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}_3 & 1 \end{bmatrix} \tag{2.1}$$

Related Work

Traditional iterative closest point (**ICP**) [10] registration works directly on points or normals of points to search corresponding pairs between the source and the template, and it has been improved to come up some new **ICP** based methods such as **G-ICP** [72] which combines two classical **ICP** registration methods: point-to-point **ICP** [10] and point-to-plane **ICP** [10] into a probabilistic framework. Most classical methods of registration have similar principle as classification and segmentation. The difference is that registration designs a descriptor for point matching between the source and the template, to form corresponding pairs. By applying some statistical method we estimate the final pose as output. [29] summarizes some 3D descriptor designs in recent years, and there are some image descriptors e.g. SIFT [44], shape content descriptor [8] that are extended from 2D to 3D, specifically, 3D SIFT [71] and 3D shape content [24]. Some descriptors are also inspired by segmentation methods e.g. FPFH [66] which also extracts multidimensional feature histograms for registration. In deep learning, registration neural network is proposed later than classification and segmentation. Especially after point-based neural networks e.g. PointNet [61], PointNetLK [4], PCRNet [70] and other similar networks came up. Although most current registration neural networks have some limitations and challenges in accuracy and robustness, there is more and more research trying to design a reliable registration neural network.

2.2.3. Shape Matching & Object Recognition

Shape matching & object recognition can be seen as a mixed problem of classification/segmentation and registration, depending on its purpose. For example, face recognition is a prevalent technique which has been intensively applied in a lot of companies. It is a problem that belongs to classification since the images as input are always human faces, and it is enough to recognize different faces from different people. As for car detection, it is more like segmentation since not only cars appear in the image, but also pedestrians, bicycles, lights, trees, etc. In addition, if we want to recognize some regular shapes, there will be distortions for single shape in different images, such as rigid or affine transformation. Under this circumstance we need to estimate the rigid or the affine transformation matrix, then the problem is close to a registration problem. Overall, processing shape matching & object recognition is flexible, depending on what people expect to output.

Related Work

In order to solve shape matching & object recognition problems out of different purposes, a prevalent method is probabilistic estimation. From this type of method, we can recognize our targets, as well as get a transformation matrix which can be accurate if there is high quality of point clouds with less noise. Gaussian mixture model (**GMM**) is a well known model which is based on probability density function (**PDF**) to formulate a maximum-likelihood estimation framework [52]. It assumes the input data is normally distributed. Based on expectation-maximization (**EM**) step proposed by [15], we find solutions by guessing parameters at first, then tuning parameters by minimizing the difference of a objective function during iterations. Coherent point drift (**CPD**) [54] improves **GMM** performance to enable non-rigid transformation by adding a regularization term to the original **GMM**. In terms of other methods, as we mentioned, shape contexts [8] realize shape

matching and object recognition by computing the similarity based on shape context descriptor to search correspondences. Current face recognition systems are designed with a neural network framework, with satisfying output which has been tested by some big information technology companies such as Google and Baidu.

2.3. Feature Descriptor

A feature descriptor is an algorithm that encodes the input e.g. point clouds into a feature vector. Ideally each vector is unique for its independent input. Generally a feature descriptor describes input point clouds locally and globally with variant scales and corresponding algorithms. Feature descriptors are divided into two types: hand-crafted and learned.

2.3.1. Hand-crafted Feature Descriptors

Many hand-crafted feature descriptors are now available in Point Cloud Library [65]. A hand-crafted descriptor is usually designed based on mathematical relationships for a point and its neighbors, e.g. fast point feature histogram (**FPFH**) [66] is extracted based on normals. Obviously, they are manually designed, and applied in classification/segmentation, registration and recognition. They are sometimes suffering from noise, low resolution, and only designed for specific data types [91] e.g. most hand-crafted feature descriptors need high point density to extract features from a point and its neighbors. Still some output satisfying results for specific tasks, e.g. a camera localization method [55] combines 2D and 3D scale invariant features transform (**SIFT**) descriptors [44] and rotation invariant feature transform (**RIFT**) descriptors [37] to realize pose estimation. The key to an effective descriptor is to find a local reference frame free (LRF-free) method. [66] uses rotation invariant point pair features (**PPF**) [17]. **PPF** is also applied in a neural network method, PPFNet [16], which is shown in Fig. 2.1.

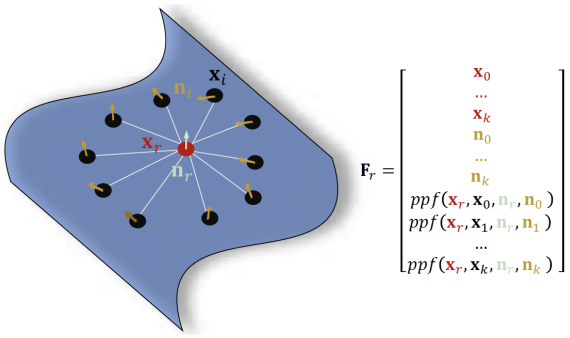


Figure 2.1: A traditional hand-crafted feature descriptor based on point pair feature (**PPF**) (PPFNet [16])

2.3.2. Learned Feature Descriptors

A learned feature descriptor is closely related to a convolutional neural network. For the purpose of registration, frameworks are voxel-based e.g. 3DMatch [91], image-based such as PoseNet [32] or point-based e.g. 3DRegNet [57], etc. Learned feature descriptors lack mathematical explanation compared with hand-crafted feature descriptors and suffer from black box training. Learned feature descriptors automatically output the final result by minimizing the impact of noise in different scenarios. The minimizing process is finished by back propagation under defined loss function. Back propagation is an algorithm that optimizes parameters in a neural network during training based on a metric that describes the difference between output and truth. The way of computing this metric is also known as the loss function. In other words, an effective neural network can 'learn' itself for specific tasks if we establish a good back propagation with a well defined loss function. For different areas, we might need to define various hand-crafted feature descriptors to separately extract information for different scenes, or improve an existed descriptor to lessen the impact of noise, especially in complicated cases. If we design an effective neural network, the only thing we need to do is to provide enough training data, then the neural network can fit all complicated scenes well under the same framework. To design such an effective neural network is challenging, and some researchers are trying to insert hand-crafted feature descriptors into a neural network, to minimize the influence of black box train-

ing, e.g. PPFNet [16] that uses **PPF** and PointNetVLAD [3], with local features extracted from PointNet as the input of NetVLAD to get the final global descriptor for place recognition. Overall, learned feature descriptors are developing fast, especially those which are point-based, inspiring more researchers to design more novel descriptors. Fig. 2.2 shows an innovative learned feature descriptor for point cloud segmentation. It applies a neural network to extract features for a neighbor of a point, then aggregates features of five neighbors as edge features for this point.

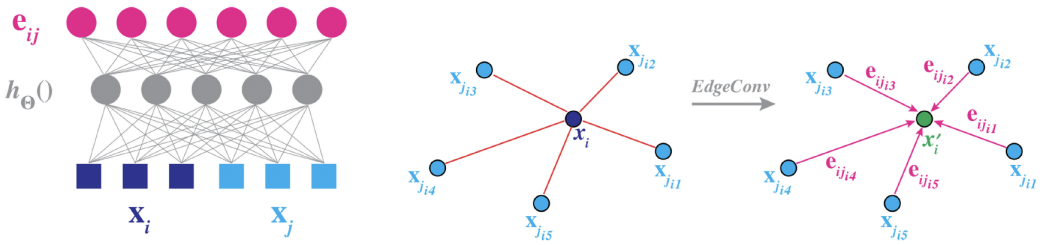


Figure 2.2: An innovative learned feature descriptor: EdgeConv from DGCNN [83]

2.3.3. Feature Extraction Methods for Point Cloud Localization

Learned features are usually related to a neural network, which we will introduce later. Different types of hand-crafted features are widely applied in point cloud localization.

Point-based features are common in feature extraction methods for point cloud localization. A point-based feature is extracted from a point and its several neighbors e.g. signature of histograms of orientations (**SHOT**) [68] and viewpoint feature histograms (**VFH**) [67]. There are point-based feature place recognition methods e.g. [75] which applies normal aligned radial features [76] with a probabilistic approach, bag of words, and pose estimation (registration) methods e.g. [34] which uses k-nearest neighbor graphs [14]. It is time consuming to extract point-based features from millions of points at a large scale. High point density is usually needed to ensure the reliability of extracted point-based features. Place recognition can also be realized by visual-based methods which extract feature descriptors from images e.g. SeqSLAM [51] and SRAL [28]. These visual-based feature descriptors highly depend on the field of view to extract features as similar as possible between a query of scans for place recognition. In order to improve the performance and the efficiency of a feature descriptor, some specific methods are designed based on 3D point clouds e.g. [33] which proposes a scan context descriptor to encode a point cloud of a 3D scan into a matrix.

Fine features can be used in both place recognition and pose refinement. For example, the scan context method [33] can also be applied for coarse registration. An accurate result needs that extracted features are reliable in two point clouds. In simultaneous localization and mapping (**SLAM**), scans and the reference map are collected by the same scanner, and there are less terrain changes because the reference map is retrieved from instant scans. For a robot on the road, the scanned terrain is also stable and the field of view is always similar. As a result, there are many feature descriptors proposed for **SLAM** localization e.g. LeGO-LOAM [73] and LOAM-LIVOX [41], improved from light detection and ranging (**LiDAR**) odometry and mapping (**LOAM**) [92] and low-drift and real-time **LOAM** [93], applying planar and edge features for place recognition and pose refinement. **IMU** is tightly coupled into **LOAM** to realize a high precision in [89].

In general, different descriptors are designed for specific tasks. It is a challenge to realize place recognition and pose refinement between newly acquired point clouds and the reference map under different laser scanning systems, especially our reference map was collected several years ago where there are temporal changes for the terrain. Place recognition is our first task because we do not have techniques e.g. **IMU** or **GNSS** to help us guess an initial position. Place recognition results are intermediate results in our research, so it is enough to find a proper feature descriptor with satisfying run time (faster than a point-based feature descriptor, close to current novel feature descriptors). Although there are a lot of choices (point-based, visual-based and other specific feature descriptors), we need to make full use of **TLS** and **MLS** point cloud attributes in the urban area. Pose refinement results are final results and we focus on the improvement in comparison to other pose estimation (registration) methods.

2.4. Point-based Neural Network

Point-based neural networks inspired many researchers to attempt to design new frameworks for point cloud applications. After PointNet [61] was proposed in 2017, this method has become the standard for point cloud processing. It works directly on 3D points, different from voxel-based and image-based methods whose inputs are voxel and pixel grids. Points go through several layers with filters for convolution computation in each layer. The size of each filter is $1 \times N_D$ where N_D is the dimension of a single point, usually equal to 1×3 in the first layer and 1×1 in others. After several hidden layers the output will be a 1D vector with C_1 channels. A channel represents a unique feature space, e.g. if $C_1 = 5$, then for each point we will have 5 features to describe it. The computation of convolution decreases incredibly compared with voxel-based and image-based methods because after the first layer, the size of each filter is 1×1, then the computation process becomes a scaling process rather than a 2D or 3D convolution.

2.4.1. 2D Convolutional Neural Network in Point-based Neural Network

The application of 2D convolutional neural networks in a point-based neural network is called multi layer perceptron (**MLP**). A **MLP** is a single layer of convolution under a filter size and a number of channels. After several **MLP**s features are extracted after a max-pooling process for each channel. The number of extracted features depends on the number of channels in the last **MLP**. After several **MLP**s and a max-pooling process for feature extraction, it is different to realize classification/segmentation and registration based on extracted features. In classification, we extract N global features after max-pooling, shown as a $1 \times N$ feature vector. A classical neural network is applied to decrease the dimension of the feature vector until its dimension is equal to the number of classes we want to label. The output is a vector with probabilities of each class. During segmentation there are some other layers to apply convolution for intermediate results together with global feature vector, then the output is a 2D matrix, with probabilities of each segment for each point. Registration neural networks take various methods after feature extraction. The difference is obvious in comparison to classification/segmentation neural networks. The design of loss function is not a simple process of computing differences between 0 and 1. Various definitions of loss functions may have totally different results. One of keys to a successful registration neural network is to properly describe the loss between transformed source point clouds (prediction) and template point clouds (truth).

2.4.2. Typical Point-based Frameworks for Point Cloud Registration

There are some typical networks of point cloud registration which inspire us a lot. PointNetLK [4] comes up with an innovative method by extracting registration results from global features, extending the application of the Lucas & Kanade (**LK**) algorithm [48] from 2D images to 3D point clouds; PCRNet [70] simplified PointNetLK by applying a neural network similar to classification after feature extraction of PointNetLK. Both networks do an iterative processing to make registration results more accurate. DeepVCP [46], 3DRegNet [57] and CorsNet [36] design methods to search or check corresponding pairs. DeepVCP combines PointNet++ [62] and L3Net [47] to apply weighting and regularization for input point clouds, then extract key points as well as their features based on a certain number of neighbors; 3DRegNet applies a ResNet [30] which is a network concatenating outputs in different layers, rather than a general **MLP** for feature extraction, and output weights for each pair of correspondence; CorsNet is similar to DeepVCP but focuses on a smaller scale, where all points from the source are considered as key points with corresponding points in the template. In terms of the loss function design, some neural networks e.g. PointNetLK consider 6 pose parameters $(\theta_x,\ \theta_y,\ \theta_z,\ \delta_x,\ \delta_y,\ \delta_z)$ in Eq. (2.1) of the transformation matrix. PCRNet compares differences between transformed source and template point clouds. Weights of corresponding pairs are sometimes added to the final loss function, and usually balancing factors will be used if there is more than one term in the function of total loss.

2.4.3. Limitations of Point-based Registration Neural Network

In a point-based neural network for registration, some **MLP**s and a max pooling layer are always applied for feature extraction, to summarize 3D coordinate information into L features ($L = 128,\ 256\ , 2^n$ in general). Point permutation invariance is a property of the point-based neural network that different permutations of input points do not have an impact on features extracted from **MLP**s and a max pooling layer. It has been proved by PointNet [61]. Extracted features from **MLP**s and a max pooling layer are reliable, and most point-based registration neural networks use this framework at the feature extraction step. As for using extracted features to estimate pose parameters, there are two main types: correspondence based method and feature

based method.

In a correspondence based method, the extracted L features are concatenated to each point as global features e.g. CorsNet [36], which is similar to the segmentation network in PointNet, to find corresponding pairs between the source and the template. This method highly depends on the quality of the input, because if source and template are from different datasets at different time scales, e.g. the source from **TLS** and the template from **MLS** in our case, it is hard to promise corresponding pairs are accurate enough. Results are even worse when the source and the template are downsampled by farthest point sampling (**FPS**). DeepVCP [46] simulates the correspondence in the template rather than find it in reality in order to make corresponding pairs more reliable, but most points from the source and the template are removed, with only key points left, leading to loss of information in most areas.

As for a feature based method, it directly uses L extracted features from the source and the template to estimate the final result, by applying an artificial neural network for concatenated features e.g. PCRNet [70], or by comparing differences of features between the source and the template e.g. PointNetLK [4]. An artificial neural network consists of several fully connected layers which take a single input as a neuron, and all neurons in adjacent layers are connected. The feature based method is a global estimation for each batch. Though it is more robust than the correspondence based method, the accuracy can hardly reach a high level because we do not have clear mathematical relationships between source features and template features. In other words, the process of tuning parameters during training for feature extraction is a black box, and hard to correct if it goes wrong. The black box of training process is also a reason why most research uses small simple objects, to make sure the high quality of input dataset.

3

Spatial Data Structures

Section 3.1 introduces preprocessing methods before place recognition and pose refinement. In Section 3.2, different data structures that will be used in place recognition and pose refinement are summarized.

3.1. Preprocessing Methods

The overview of the whole research is shown in Fig. 3.1. Particularly, before each main step of place recognition in Chapter 4 and pose refinement in Chapter 5, we need to apply preprocessing methods on original point clouds, to ensure consistency between terrestrial laser scanning (**TLS**) and mobile laser scanning (**MLS**) data.

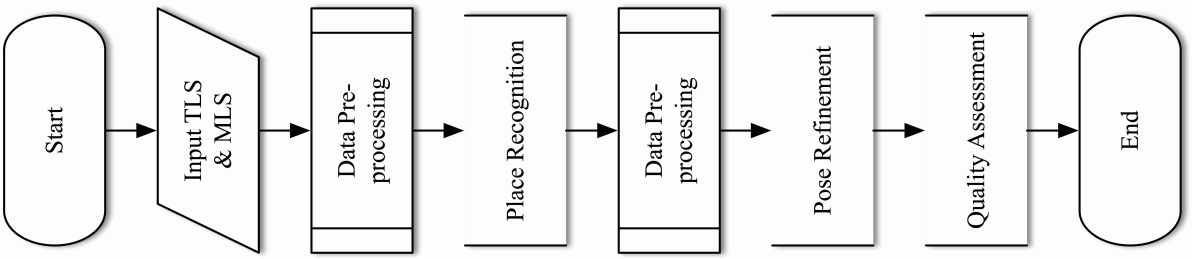


Figure 3.1: Overall workflow of 3D point cloud localization

3.1.1. Preprocessing before Place Recognition

In place recognition, the input data is collected from both **TLS** and **MLS** system, recalling with many differences. Firstly, due to the fact that data from **MLS** locates in a larger geographic system, coordinate values are usually large (from 1.0×10^4 to 1.0×10^5). We apply recentering for **MLS** coordinates (1.0×10^4, 1.0×10^5 to several hundreds) to avoid computation loss during processing. Then another necessary step is to make sure the point density is consistent between different systems, which is done by the downsampling method from Open3D [96], which is voxel downsampling with 0.1 m voxel size. After downsampling, we also apply the statistical outlier removal algorithm from Open3D with 20 nearest neighbors and 3.0 standard deviation (**STD**) ratio as threshold, to make our input less noisy. Finally, typical scenes (definition in Section 3.2.3) from **MLS** are manually selected as template point clouds for later place recognition. The **TLS** point clouds are also pre-processed by the same voxel downsampling and the same statistical outlier removal as **MLS** point clouds, to be as source point clouds during place recognition. If necessary, both scans from **TLS** and typical scenes from **MLS** are resampled by a large cell to generate a sequence of subsets to make point clouds more accessible at different scales.

3.1.2. Preprocessing before Pose Refinement

Based on results of place recognition, **TLS** point clouds and **MLS** point clouds should be close to each other with large overlap, and we generate patches (definition in Section 3.2.3) for both. Patches from **TLS** data are

defined as source patches and those from **MLS** are defined as template patches. During training we will input a large number of batches (definition in Section 3.2.3), including every possible patches from **TLS** and **MLS** point clouds, to tune parameters of our neural network during training. It is a prerequisite that the shape for each batch is the same. To enable this same shape precondition, we apply a sampling method - farthest point sampling (**FPS**) which is also applied in PointNet++ [62], to resample 256 points for every patch. **FPS** is a sampling method based on the idea of repeatedly finding the next sample point from the current least-known sampling area. Only point density decreases after **FPS**, and the presentation in real world is well kept.

3.2. Data Arrangement

Data arrangement is the process of organizing data by resampling point clouds under different data structures. It is important in both place recognition and pose refinement, because we will apply certain algorithms based on different types of geometric information of different data structures. In our research, point clouds from **TLS** and **MLS** are collected at different times. After applying preprocessing methods, **TLS** and **MLS** point clouds will have identical point density, less noise, proper range of coordinates. Then we arrange point clouds in different data structures for later processing, shown in Fig. 3.6.

3.2.1. Terrestrial Laser Scanning Data

As Fig. 3.2 (a) shows, our Leica P40 [1] **TLS** system collected point clouds of TU Delft campus from a stationary tripod, with high point density (about 20 million points per scan). Table 3.1 gives more details of the **TLS** system. In our research, each scan covers about $200m \times 200m$ area. The selection of **TLS** data in the experiment is introduced in Chapter 6.

3.2.2. Mobile Laser Scanning Data

Our **MLS** data covers the same area as **TLS** data, with the same format as [82], shown in Table 3.1, by the Fugro Drive-Map **MLS** system as Fig. 3.2 (b) shows. There are nearly 4 years passed between **TLS** data and **MLS** data acquisition, so it is challenging to realize localization between them, as many constructions or other changes occurred within these years. The selection of **MLS** data in the experiment is also introduced in Chapter 6.

Table 3.1: Description of **TLS** & **MLS** system

Parameter	**TLS**	**MLS**
Laser Pulse Rate	1,000,000 p/s	1,333,000 p/s
Ranging Accuracy	3 mm / 50 m	<2 cm
Maximum Range	270 m	100 m
Scanner	Leica Geosystems P40	Riegl VQ 250
Swath Angle	360 degrees	360 degrees
Platform	Tripod mounted	Vehicle mounted
Panoramic Camera	/	Ladybug 3
Collecting Date	March, 2020	March, 2016

3.2.3. Data Structures

During processing in place recognition or pose refinement, we use different data structures to resample our input point clouds for different purposes, at different scales. For example, if we need to extract local features, a small cell size e.g. (5 m, 5m) w.r.t. x and y axis will be defined. A larger cell size e.g. (50 m, 50 m) will be applied firstly before we need to search neighbors, to reduce computations.

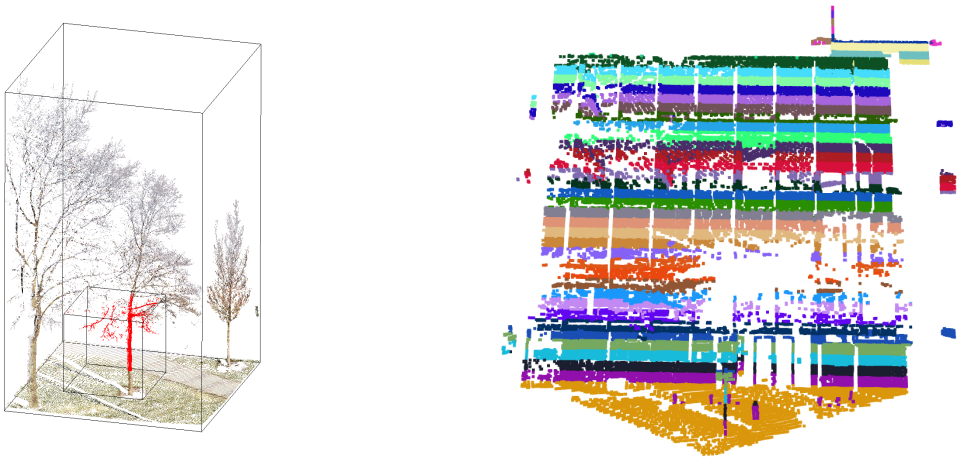
Voxel/Stixel Cell

A voxel cell is a cuboid in 3D space, with predefined edge lengths in x, y, z axis. It is widely used in classical point cloud processing, representing points in a voxel cell with a single value. A voxel cell is also useful for resampling point clouds. By defining certain voxel cells, some local features are preserved at a small scale, then local feature descriptors can be designed. In our research, point clouds within a voxel cell are helpful to realize both feature extraction in place recognition and pose refinement.

A stixel cell is similar to a voxel cell, but with only predefined edge lengths in x and y axis. The edge length of z axis of a stixel cell depends on the largest z coordinate of points within this stixel cell. In our research,

(a) (b)

Figure 3.2: (a) A static **TLS** system (b) A side view of **MLS** system [82]



(a) (b)

Figure 3.3: (a) Stixel cell (original points) & voxel cell (red points) (b) Generated slices w.r.t. height for a sparse facade

the z axis is parallel between our **TLS** and **MLS** data, so there is almost no rotation w.r.t. x and y axis. θ_x and θ_y in Eq. (2.1) are small. In some cases e.g. feature extraction during place recognition, a stixel cell is more like a 3D sliding box extended from 2D space, which is similar to a 2D sliding box in car detection. Both point clouds within a larger stixel cell and a smaller voxel cell are shown in Fig. 3.3 (a):

Slice

A slice is a set of points whose heights are within a range $(h_1,\ h_2)$, and $h_2 - h_1$ is the width for this slice. In our research, the height is the same as the coordinate in z axis. Slices of a plane, e.g. a facade, are represented as strips, as Fig. 3.3 (b) shows. Slices of a facade can be applied for vertical feature extraction e.g. vertical line feature in automatic registration [88].

Typical Scene of MLS

In urban areas, there are many blocks or corners which visually separate the **MLS** point clouds, so we manually select scenes with overlap as our typical scenes. Each typical scene may include large buildings, corners, or other obvious objects and landmarks. Then we can lessen the impact of similarity in different scenes. Fig. 3.4 shows some typical scenes without overlap area for a better visualization:

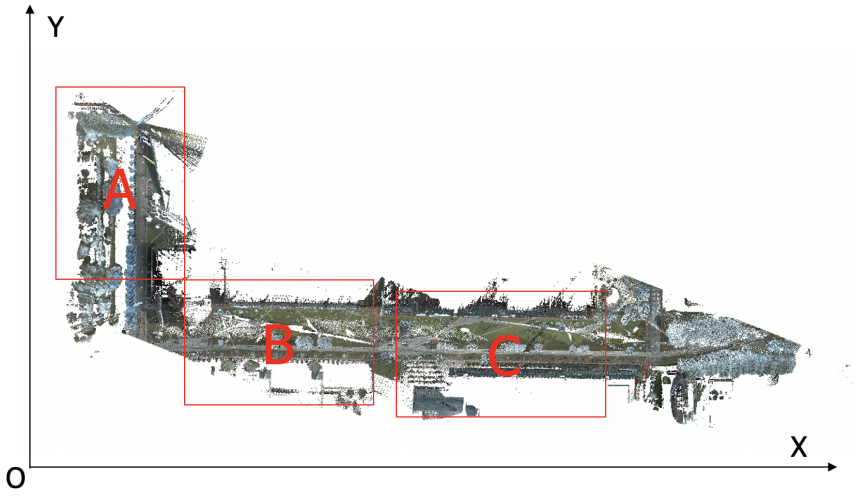


Figure 3.4: Three typical scenes (A, B, C) separated by corners in a **MLS** point cloud from an aerial view

Patch & Cluster

In general, any connected set of points after resampling by a voxel cell, a stixel cell, or slicing in our research is defined as a single patch. The patch contains local features, especially geometric information, although its visualization might be irregular without any clear class. A cluster is the output of any algorithm that connects points with close distances or geometric attributes e.g. DBSCAN, where points tend to belong to the same part of objects, as Fig. 3.5 shows:

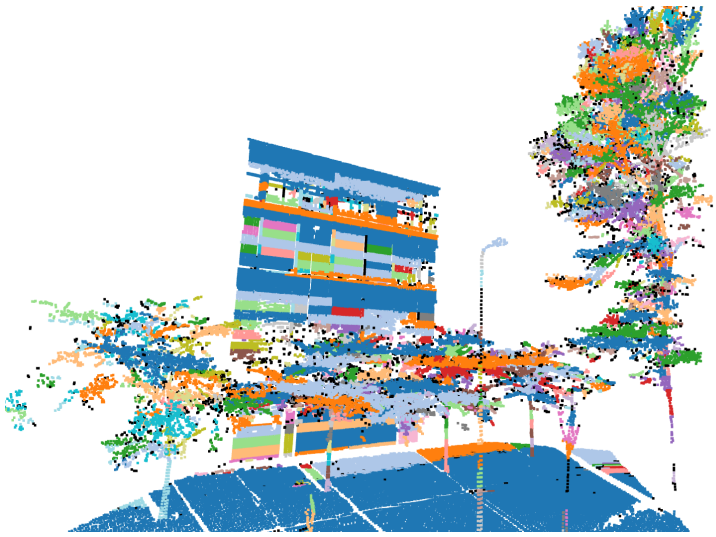


Figure 3.5: Clusters created by DBSCAN w.r.t. slices in a patch resampled from a stixel cell (black represents noise)

Batch

Batch is a concept in deep learning describing a single sample of input data. For each batch the shape should be the same during training. In our research, each batch of input point clouds should be a $N_0 \times 3 \times 1$ matrix containing a set of patches, with a defined number of N_0 points e.g. $N_0 = 2560$, which is generated by a data pre-processing method before pose refinement.

Source & Template
In general, the source in our research always consists of **TLS** data and the template always consists of **MLS** data. The concrete definition is flexible, and it can be point clouds within scans, typical scenes during place recognition, or generated patches from **TLS** scans or **MLS** scenes during pose refinement.

Overview of Different Structures
In Fig. 3.6, we give an overview of different data structures in different steps, connected our workflow. In general, original point clouds are taken as input, then depending on different purposes after preprocessing before place recognition, point clouds are resampled with stixel cells or voxel cells for further processing.

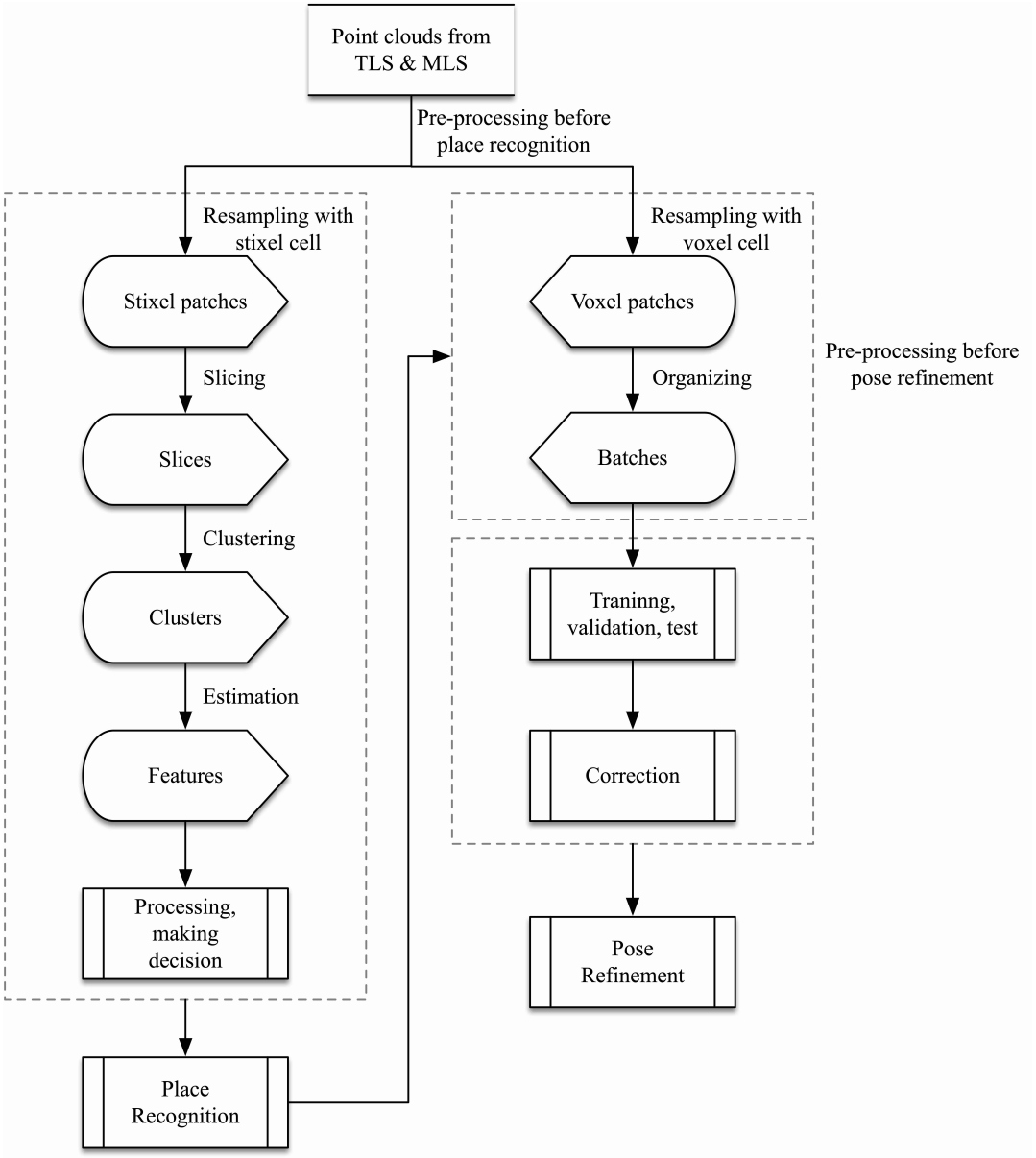


Figure 3.6: Main data structure overview in different processings

4

Method for Place Recognition

This chapter firstly reviews the place recognition problem in Section 4.1. Then characteristics of the urban environment are summarized in Section 4.2, and an object feature descriptor is decided in Section 4.3. Finally the algorithm of how to form corresponding pairs between the source and the template is introduced in Section 4.4.

4.1. Introduction

In Section 1.5 we define place recognition which aims to find the area in mobile laser scanning (**MLS**) data corresponding to an area sampled by terrestrial laser scanning (**TLS**) system. Place recognition is similar to object recognition. But due to the fact that our input size is much larger, with millions of points, we cannot work directly on point clouds. It is time consuming, and hard to describe the similarity between the source (**TLS** point clouds) and the template (**MLS** point clouds) collected at different times. Section 2.3.3 reviews some localization methods based on feature extraction and shows their limitations for our scenario. Considering attributes of our **TLS** and **MLS** data, we propose an efficient way, a hand-crafted object feature descriptor that does not need to do point-based matching, to realize place recognition.

4.2. Urban Environment

Both **TLS** and **MLS** point clouds in our research are collected in an urban area. An urban area has following characteristics:

1. **Unique appearance**. An urban area is closely related to human activities. Buildings, bridges and roads are built by construction workers; the layout of a city is designed under urban planning; even most trees are manually planted out of landscape design, especially street trees. In a large view, an urban area can consist of many blocks separated by roads, thus in Chapter 3 we introduce typical scenes of **MLS** reference map. In an natural area, we cannot find these obvious blocks and different scenes are continuous. In details, an urban area has full of geometric information that can be extracted from buildings, street trees and lights, cars, etc. This type of information can hardly be retrieved in an natural area because natural objects have more random appearances.

2. **Stable terrain**. The mobility of an urban area is large at a short time scale, e.g. pedestrians and cars go up and down the streets within a day. Pedestrian or car flow detection has been applied in the urban environments such as Auto-DeepLab [42] based on images. While at a long time scale e.g. a week, a month, even a year, the overall appearance of the real world is relatively stable, although some new trees might be planted or some constructions might be finished with time going on. If we use a laser scanner to keep collecting data for several minutes, the mobility at the short time scale will be removed as well. In one word, as long as there are not large changes, the terrain in an urban area is stable, which helps us to do the place recognition between newly acquired **TLS** scans and old **MLS** data.

3. **Asymmetric information between TLS and MLS point clouds**. Different objects in an urban area provide plenty geometric information, but sometimes bad sampling from these objects during data collection can be troublesome for retrieving features. For example, there is many building facades in an

urban area, but usually **MLS** data provides more detailed information than **TLS** data, because it is hard to well represent a whole facade in a single static scan. So it is not sufficient to only depend on features extracted from facades. Besides, many objects are easily intersected with each other e.g. cars and buildings, different trees, etc. under different viewing angles in **TLS** or **MLS** point clouds. As a result, it is difficult to remove the impact of intersection to retrieve precise point clouds for the same object from both **TLS** and **MLS** data.

4.3. Feature Descriptor Design

Based on the characteristics of the urban environment, we should find a feature descriptor that is common, stable in the urban environment, and easily retrieved, robust during feature extraction.

4.3.1. Cylinder-like Object

We choose cylinder-like objects as our feature descriptor. Strictly, there are cylinders such as street lights and pillars which well fit a cylinder equation. Besides, trees' trunks are also cylinder-like, especially street trees and most artificially planted trees of older age.

4.3.2. Contribution, Challenges & Problems

A cylinder-like object feature descriptor well corresponds to the characteristics of the urban environment, with following properties:

1. **Common object with geometric information**. There are cylinder-like objects in almost every block because trees and lights are common objects in an urban environment. In each typical scene of the **MLS** point clouds or **TLS** scan, there are always hundreds of these objects within an about 200 m × 200 m area. The representation of a cylinder in 2D space is a circle, based on this simple geometric attribute we can design several steps to easily retrieve them.

2. **Time invariance**. All cylinder-like objects are stable over time, e.g. lights and pillars will always stay if there is no construction; as for trees, they grow slowly, so a time of several years does not have a huge influence. There are some changes between our **TLS** and **MLS** data due to constructions at TU Delft campus. Overall, most cylinder-like objects e.g. trees and street lights do not change much.

3. **Robust to asymmetric information**. Cylinder-like objects are usually close to roads, so a **MLS** system scans most of them. During **TLS** point cloud acquisition, we set our tripod along the center line of the campus to make sure as many as cylinder-like objects are collected. As a result, most cylinder-like objects in the same area are shown in both **TLS** and **MLS** data. As for the impact of intersection, it decreases remarkably as long as we scan parts of a cylinder-like object. For example, if we have the upper part of a light in **TLS** data and the lower part of this light in **MLS** data, we can retrieve this light both from **TLS** and **MLS** data from 2D space.

During retrieving cylinder-like objects, some errors will impact final results. There are three main types: canopies of trees, pillars on buildings and newly planted trees.

Canopies are the most general fake cylinder-like objects because of their too variant layout of point clouds and larger data size than trunks, as Fig. 4.1 shows. It always leads to some parts of canopies being wrongly estimated as cylinder-like objects, especially for our point clouds collected at a time when canopies are obvious enough. Fortunately, these fake cylinder-like objects are always close to true values, so we merge them all to form a new virtual cylinder, which is sensible in a recognition process.

Pillars on buildings are stable regular cylinders in the urban environment, including some pillar-like objects that belong to facades, foundations, etc., with various radius from centimeters to meters. There will always be occlusions when we scan buildings from different views, and in some cases we will not intend to scan the whole facade unless for specific tasks. As a result, in comparison to trees and lights, point clouds of pillars on buildings are relatively sparse, and many of them are occasionally visible, which depends on the field of view of the scanner. Although some of them are also close to trees or lights alongside the front path of the building, it is better to remove this impact by applying filtering methods.

New trees are the most dominant error whose influence is difficult to remove because they are total new cylinder-like objects that only appear on our **TLS** scans. Applying merging or filtering algorithms is less useful for this type of error. New trees will more or less determine the final result of place recognition, which depends on how many of them are newly planted in our area.

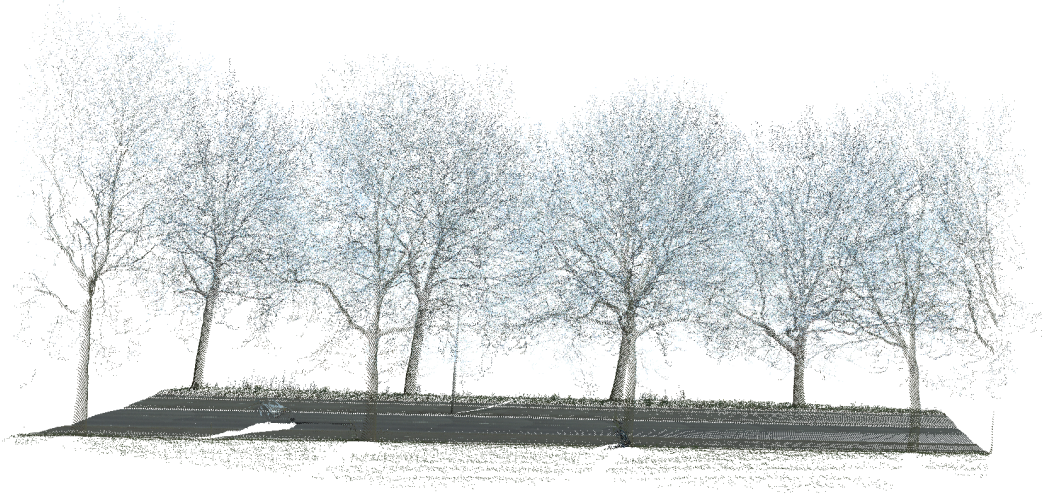


Figure 4.1: Canopies and trunks of street trees in a **MLS** scene, shown in RGB

4.3.3. Feature Extraction Method

We design a 9-step feature extraction method to retrieve all cylinder-like objects in an urban area, as well as decrease the impact of different types of error, shown in Fig. 4.2 and the upper part in Fig. 4.4. At the same time, line features from facades are also extracted and they are only applied for filtering (step 8 **pillars on buildings removal**).

1. **Resampling using stixel cells**. Cylinder-like objects are expected to be nearly vertical with different heights and radii, so we define a small stixel cell as the bounding box e.g. $5m \times 5m$ in our research for later cylinder extraction. Similarly, line features from facades can also be extracted by a stixel bounding box, with a larger size e.g. $20m \times 20m$. We will merge lines with closeby slope and intercept, as a stixel cell divides a long non-vertical line into several parts. After resampling, point clouds from **TLS** or **MLS** are divided into two types of set for cylinder and line, respectively, where $\boldsymbol{S}$ represents the set of input **TLS** or **MLS** data and $\mathbf{S}$ is a stixel patch after resampling, with superscript as index and subscript as the type of bounding box, and N^c_{stx}, N^l_{stx} are the number of stixel patches under different bounding boxes, as Eq. (4.1) shows. Fig. 4.2 (a) is a single patch resampled by a stixel cell $20m \times 20m$ in a **TLS** scan, shown in a scalar field:

$$\begin{aligned} \boldsymbol{S} &= \left\{ \mathbf{S}^1_{cylinder}, \mathbf{S}^2_{cylinder}, \dots \mathbf{S}^{N^c_{stx}}_{cylinder} \right\} \quad or \\ \boldsymbol{S} &= \left\{ \mathbf{S}^1_{line}, \mathbf{S}^2_{line}, \dots \mathbf{S}^{N^l_{stx}}_{line} \right\} \end{aligned} \tag{4.1}$$

2. **Slicing w.r.t. z axis**. Since we do not know the exact height of different cylinder-like objects, we apply slicing w.r.t. z axis, after which some slices are generated, as Fig. 4.2 (b) shows. Slicing is also suitable for line feature extraction. The width w.r.t. z axis of each slice is 0.5 m, which aims to detect and estimate as many as possible cylinder-like objects or lines. We do not use a small width e.g. 0.1 m, since point clouds are preprocessed by voxel downsampling with 0.1 m voxel size. There will be few points in a thin slice to apply clustering algorithm. Slicing is only applied for points whose z coordinate is smaller than 10 m during extracting cylinder-like objects because most of these objects are close to the ground. There is no threshold of z coordinate during line feature extraction. Several slices are generated for each stixel patch after slicing, where $\mathbf{s}$ represents a slice with superscript as index and subscript as corresponding patch, and N_{slc} is the number of slices, as Eq. (4.2) shows:

$$\mathbf{S}^i = \left\{ \mathbf{s}^1_i, \mathbf{s}^2_i, \dots \mathbf{s}^{N_{slc}}_i \right\} \tag{4.2}$$

3. **Slice clustering**. A cylinder-like object splits in small parts distributed in different slices, and a facade splits in several planes shown as strips. In order to estimate a complete object or the line of a facade,

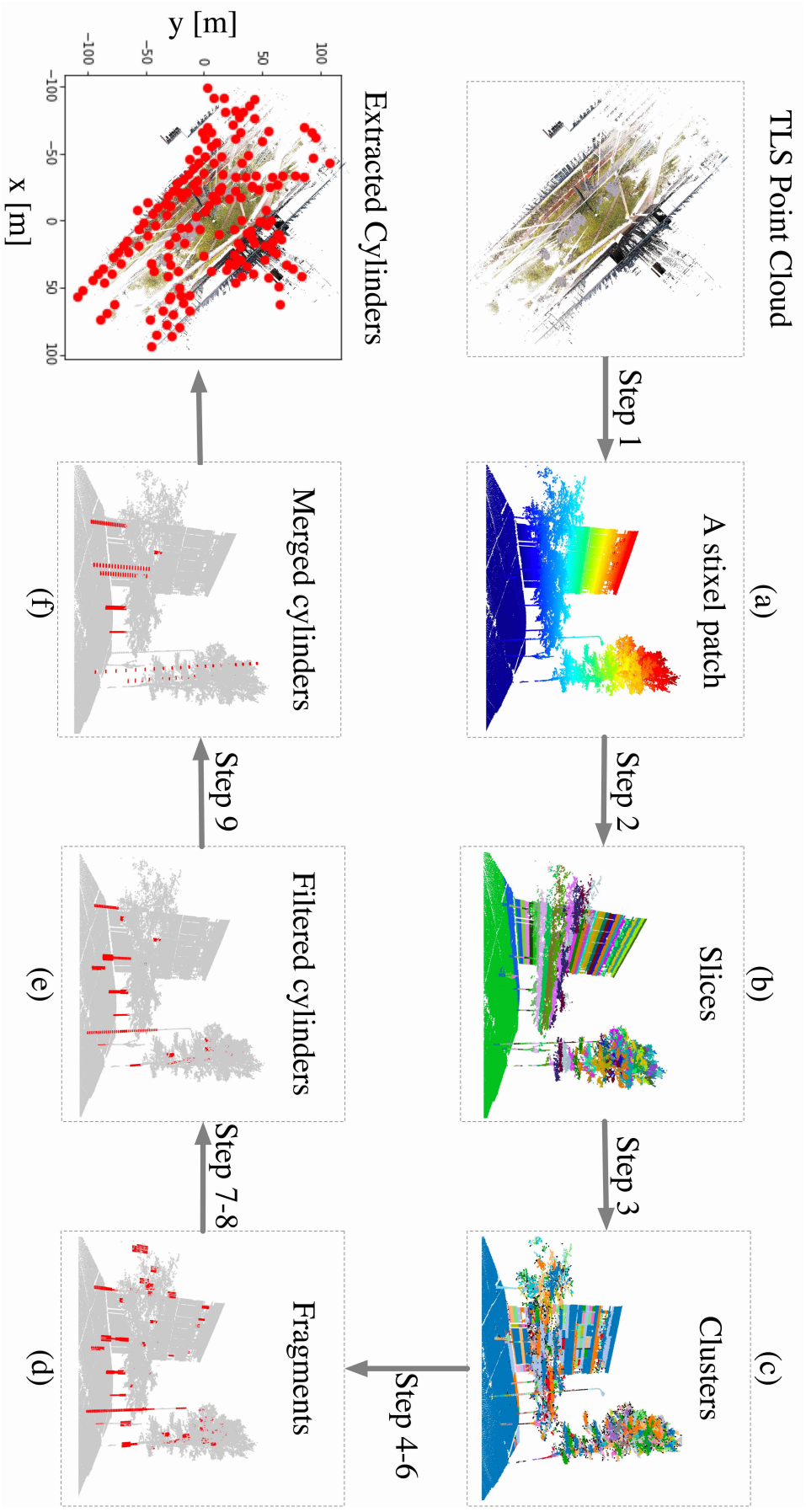


Figure 4.2: Workflow of cylinder extraction from **TLS** point clouds, including intermediate results

we apply a clustering algorithm to retrieve all candidates of part belonging to a cylinder or a facade. We choose a classical algorithm DBSCAN [20] with 0.2 m density parameter and a minimum number of points of 5. After clustering, some clusters are discovered with various sample size. A filter is applied based on the experience, that the number of points for parts belonging to a cylinder-like object is not large, and it is much larger for parts belonging to a facade. As a result, during extracting cylinder-like objects, only clusters containing between 5 to 100 points are saved, and 50 to 1.0×10^4 during line feature extraction. Finally, each slice is expressed as several filtered clusters, where $\mathbf{c}$ represents a cluster with superscript as index and subscript as corresponding slice, and N_{clu} is the number of clusters, as Eq. (4.3) shows, with clusters' visualization in Fig. 4.2 (c) (black dots represents noise):

$$\mathbf{s}_i^k = \left\{ \mathbf{c}_{ik}^1, \mathbf{c}_{ik}^2, ... \mathbf{c}_{ik}^{N_{clu}} \right\} \tag{4.3}$$

4. **Adjacent clusters connection**. At the end of the previous step we apply a coarse filter to remove clusters whose size is not close to the reality. Still there are many clusters which are not part of a facade or a cylinder-like object after the coarse filtering. We design a connection step for adjacent clusters. If two clusters have the similar size in adjacent slices, they will be connected for further processing, because different parts of a cylinder-like object or a facade should be closeby. Another purpose of this connection step is to remove ground points, as a cluster of ground points has a large number of points and usually distributes in a single slice. The algorithm is shown in Algorithm 1:

Algorithm 1: Cluster connection algorithm

input: clusters from the k^{th} slice $\mathbf{s}^k$ and the $k+1^{th}$ slice $\mathbf{s}^{k+1}$, threshold $T_{ratio} = 2.0$
$\mathbf{S}_{pair,k,k+1} = \{\}$
for $\mathbf{c}^i$ *in* $\mathbf{s}^k$ **do**
  for $\mathbf{c}^j$ *in* $\mathbf{s}^{k+1}$ **do**
    if $size(\mathbf{c}^i)/size(\mathbf{c}^j) \leq T_{ratio}$ ***or*** $size(\mathbf{c}^j)/size(\mathbf{c}^i) \leq T_{ratio}$ **then**
      $\mathbf{S}_{pair,k,k+1} += (\mathbf{c}^i, \mathbf{c}^j)$
    end
  end
end
output: a set of cluster pairs for the k^{th} slice and the $k+1^{th}$ slice $\mathbf{S}_{pair,k,k+1} = \{(\mathbf{c}_k^a, \mathbf{c}_{k+1}^b), ...\}$

5. **Connected pair concatenation**. Linking pairs of adjacent clusters, we concatenate them with another filter. We compare both centers of clusters in each pair. If both centers are close, we will concatenate these 2 clusters to form a new cluster. Particularly, during cylinder-like object extraction, we additionally compute the horizontal range, $\mathbf{max}\{|x_{max} - x_{min}|, |y_{max} - y_{min}|\}$ of new clusters and remove those with range larger than 1.0 m, to only keep cylinder-like objects with small radius. Algorithm 2 shows details of this process, where N_{new} is the number of new clusters:

6. **Cylinder/Line fragment estimation**. We finally get some reliable clusters which tend to be parts of a cylinder-like object or a facade. The estimation method assumes cylinders are vertical, then in xOy plane cylinder points fit a circle as Eq. (4.4) shows, and facade points fit a line as Eq. (4.5) shows:

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{4.4}$$

$$y = kx + b \tag{4.5}$$

where (x, y) are 2D coordinates and (x_0, y_0) is the center of a cylinder fragment, with radius r; k, b are slope, intercept of the line of a facade fragment, respectively.

Based on the least square estimation, as Eq. (4.6) and Eq. (4.7) show,

$$\underset{(\hat{x}_0, \hat{y}_0)}{\arg\min} \sum_{i=1}^{N} \left| R_i - \bar{R} \right|^2 \tag{4.6}$$

Algorithm 2: Cluster concatenation algorithm

input: a set of pairs for the k^{th} slice and the $k+1^{th}$ slice $\mathbf{S}_{pair,k,k+1}$, threshold $T_{center} = 0.3m$, $T_{range} = 1.0m$

```
s^{k,k+1} = {}
for c^i, c^j in S_{pair,k,k+1} do
    if |center(c^i) − center(c^j)|_2 ≤ T_center then
        c_{k,k+1} = concat(c^i, c^j)
        switch mode do
            case cylinder do
                if max{range(c^i), range(c^j)} ≤ T_range then
                    s^{k,k+1} += c_{k,k+1}
                else
                    S_{pair,k,k+1} −= (c^i, c^j)
                end
            case line do
                s^{k,k+1} += c_{k,k+1}
        end
    end
end
```

output: a set of new clusters for the k^{th} slice and the $k+1^{th}$ slice $\mathbf{s}^{k,k+1} = \left\{\mathbf{c}^1_{k,k+1}, \mathbf{c}^2_{k,k+1}, \ldots \mathbf{c}^{N_{new}}_{k,k+1}\right\}$

$$\underset{(\hat{k},\hat{b})}{\arg\min} \sum_{i=1}^{N} \left| y_i - \left(\hat{k}x_i + \hat{b}\right)\right|^2 \tag{4.7}$$

where $R_i = \sqrt{(x_i - \hat{x}_0)^2 + \left(y_i - \hat{y}_0\right)^2}$ and $\bar{R} = \frac{1}{N}\sum_{i=1}^{N}\sqrt{(x_i - \hat{x}_0)^2 + \left(y_i - \hat{y}_0\right)^2}$, we estimate the circle center for a cylinder fragment, as well as slope and intercept of facade line. We also compute the standard deviation (σ) and confidence interval from the estimation. We apply another filter based on these statistics, with $\sigma = 0.1$ m for cylinder estimation and $\sigma = 0.5$ m for line estimation, under 90% of 2σ confidence interval. Estimated radius should be smaller than 0.5 m as well. Information which is helpful for visualization is extracted as well e.g. the estimated height of cylinder fragments, the center of line fragments. As a result, we retrieve two types of dictionary for a cylinder fragment and a line fragment, respectively, as Eq. (4.8) and Eq. (4.9) show:

$$\mathbf{Dict}_{cylinder} = \{center_x, center_y, radius, height\} \tag{4.8}$$

$$\mathbf{Dict}_{line} = \{center_x, center_y, k, b\} \tag{4.9}$$

Finally $\mathbf{S}$ in Eq. (4.2) is expressed as Eq. (4.10):

$$\begin{aligned} \mathbf{S}^i_{cylinder} &= \left\{\mathbf{Dict}^1_{i,cylinder}, \mathbf{Dict}^2_{i,cylinder}, \ldots \mathbf{Dict}^{N_{cylinder}}_{i,cylinder}\right\} \quad or \\ \mathbf{S}^i_{line} &= \left\{\mathbf{Dict}^1_{i,line}, \mathbf{Dict}^2_{i,line}, \ldots \mathbf{Dict}^{N_{line}}_{i,line}\right\} \end{aligned} \tag{4.10}$$

where $N_{cylinder}$ and N_{line} are the number of extracted cylinder fragments and line fragments, respectively.

Fig. 4.2 (d) shows an example of cylinder fragment estimation, with extracted cylinder fragments shown in red. We notice some wrong fragments in the canopy of the left and the right tree. Pillars on the building behind are shown as well. Further processing is needed to decrease these wrong cylinders.

7. **Cylinder/Line fragments integration**. We design a integration process similar to Algorithm 2 to decrease the number of fake cylinder fragments caused by canopies, with 0.2 m center difference and 0.1 m radius difference as thresholds. Closeby line fragments are also integrated under thresholds of 0.02

slope difference and 0.4 m intercept difference. Then $\boldsymbol{S}$ in Eq. (4.1) is shown as Eq. (4.11):

$$\begin{aligned}\boldsymbol{S} = \Big\{\boldsymbol{S}\{\mathbf{Dict}_{cylinder}\},\, \boldsymbol{S}\{\mathbf{Dict}_{line}\}\Big\} = \\ \Big\{\mathbf{S}^1_{cylinder,integrated},\, \mathbf{S}^2_{cylinder,integrated},\, \dots \mathbf{S}^{N^c_{stx}}_{cylinder,integrated},\\ \mathbf{S}^1_{line,integrated},\, \mathbf{S}^2_{line,integrated},\, \dots \mathbf{S}^{N^l_{stx}}_{line,integrated}\Big\}\end{aligned} \tag{4.11}$$

8. **Pillars on buildings removal**. In order to remove pillars on buildings, we compute the minimum distance of each integrated cylinder to integrated lines. As Algorithm 3 shows, since facade lines have limited length, we set $T_{cpl} = 100$ to compute point-to-line distances for lines not far from a cylinder. If the minimum point-to-line distance of a cylinder is smaller than 0.1 m, we remove this cylinder from $\boldsymbol{S}$:

Algorithm 3: Cylinder on line removal algorithm

input: a set of dictionaries of integrated cylinders and integrated lines $\boldsymbol{S}$, thresholds $T_{cpl} = 100m$, $T_{p2l} = 0.1m$
for $cylinder,\, line$ ***in*** $\boldsymbol{S}\{\mathbf{Dict}_{cylinder}\},\, \boldsymbol{S}\{\mathbf{Dict}_{line}\}$ **do**
  if $\left|center\left(cylinder\right) - center\left(line\right)\right|_2 \le T_{cpl}$ & **dist**$\left(cylinder,\, line\right) \le T_{p2l}$ **then**
    $\boldsymbol{S} -= cylinder$
  end
end
output: a filtered set of dictionaries $\boldsymbol{S}$

Fig. 4.2 (e) gives a filtered result and we see an obvious decrease of number of fake cylinder fragments in the left and the right tree, The number of pillars on the building behind also decreases.

9. **Cylinder merging at a larger scale**. Finally, residuals are left to be processed. We merge all cylinders within a 5 m radius circle to make each merged cylinder more distinct and obvious, as Fig. 4.2 (f) shows. After this step, some fake cylinders, pillars on buildings or neighboring trees are merged together, so extracted cylinders are not the same as the ground truth. This merging process makes sure the density of extracted cylinders is closely uniform in different areas, which decreases the impact of certain areas where there are a lot of extracted cylinders. $\boldsymbol{S}$ in Eq. (4.11) is shown as Eq. (4.12):

$$\boldsymbol{S} = \Big\{\mathbf{S}^1_{cylinder,merged},\, \mathbf{S}^2_{cylinder,merged},\, \dots \mathbf{S}^{N^c_{stx}}_{cylinder,merged}\Big\} \tag{4.12}$$

All cylinders are extracted from a **TLS** scan or a typical **MLS** scene based on this 9-step feature extraction method. A **TLS** scan has hundreds of cylinders, shown in the last plot in Fig. 4.2. Extracted cylinders well represent the layout of **TLS** or **MLS** point clouds, covering the whole area.

4.4. Corresponding Pair Estimation between TLS and MLS Data

As Fig. 4.3 shows, the number of extracted cylinders from **MLS** scenes is in general much larger than that from **TLS** scans. Still we find similarities in the layout of cylinders extracted from each pair of **TLS** and **MLS** data. The next step is to find the correct corresponding pairs of **TLS** scans and typical **MLS** scenes. We clip **MLS** scenes after forming all corresponding pairs, to further decrease the size of **MLS** data. The workflow is also shown as the lower part in Fig. 4.4.

4.4.1. Probabilistic Estimation Framework

The Gaussian mixture model (**GMM**) is a widely used probabilistic model. It is used when data is close to being normally distributed and we do not know the accurate relationship between two datasets. In our research, the expression of **GMM** for extracted cylinders $\mathbf{t}$ in a **MLS** scene is shown in Eq. (4.13):

$$\begin{aligned} p(\mathbf{t}) &= \frac{1}{N_s}\sum_{i=1}^{N_s} P\left(s_i\right) p\left(\mathbf{t}|s_i\right) \\ p\left(\mathbf{t}|s_i\right) &= \frac{1}{\left(2\pi\sigma^2\right)^{(D/2)}} e^{-\frac{\|\mathbf{t}-s_i\|^2}{2\sigma^2}} \end{aligned} \tag{4.13}$$

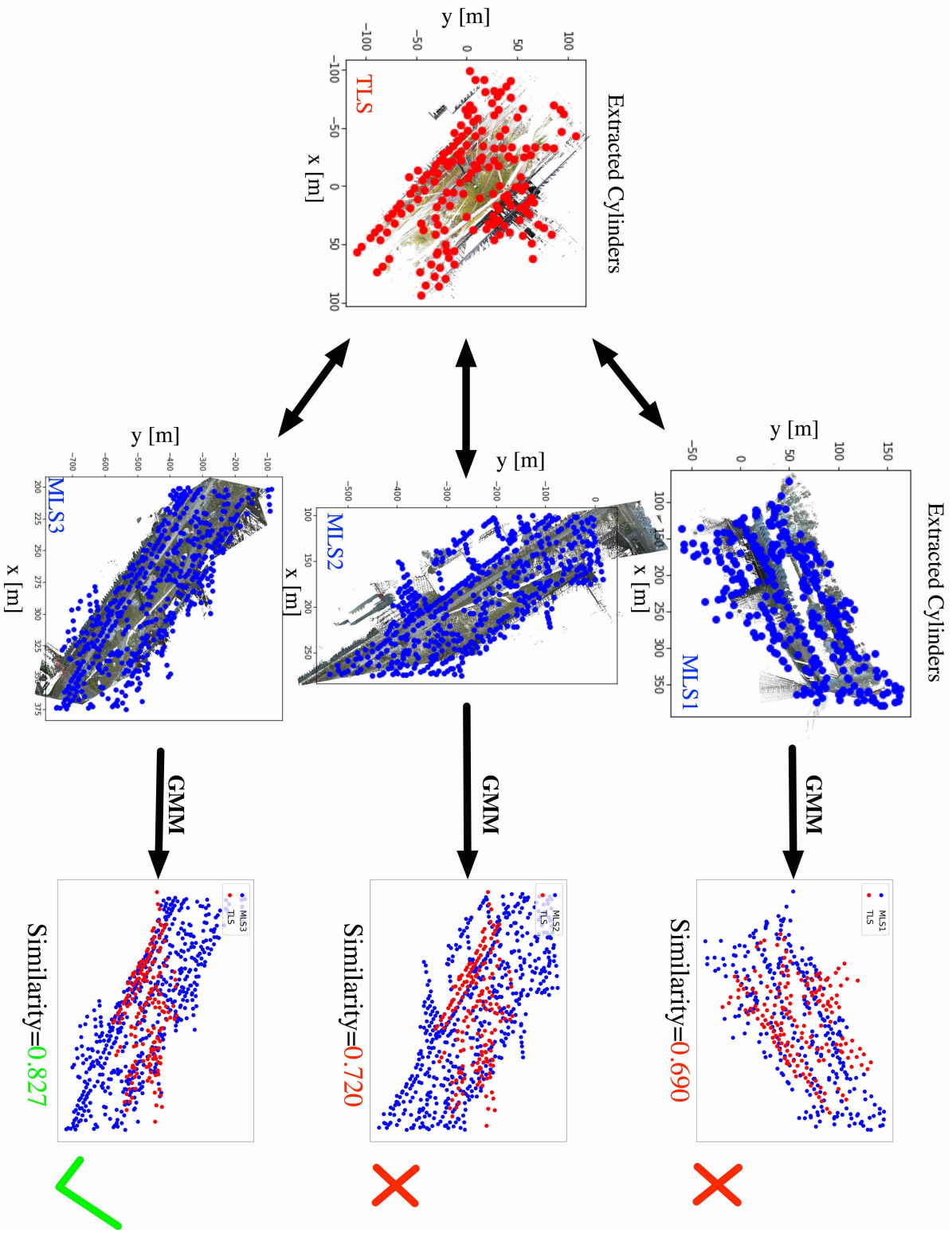


Figure 4.3: An aerial view of extracted cylinders in one **TLS** (red dots) scan and three **MLS** (blue dots) scenes, as well as similarities between **TLS** and **MLS** after applying Gaussian mixture model (**GMM**) probabilistic estimation

with weight function $P(s_i)$ for each cylinder s_i in a **TLS** scan and N_s the number of cylinders in this **TLS** scan; D, σ^2 are dimension and variance, respectively.

The expectation maximization (**EM**) algorithm proposed by [15] gives the objective function f_{EM} which enables us to estimate parameters based on **GMM**, shown in Eq. (4.14):

$$f_{EM} = -\sum_{j=1}^{N_t}\sum_{i=1}^{N_s} P_{old}\left(s_i|t_j\right)\log\left(P_{new}\left(s_i\right)p_{new}\left(t_j|s_i\right)\right) \tag{4.14}$$

with $P(s|t) = P(s)p(t|s)/p(t)$ and N_t the number of members in the template.

Specifically, coherent point drift (**CPD**) [54] applies **GMM** for registration to estimate a rotation matrix **R**, a translation vector **T**, shown in Eq. (4.15) and Eq. (4.16):

$$f_{EM}\left(\mathbf{R},\mathbf{T},\sigma\right) = \frac{1}{2\sigma^2}\sum_{j=1}^{N_t}\sum_{i=1}^{N_s} P_{old}\left(s_i|t_j\right)\left\|t_j - \mathbf{R}s_i - \mathbf{T}\right\|^2 + \frac{N_\mathbf{P}D}{2}\log\sigma^2 \tag{4.15}$$

$$P_{old}\left(s_i|t_j\right) = \frac{e^{-\left\|t_j-\mathbf{R}s_i-\mathbf{T}\right\|^2/2\sigma^2_{old}}}{\sum_{k=1}^{N_s} e^{-\left\|t_j-\mathbf{R}s_k-\mathbf{T}\right\|^2/2\sigma^2_{old}}} \tag{4.16}$$

where $N_\mathbf{P} = \sum_{j=1}^{N_t}\sum_{i=1}^{N_s} P_{old}\left(s_i|t_j\right)$.

Algorithm 4: Pseudo rigid registration algorithm

input: cylinders from source **TLS** point clouds and template **MLS** point clouds $\mathbf{s} = \{s_1,\ s_2,\ \ldots\ s_{N_s}\}$, $\mathbf{t} = \{t_1,\ t_2,\ \ldots\ t_{N_t}\}$ where $s, t = \mathbf{Dict}_{cylinder}\{center_x,\ center_y\}$ from the source or the template, thresholds $T_N = 30$, $T_\Delta = 10^{-4}$;

initialization: rotation matrix $\mathbf{R} = \mathbf{I}_2$, translation vector $\mathbf{T} = \mathbf{0}_2$, variance $\sigma^2 = \frac{1}{2N_tN_s}\|\mathbf{t}-\mathbf{s}\|^2$; $\mathbf{I}$, $\mathbf{0}$, $\mathbf{1}$ are identity matrix, zero vector and one vector respectively;

while iteration $\leq T_N$ & $\mathbf{\Delta} > T_\Delta$ **do**

$f_0 = f_{EM}\left(\mathbf{R},\mathbf{T},\sigma\right)$

E-step:

begin

$$\mathbf{P}(i,j) = \frac{e^{-\left\|t_j-\mathbf{R}s_i-\mathbf{T}\right\|^2/2\sigma^2}}{\sum_{k=1}^{N_s} e^{-\left\|t_j-\mathbf{R}s_k-\mathbf{T}\right\|^2/2\sigma^2}}$$

end

M-step:

begin

$N_\mathbf{P} = \mathbf{1}^T\mathbf{P}\mathbf{1}$, $\mu_t = \frac{1}{N_\mathbf{P}}\mathbf{t}^T\mathbf{P}^T\mathbf{1}$, $\mu_s = \frac{1}{N_\mathbf{P}}\mathbf{s}^T\mathbf{P}\mathbf{1}$

$\hat{\mathbf{t}} = \mathbf{t} - \mathbf{1}\mu_t^T$, $\hat{\mathbf{s}} = \mathbf{s} - \mathbf{1}\mu_s^T$

$\mathbf{H} = \hat{\mathbf{t}}^T\mathbf{P}^T\hat{\mathbf{s}}$,;

$SVD(\mathbf{H}) = \mathbf{U}\mathbf{S}\mathbf{V}^T$

$\mathbf{R} = \mathbf{U}\mathbf{C}\mathbf{V}^T$, where $\mathbf{C} = diag\left(1,1,\ldots det\left(\mathbf{U}\mathbf{V}^T\right)\right)$

$\mathbf{T} = \mu_t - \mathbf{R}\mu_s$

$\sigma^2 = \frac{1}{2N_\mathbf{P}}\left(tr\left(\hat{\mathbf{t}}^T diag\left(\mathbf{P}^T\mathbf{1}\right)\mathbf{t}\right) - tr\left(\mathbf{H}^T\mathbf{R}\right)\right)$

end

$f = f_{EM}\left(\mathbf{R},\mathbf{T},\sigma\right)$

$\mathbf{\Delta} = \left|f_0 - f\right|$

iteration$+ = 1$

end

$$\mathbf{R}_3 = \begin{bmatrix}\mathbf{R} & 0\\ 0 & 1\end{bmatrix},\ \mathbf{T}_3 = \begin{bmatrix}\mathbf{T} & 0\end{bmatrix}$$

output: $\mathbf{R}$, $\mathbf{T}$, $\mathbf{R}_3$, $\mathbf{T}_3$

4.4.2. GMM for Place Recognition

We choose the probabilistic estimation based on **GMM** to realize place recognition for two main reasons:

1. Different from traditional registration methods based on fine feature descriptors, we design a descriptor to roughly extract cylinder-like objects from given point clouds. Probabilistic estimation is suitable for our purpose: we only need to find a maximum likelihood solution to estimate correspondences between template **MLS** scenes and source **TLS** scans.

2. The **GMM** probabilistic estimation is time consuming when the sample size is large, more than 100 seconds to process 1 million points. But the number of our extracted cylinders is small, only several hundreds of cylinders, so it is an efficient way to quickly estimate corresponding pairs between **TLS** scans and **MLS** scenes.

The **GMM** probabilistic estimation based on extracted cylinders is 2D estimation, which saves time. The input is $\boldsymbol{S}\{\mathbf{Dict}_{cylinder}\{center_x, center_y\}\}$ from **TLS** and **MLS** data. The output is a rotation matrix **R** and a translation vector **T**, respectively. Based on Eq. (4.15), the algorithm is shown in Algorithm 4, which we call pseudo rigid registration.

4.4.3. Similarity & Mean Distance Definition

We apply the 2D pseudo rotation matrix **R** and translation vector **T** for cylinders from the source to get the predicted template $\mathbf{t}' = \mathbf{Rs} + \mathbf{T}$. For a pair $(\mathbf{t}', \mathbf{t})$, we define a metric, similarity ρ which describes the degree of overlap between the predicted template and the true template, as Eq. (4.17) shows, with threshold $T_{dis} = 8$ m in our research:

$$\rho = \frac{1}{N_s} \sum_{i=1}^{N_s} \left(\min \left(\mathbf{d}_{t_i', \mathbf{t}} \right) \leq T_{dis} \right) \tag{4.17}$$

where $d_{t_i', t_j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

In addition, we also define a mean distance μ which helps us to remove some outliers, as Eq. (4.18) shows:

$$\mu = \frac{1}{N_s} \sum_{i=1}^{N_s} \min \left(\mathbf{d}_{t_i', \mathbf{t}} \right) \tag{4.18}$$

An example of similarity computation using **GMM** probabilistic estimation is shown in Fig. 4.3.

4.4.4. Decision Making Strategy

For each pair of **TLS** scans and **MLS** scenes, we always compute a similarity and a mean distance. We need to decide each corresponding pair step by step, under following decision making strategy:

1. **Metric (similarity & mean distance) computation**. For each pair of **TLS** scans and **MLS** scenes, we compute the similarity and the mean distance based on Eq. (4.17) and Eq. (4.18), to get $\boldsymbol{\rho}(i, j)$ and $\boldsymbol{\mu}(i, j)$ between the i^{th} **TLS** scan and the j^{th} **MLS** scene;

2. **Outlier removal**. We set a 0.6 threshold for computed similarities and 10 m for mean distances, which means that if the similarity is smaller than 0.6 or the mean distance is larger than 10 m, the pair will be removed as non-matching, as the step **outlier removal** in Algorithm 5 shows;

3. **The most dominant solution search**. If we have N_{TLS} **TLS** scans and N_{MLS} **MLS** scenes, then for each **TLS** scan we have N_{MLS} candidates. Before selecting the candidate with the highest similarity, we also compute the difference between the highest similarity and the second highest similarity for each **TLS** scan. This difference Δ we call the dominance. For example, for the i^{th} **TLS** scan,

$$\Delta_i = \max_{j \in Idx_{MLS}} \boldsymbol{\rho}(i, j) - \max_{j \in Idx_{MLS} \,\&\, j \neq j_{max,i}} \boldsymbol{\rho}(i, j) \tag{4.19}$$

   where $Idx_{MLS} = \{1, 2, ... N_{MLS}\}$ and $j_{max,i} = \arg\max_{j \in Idx_{MLS}} \boldsymbol{\rho}(i, j)$.

   We match the **TLS** scan with the largest Δ with its correspondence, as the step **dominance search** in Algorithm 5 shows;

4. **Inactivation**. We need to label other candidates as non-matching once a matching pair is decided, as the step **inactivation** in Algorithm 5 shows;

Repeat above step 3 and 4 until all possible corresponding pairs are estimated.

Algorithm 5: Corresponding pair estimation based on computed similarities and mean distances

input: computed similarities $\boldsymbol{\rho}$ and mean distances $\boldsymbol{\mu}$ between N_{TLS} **TLS** scans and N_{MLS} **MLS** scenes; thresholds $T_\rho = 0.6$, $T_\mu = 10$

initialization: a decision matrix $\mathbf{X}$, where we use x, 0, 1 to represent an undecided pair, a not-matching pair and a matching pair;

outlier removal:

$\mathbf{X}(i,j) = 0$ if $\boldsymbol{\rho}(i,j) < T_\rho$ or $\boldsymbol{\mu}(i,j) > T_\mu$

while $\exists\ x\ in\ \mathbf{X}$ **do**

  dominance search:

  begin

    $\boldsymbol{\Delta} = (\Delta_1, \Delta_2, ...\Delta_i)$, $1 \le i \le N_{TLS}$, $\nexists$ j where $\mathbf{X}(i,j) = 1$

    $\hat{a} = \arg\max_{\Delta_a \in \boldsymbol{\Delta}} \Delta_a$

    $\hat{b} = \arg\max_{1 \le b \le N_{MLS}} \boldsymbol{\rho}(\hat{a}, b)$

    $\mathbf{X}(\hat{a}, \hat{b}) = 1$

  end

  inactivation:

  begin

    $\mathbf{X}(\hat{a}, b) = 0$, $1 \le b \le N_{MLS}$ & $b \ne \hat{b}$

    $\mathbf{X}(a, \hat{b}) = 0$, $1 \le a \le N_{TLS}$ & $a \ne \hat{a}$

  end

end

output: a decision matrix $\mathbf{X}$ with 0 or 1 only

4.4.5. Range Tuning of MLS Scenes

There are small offsets between **TLS** scans and **MLS** scenes w.r.t. z axis. We assume there is no offset in order to extend 2D $\mathbf{R}$ and $\mathbf{T}$ extracted from cylinders to 3D for point clouds, shown as $\mathbf{R}_3$, $\mathbf{T}_3$ in Algorithm 4. For a corresponding pair $\left(\boldsymbol{S}^i_{source}, \boldsymbol{S}^j_{template}\right)$, we transform source **TLS** point clouds $\boldsymbol{S}^i_{source}$ with $\mathbf{R}_3$, $\mathbf{T}_3$, and clip the corresponding template **MLS** point clouds $\boldsymbol{S}^j_{template}$ based on the range of transformed $\boldsymbol{S}^i_{source}$, as it shows in Eq. (4.20):

$$
\begin{aligned}
&\boldsymbol{S}'^{\,i} = \mathbf{R}_3 \boldsymbol{S}^i_{source} + \mathbf{T}_3 \\
&\boldsymbol{S}'^{\,j}_{template} = \boldsymbol{S}^j_{template}\{t\},\ (t_x, t_y) \in \mathbf{range}_{2D}\left(\boldsymbol{S}'^{\,i}\right) \\
&\mathbf{range}_{2D}\left(\boldsymbol{S}'^{\,i}\right) = \left\{\left[x_{min} - \Delta_{range},\ x_{max} + \Delta_{range}\right],\ \left[y_{min} - \Delta_{range},\ y_{max} + \Delta_{range}\right]\right\},\ (x,y) \in \boldsymbol{S}'^{\,i}
\end{aligned}
\tag{4.20}
$$

As the pair $\left(\boldsymbol{S}'^{\,i}_{source}, \boldsymbol{S}^j_{template}\right)$ is not completely overlapped, $\Delta_{range} = 10.0m$ is a term that allows the situation where there is offset between a **TLS** scan and corresponding **MLS** scene.

The range tuning process makes sure that data from **TLS** and **MLS** have similar sample size. We realize place recognition finally, with the output $\left(\boldsymbol{S}'^{\,i}_{source}, \boldsymbol{S}'^{\,j}_{template}\right)$, and we input all pairs as our input of the next main step – pose refinement.

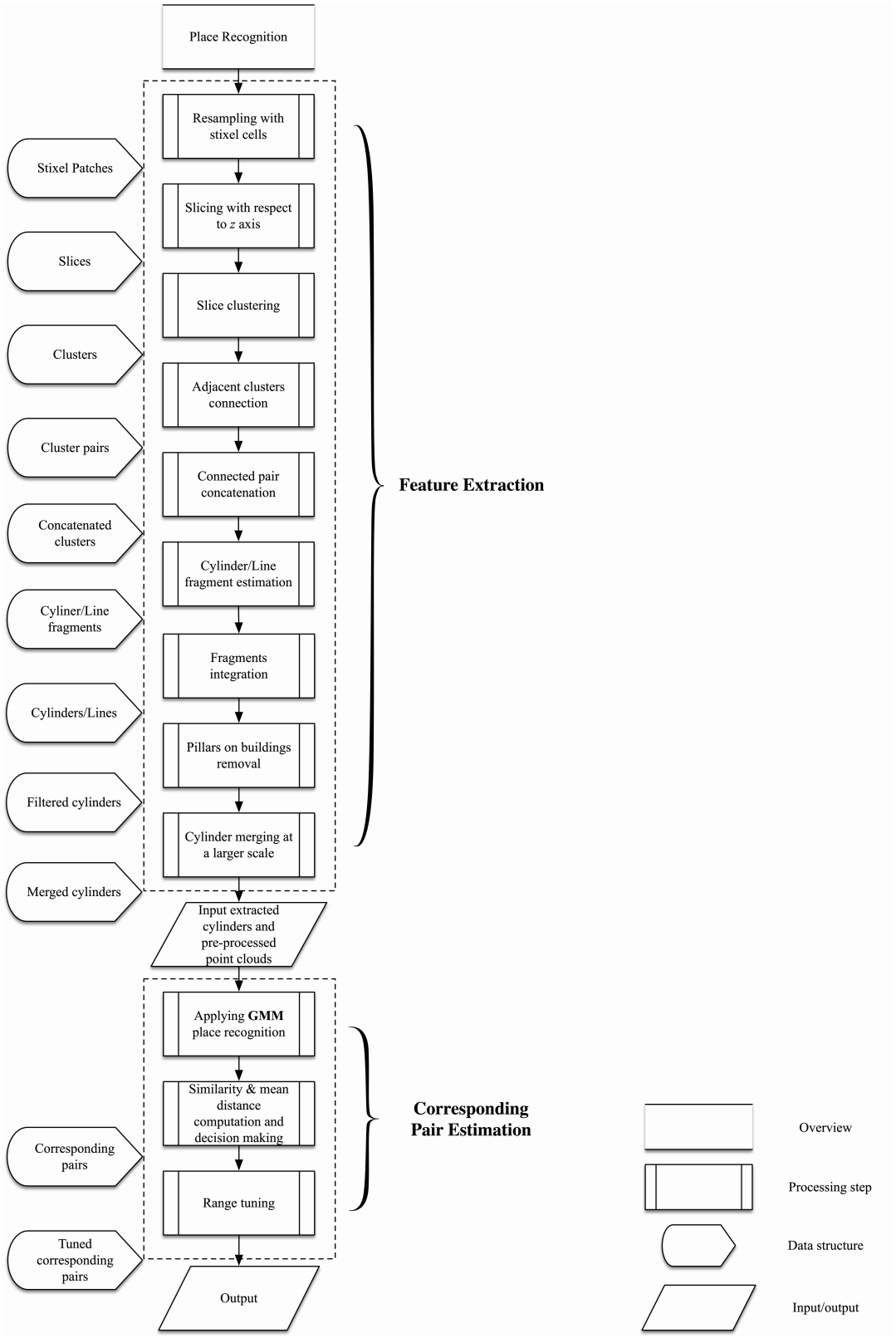


Figure 4.4: Workflow of feature extraction from point clouds, and corresponding pair estimation, together with flowchart of different data structures

5

Method for Pose Refinement

This chapter firstly reviews the pose refinement problem in Section 5.1. Then the design of our neural network for pose refinement which consists of 3 main blocks is introduced in Section 5.2. Finally the loss function design during training and the global refinement of prediction after training are given in Section 5.3 and Section 5.4, respectively.

5.1. Introduction

Based on the result of place recognition, we get two point clouds which are close to each other from terrestrial laser scanning (**TLS**) and mobile laser scanning (**MLS**), respectively. In Section 1.5 we define the pose refinement as a process of fine registration which tunes the place recognition result by a neural network. Section 2.4 summarizes some typical registration networks and shows their limitations. In general, we design a registration neural network that consists of several multi layer perceptrons (**MLP**s) and a max pooling layer for learned feature extraction, and some fully connected layers to estimate 6 pose parameters in a transformation matrix, shown as $(\theta_x,\ \theta_y,\ \theta_z,\ \delta_x,\ \delta_y,\ \delta_z)$ in Eq. (2.1), from learned features.

5.2. Registration Neural Network Structure at a Large Scale

In the data preprocessing step before pose refinement introduced in Chapter 3, a pair of corresponding point clouds from results of place recognition are divided into patches of 256 points each as the input of pose refinement. Patches are chosen as the input due to several differences from the input of most point-based neural networks for registration, to enable a registration neural network in an urban area:

1. Most registration neural networks use a dataset of small simple objects e.g. tables, chairs, etc. The dataset is usually well labeled in different categories e.g. ModelNet40, a dataset containing 40 object categories. In an urban area, it is difficult to label these small simple objects because of millions of points as input and the complicated environment. In place recognition, we coarsely extract some cylinder-like objects e.g. street lights, but they are filtered and merged results, not reliable to be taken as the input to a neural network. Large objects such as buildings, cars, trees are common components of an urban environment.

2. An urban area has objects with various ranges and numbers of points. For example, a car has thousands of points in a 5 m × 2 m × 1.5 m space, and a building has more than 10 thousands of points in a 50 m × 30 m × 20 m space. As a result, a single batch in a neural network cannot well represent all types of object in an urban environment, under the precondition of the same shape for each batch.

3. There are a lot of planar parts in an urban area e.g. grasslands, facades. The information of plane normals is helpful, but these planes also cause similarity ambiguities and ambiguous changes because of temporal change.

As a result, we do not label our **TLS** and corresponding **MLS** data, but resample point clouds using a voxel size e.g. 5 m × 5 m × 3 m to generate patches. A patch randomly represents a part of **TLS** or **MLS** point clouds. A batch in our neural network consists of several patches instead of one patch because we want to minimize

the impact when the patch in the source is obviously different from the patch in the template due to temporal change or occlusions during data acquisition. Fig. 5.1 shows four example patches distributed in different parts of a building, and there some differences between a source patch from **TLS** data and corresponding template patch from **MLS** data.

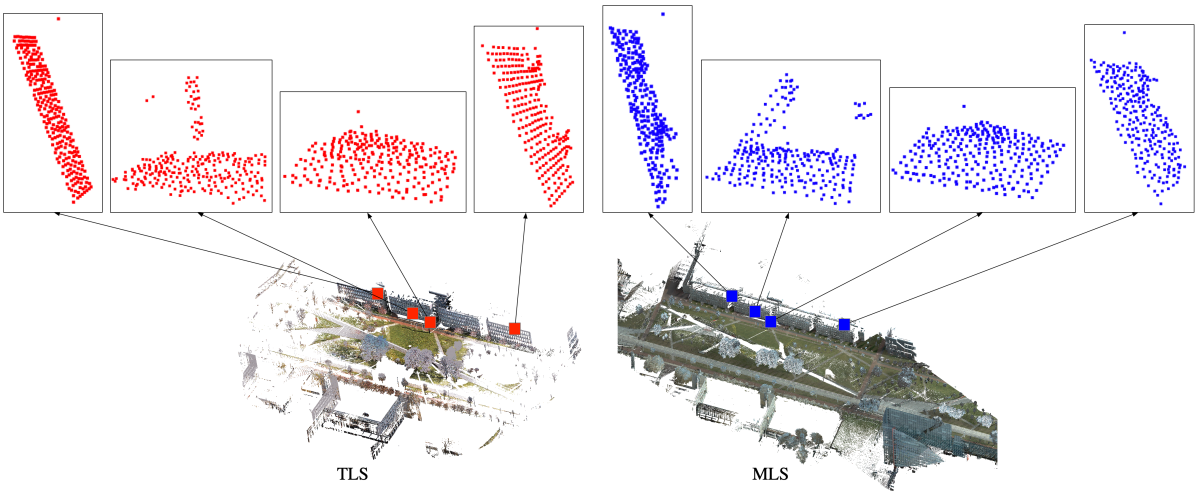


Figure 5.1: Four voxel patches of voxel size (5m, 5m, 3m) from **TLS** (left) and their correspondences in **MLS** (right).

5.2.1. Patch Based Neural Network for Pose Refinement

In order to realize pose refinement in our research, we design a patch based network, shown in Fig. 5.2. The input consists of two patch lists from source and template, respectively, shown as Eq. (5.1):

$$\begin{aligned} \boldsymbol{P}_{source} &= \left[P^1_{source}, P^2_{source}, \dots P^{N_{nn}}_{source}\right] \\ \boldsymbol{P}_{template} &= \left[P^1_{template}, P^2_{template}, \dots P^{N_{nn}}_{template}\right] \end{aligned} \tag{5.1}$$

where P represents urban patches of shape (B, N_{patch}, D, C) (batch size, number of points for one patch, dimension for each point, number of channels), and the length of two lists is the same N_{nn} during training. When we only input 3D coordinates information, then $D = 3$, $C = 1$. The selection of N_{patch} depends on the size of each patch. A large patch needs more points to well describe it. We use $N_{patch} = 256$ in combination with (5 m, 5 m, 3 m) voxel cell in our research.

As Fig. 5.2 shows, both the source list and the template list go through three main blocks step by step: block **A** – the feature extraction block (**FEB**), block **B** – the patch correspondence search block (**CSB**) and block **C** – the pose estimation (registration) block (**PEB**).

5.2.2. Block A: the Feature Extraction Block (FEB)

Different from the general point-based network, we have two **FEB**s in block **A**, **FEB A** for block **C** – **PEB** and **FEB B** for block **B** – **CSB**, respectively, shown in Fig. 5.2.

Invariant Patch Feature Extraction in FEB B

In PointNet, to ensure the accuracy of classification and segmentation, a T-Net is applied to make input point clouds invariant to transformation [61]. In our case, the problem of patch correspondence search is more like a correspondence recognition problem, which is much simpler than classification and segmentation. So we design a network without T-Net to make it less complicated. There are four main steps as following:

1. **Input patch recentering**. Firstly, we apply recentering for each patch in the source list and the template list to make it translation invariant, then the input list is shown as Eq. (5.2)

$$\begin{aligned} \boldsymbol{P}^{Re}_{source} &= \left[P^1_{source,Re}, P^2_{source,Re}, \dots P^{N_{nn}}_{source,Re}\right] \\ \boldsymbol{P}^{Re}_{template} &= \left[P^1_{template,Re}, P^2_{template,Re}, \dots P^{N_{nn}}_{template,Re}\right] \end{aligned} \tag{5.2}$$

   where $P^i_{Re} = P^i - center\left(P^i\right)$;

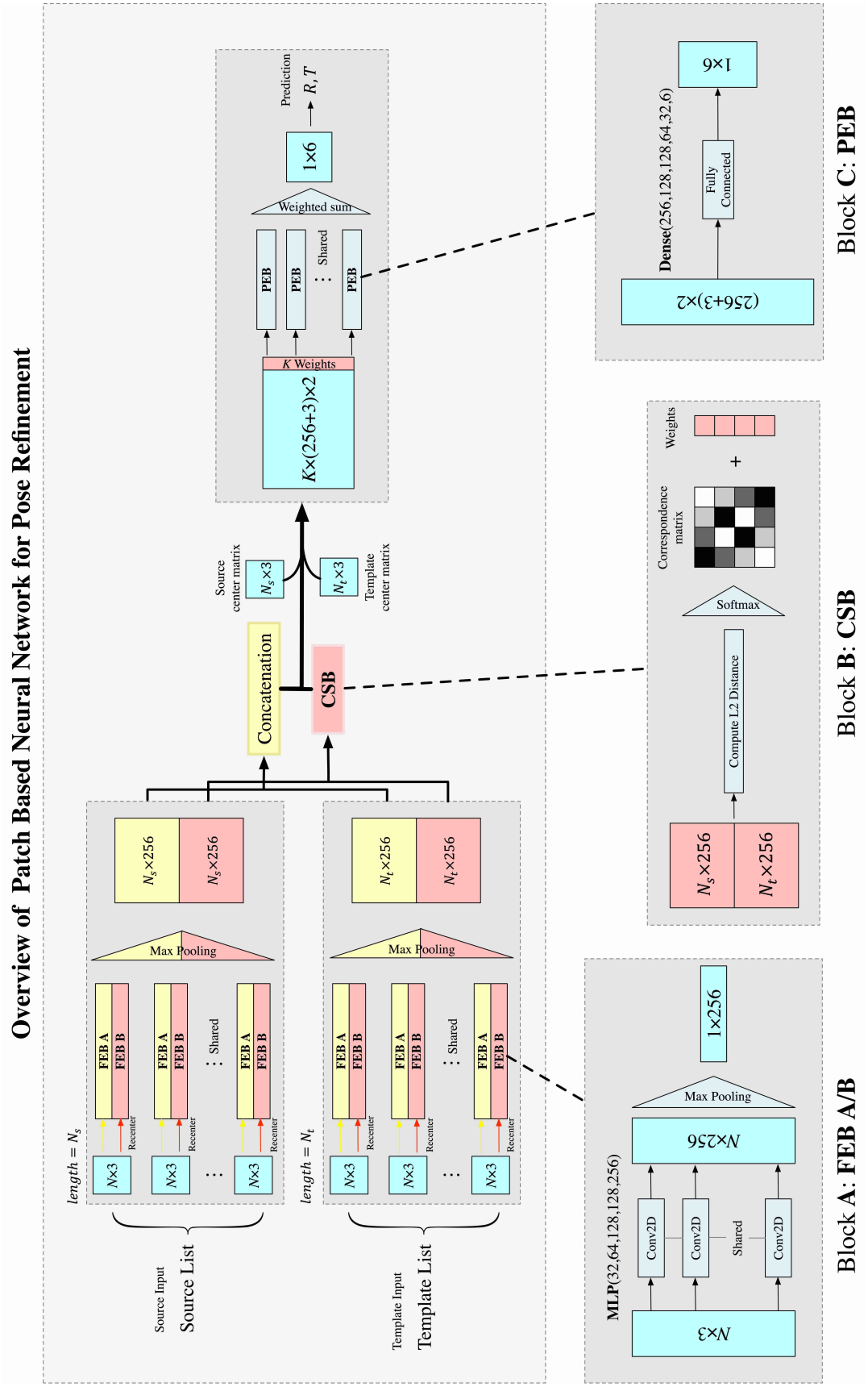


Figure 5.2: Overview of the patch based neural network for pose refinement. In general, each patch from the source list and the template list firstly goes through block **A** to extract two types of features, invariant features shown in red and general features shown in yellow. Then different features of different patches are concatenated to form a feature matrix. Red features are applied to estimate corresponding pairs between source patches and template patches in block **B**, as well as a set of weights. Yellow features are concatenated based on the result of block **B**, together with center information. For each pair, block **C** predicts a set of 6 pose parameters. The prediction is extracted by a weighted sum over all pairs.

2. **Point feature extraction with MLPs**. A five-layer **MLP** with dimension (32, 64, 128, 128, 256) is applied for each recentered patch in the source list and the template list, to summarize 3D coordinate information into 256 features for each point;

3. **Patch feature extraction with max pooling**. For each feature space within a recentered patch, we apply a max pooling process to keep the feature with the largest value, to form 256 features for this patch as patch features;

4. **Feature matrix output**. After **MLP**s and a max pooling process, we concatenate all features of patches in the source list or the template list. When both the length of the source list N_s and the length of the template list N_t are equal to N_{nn} for each batch, $N_{nn} = 10$ in our case, the output will be a $(10, 256)$ matrix, shown as the red part in **FEB** in Fig. 5.2.

We realize translation invariance by recentering. As for rotation invariance for extracted features, we want to realize it in a loose way based on the output of feature matrix, by defining a loss and tuning it during training, rather than a strict transformation matrix which is inserted inside the network as parameters.

General Patch Feature Extraction in FEB A

We need to preserve geometric information during general patch feature extraction, so the input is the same as the original, as Eq. (5.1) shows. Then we apply same steps, step 2 to step 4 during invariant patch feature extraction, to output the other $(10, 256)$ feature matrix, shown as the yellow part in **FEB** in Fig. 5.2.

5.2.3. Block B: the Patch Correspondence Search Block (CSB)

After block **A** – **FEB**, we have 10 patches with 256 invariant or general features each in the source list and the template list, respectively. The main purpose of block **B** – **CSB** is to form a patch corresponding pair $\left(P^i_{source}, P^j_{template}\right)$ between the source list and the template list based on invariant patch features. Then we concatenate extracted general patch features from the source patch and corresponding template patch, as well as patch center information, to estimate pose parameters by block **C** – **PEB**.

We design a method to search corresponding template patches for source patches as following steps, with an example shown in Fig. 5.4.

1. **Feature distance matrix computed by L_2 distance**. We choose L_2 distance to compute feature distance matrix. The L_2 distance between one patch from a source list and one patch from a template list is defined as Eq.(5.3):

$$L_2\left(f_{source}, f_{template}\right) = \sqrt{\sum_{i=1}^{L_{cor}} \left(f^i_{source} - f^i_{template}\right)^2} \tag{5.3}$$

   where f is a feature vector;

   From all patches from the source list and the template list, respectively, we compute the feature distance matrix $\mathbf{D}(a,b) = L_2(a,b)$ between the a^{th} source invariant feature vector and the b^{th} template invariant feature vector. A small L_2 distance indicates high similarity between source and template patch, but a distance close to 0 is hard to achieve in reality because a source patch is not the same as corresponding template patch. Anyway the feature distance of true corresponding pairs will be lower compared to distances between non-corresponding pairs.

   The reason why we select L_2 distance for describing similarity between patches from the source and the template is that we cannot predict the range of extracted invariant patch features, which depends on the hyperparameter selection for the whole neural network. Ideally, computed L_2 feature distance should be low for true patch corresponding pairs, but in reality it varies a lot because of different types of errors. For example, the source patch is obviously different from the template patch, as Fig. 5.3 shows, where the source patch is a canopy but the template patch is a trunk with ground points. As a computed L_2 distance varies from 0 to large numbers, it is flexible to tune the network by giving a threshold, based on the validation dataset. A large range also makes sure the extracted features are distinct enough.

2. **Correspondence matrix computed by Softmax function**. The feature distance matrix is a relative matrix for describing similarity. The **Softmax** function is a good way to transform these relative values to

probabilities, as shown in Eq. (5.4):

$$\sigma(i,j) = \frac{e^{L^i_{2,max} - L_2(i,j)}}{\sum_{j=1}^{N_{nn}} \left(e^{L^i_{2,max} - L_2(i,j)} \right)} \quad \textbf{if} \min_{1 \le j \le N_{nn}} L_2(i,j) \le T_{L_2}, \quad \textbf{else}$$
$$\sigma(i,j) = 0 \quad 1 \le i \le N_{nn},\ 1 \le j \le N_{nn} \tag{5.4}$$

where $L^i_{2,max} = \max_{1 \le j \le N_{nn}} L_2(i,j)$ and T_{L_2} is a threshold for feature distance. Correspondence matrix $\mathbf{C}(a,b) = \sigma(a,b)$, shown as the grey matrix in **CSB** in Fig. 5.2.

Ideally, a true correspondence matrix $\mathbf{C}_{true}$ has the following attributes:

(a) $\mathbf{C}_{true}(i,j) = 0 \; or \; 1$

(b) $\sum_i \mathbf{C}_{true}(i,j) = \sum_j \mathbf{C}_{true}(i,j) = 1$

In experiments, for **CSB** we will only choose corresponding pairs with $\mathbf{C}(i,j) \ge r$ where r is the threshold of probability calculated by Eq. (5.4). T_{L_2} will be decided by a validation set after training.

3. **Corresponding pair weight**. After retrieving the correspondence matrix, we will get a list of corresponding pairs by selecting the template patch with the largest possibility for a source patch, as Eq. (5.5) shows

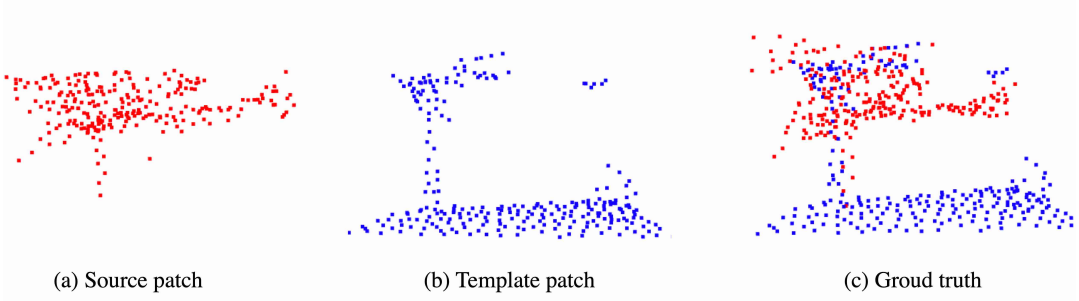
$$\mathbf{p} = \left[cor_1,\ cor_2,\ \ldots\ cor_{N_{nn}} \right] \tag{5.5}$$

where $cor_i = (i,j)$ for the i^{th} patch in the source list and its most possible corresponding patch, the j^{th} patch in the template list, and corresponding L_2 feature distance vector, as Eq. (5.6) shows:

$$\mathbf{d}_{cor} = \left(L_2(cor_1),\ L_2(cor_2),\ \ldots\ L_2(cor_{N_{nn}}) \right) \tag{5.6}$$

We apply **Softmax** for vector $\mathbf{d}_{cor}$ and use its output $\boldsymbol{\omega}_{cor}$ as the weight for each corresponding pair, shown as the red vector in **CSB** in Fig. 5.2. Noticeably, this weight is only used during training to make most similar patch corresponding pairs more dominant, as well as to make the training process efficient. For example, if a patch corresponding pair has a low L_2 feature distance, it will dominate the output a lot. If this pair with low L_2 feature distance has bad quality of the output, then parameters will be tuned by generating a larger L_2 feature distance to assign this pair a smaller weight. During validation and testing, the weight is equal for each corresponding pair since we do not know if each pair has good quality.

4. **Feature concatenation**. Finally, we concatenate general features for each patch corresponding pair based on correspondence matrix, and include patch centers from the source and the template as extra geometric information, to form a concatenated feature vector with length $(256+3) \times 2$, shown as the blue part in **PEB** in Fig. 5.2. The concatenated feature matrix with shape $(K, 2 \times (256+3))$ where K is the number of corresponding pairs, will be applied in step **C** – **PEB**.

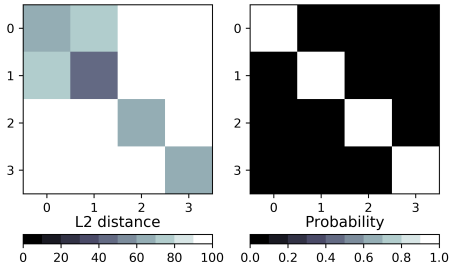


(a) Source patch (b) Template patch (c) Groud truth

Figure 5.3: An example of the obvious difference between a source patch (a) and a template patch (b) together with the ground truth (c), with computed feature distance $L_2 = 89.69$

Row: TLS patch; Col: MLS patch;				
	63.48	76.02	100.28	99.75
	67.01	43.92	102.38	100.74
	124.12	112.74	62.90	109.54
	97.44	92.64	93.11	64.77

(a)



(b)

Figure 5.4: (a) Computed feature distances for 4 example patches in Fig. 5.1 (b) Corresponding feature distance matrix & correspondence matrix

5.2.4. Block C: the Pose Estimation (Registration) Block (PEB)

After block **A** – **FEB** and block **B** – **CSB**, we will get a 10-weight vector $\boldsymbol{\omega}$ and a concatenated feature matrix with shape $(10, 2 \times (256+3))$, if all patches in the source list have their correspondence in the template list. For each corresponding pair, we apply six fully connected layers, with dimension (256, 128, 128, 64, 32, 6), to summarize these $2 \times (256+3)$ features to final 6 pose parameters $\boldsymbol{\xi}$. The final pose parameters are computed based on Eq. (5.7):

$$\boldsymbol{\xi} = \sum_{i=1}^{N_{nn}} \omega_i \boldsymbol{\xi_i} \tag{5.7}$$

The predicted rotation matrix and translation vector are computed based on pose parameters and Eq. (2.1).

5.3. Training Process & Loss Function

In general, after designing a neural network, there are three preparations before applying training:

1. **Dataset preparation**. A dataset in a neural network consists of a training set, a validation holdout set and a test set. Particularly, both the training set and the validation holdout set cover the same area in our research. But the validation holdout set does not participate in training and is only applied to tune the pre-trained network. The test set comes from a new area. For example, the validation holdout set is generated by randomly selecting one fifth of patches from several **TLS** scans and their corresponding **MLS** scenes. The test set is patches from another new **TLS** scan with its corresponding **MLS** scene.

2. **Hyperparameter selection**. Hyperparameters in a neural network are factors that we can change or pre-define before training, e.g. the number of **MLP**s, learning rate, etc. Hyperparameters are fixed during training. Proper hyperparameters are selected based on results from different pre-trained neural networks under different sets of hyperparameters.

3. **Loss function definition**. The loss function is the most important component for training. It describes the difference between the predicted result from a trained neural network and the ground truth. Different loss functions applied in combination with the same network architecture may result in completely different learners, as the performance of a neural network is tuned by the gradient of loss function in each epoch.

In Chapter 2 several types of loss function in registration neural network are introduced. Total loss of our network consists of two parts: labeling loss and pose loss. Labeling loss compares the predicted correspondence matrix with the true correspondence matrix in block **B** – **CSB**. Pose loss describes the difference

between the predicted pose parameters and the ground truth. There are two types of total loss and four types of pose loss that are introduced in our research.

5.3.1. Labeling Loss

A cross entropy loss is usually used in classification. It is computed by comparing the probability between each predicted result and each class, to minimize the difference between the prediction and the true class and maximize differences between the prediction and false classes. Labeling loss is similar to cross entropy loss with **Softmax**. By minimizing it we ensure the probability of true corresponding pairs in the correspondence matrix is close to 1. Different from cross entropy loss where probabilities of different classes are included, we only compare the probability of each true corresponding pair with the prediction. It ensures extracted invariant features are distinct enough. The labeling loss is defined as Eq. (5.8):

$$\mathbf{Loss}_{cor} = \sum_{i=1}^{N_{nn}} \left| \mathbf{C}_{pred}(i, i_{cor}) - \mathbf{C}_{true}(i, i_{cor}) \right| \tag{5.8}$$

where $i_{cor} = \arg_{1 \le j \le N_{nn}} \mathbf{C}_{true}(i, j) = 1$, C_{pred} and C_{true} are predicted and true correspondence matrix, respectively.

5.3.2. Pose Loss

Pose parameters contain six elements, three for a rotation matrix and three for a translation vector, as it shows in Eq. (2.1). The unit of rotation is in radian(s) and the unit of translation is in meter(s), and are unit incompatible. One choice is to compute the loss for rotation and translation separately, the other choice is to use one term to compute a combined loss for both rotation and translation.

Rotation Loss

The rotation loss is defined as the root mean square error (**RMSE**) between the predicted rotation matrix $\mathbf{R}_{pred}$ and the truth $\mathbf{R}_{true}$. Instead we compute the **RMSE** between $\left(\mathbf{R}_{pred}\right)^{-1}\mathbf{R}_{true}$ and the corresponding 3×3 identical matrix $\mathbf{I}_3$, to ensure the predicted rotation matrix is always invertable, as Eq. (5.9) shows:

$$\mathbf{Loss}_R = \mathbf{RMSE}\left(\left(\mathbf{R}_{pred}\right)^{-1}\mathbf{R}_{true}, \mathbf{I}_3\right) \tag{5.9}$$

Translation Loss

Similar to rotation loss, we compute the **RMSE** between the predicted translation vector $\mathbf{T}_{pred}$ and the true translation vector $\mathbf{T}_{true}$ as the translation loss, shown in Eq. (5.10):

$$\mathbf{Loss}_T = \mathbf{RMSE}\left(\mathbf{T}_{pred}, \mathbf{T}_{true}\right) \tag{5.10}$$

Transformation Loss

To combine rotation and translation loss, we compute a transformation matrix $\mathbf{\Phi} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0} & 1 \end{bmatrix}$, then compute a transformation loss shown in Eq. (5.11) which is similar to the rotation loss in Eq. (5.9):

$$\mathbf{Loss}_{transform} = \mathbf{RMSE}\left(\left(\mathbf{\Phi}_{pred}\right)^{-1}\mathbf{\Phi}_{true}, \mathbf{I}_4\right) \tag{5.11}$$

where $\mathbf{\Phi}_{pred}$ and $\mathbf{\Phi}_{true}$ are the predicted and the true transformation matrix, respectively; $\mathbf{I}_4$ is a 4×4 identical matrix.

Center Loss & Center Drift

Rotation loss $\mathbf{Loss}_R$, translation loss $\mathbf{Loss}_T$ and transformation loss $\mathbf{Loss}_{transform}$ use a matrix to compute the difference between the predicted pose parameters and the ground truth. Instead we define a center loss which compares the difference between the center of the predicted template patch and the center of the true corresponding template patch. Center loss $\mathbf{Loss}_{center}$ is point-based, and describes the difference in meters, which is unit compatible. In detail, after a predicted rotation matrix $\mathbf{R}^i_{pred}$ and a predicted translation vector $\mathbf{T}^i_{pred}$ is retrieved for a source patch P^i_{source}, we apply them for the source patch P^i_{source} to compute

a predicted template patch $P^{pred,i}_{template}$. Center loss is computed as a mean value of **RMSE** by comparing the center of each predicted template patch with the center of its true template patch, shown in Eq. (5.12). The center drift Δ_{center} for each predicted template patch is calculated as well, shown in Eq. (5.13):

$$\mathbf{Loss}_{center} = \frac{1}{N_{nn}} \sum_{i=1}^{N_{nn}} \mathbf{RMSE}\left(\frac{1}{N_{patch}} \sum_{j=1}^{N_{patch}} P^{pred,i}_{template,j}, \frac{1}{N_{patch}} \sum_{j=1}^{N_{patch}} P^{i}_{template,j} \right) \tag{5.12}$$

$$\Delta^{i}_{center} = \frac{1}{N_{patch}} \sum_{j=1}^{N_{patch}} \left(P^{i}_{template,j} - P^{pred,i}_{template,j} \right) \quad 1 \le i \le N_{nn} \tag{5.13}$$

where $P^{pred,i}_{template,j} = \mathbf{R}^{i}_{pred} P^{i}_{source,j} + \mathbf{T}^{i}_{pred}$, N_{nn} is the number of patches in a source or a template list, N_{patch} is the number of points for a single patch.

5.3.3. Total Loss Composition

The overall expression of the total loss contains two parts: a pose loss $\mathbf{Loss}_{pose}$ and a labeling loss $\mathbf{Loss}_{cor}$, with balancing factor α, β, respectively, as Eq. (5.14) shows:

$$\mathbf{Loss} = \alpha \mathbf{Loss}_{pose} + \gamma \mathbf{Loss}_{cor} \tag{5.14}$$

In our research we design two types of the pose loss $\mathbf{Loss}_{pose}$: a rotation loss $\mathbf{Loss}_{R}$ together with a combined pose loss $\mathbf{Loss}_{cob}$ which is a transformation loss $\mathbf{Loss}_{transform}$ or a center loss $\mathbf{Loss}_{center}$, because rotation has a larger impact on the final result e.g. a 0.01 rad offset in rotation will lead to about 1 m offset in translation, if we multiply a coordinate of 100 m. We assign more weight to the rotation loss by giving a large balancing factor in the total loss function. The pose loss is defined as Eq. (5.15):

$$\mathbf{Loss}_{pose} = \mathbf{Loss}_{R} + \epsilon \mathbf{Loss}_{cob} \tag{5.15}$$

where $\mathbf{Loss}_{cob} = \mathbf{Loss}_{center}$ *or* $\mathbf{Loss}_{transform}$ and ϵ is the balancing factor.

The final loss contains three parts: a labeling loss, a rotation loss and a combined pose loss, with balancing factor $\alpha, \beta = \alpha \cdot \epsilon$ and γ, as shown in Eq. (5.16):

$$\mathbf{Loss} = \alpha \mathbf{Loss}_{R} + \beta \mathbf{Loss}_{cob} + \gamma \mathbf{Loss}_{cor} \tag{5.16}$$

Translation loss $\mathbf{Loss}_{T}$ is less meaningful because we have included the rotation error. The predicted translation vector will vary a lot, especially as coordinates range from about -200 m to 200 m in our research. We do not include the translation loss in the total loss during training, and only compute the predicted translation for comparison in the assessment.

5.4. Global Prediction Refinement

It is challenging to train a neural network accurately for registration at large scale. Even for point-based networks based on simple objects, have obvious transformation offsets sometimes [36, 70]. Different from one single object, we have N_{nn} patches for each batch distributed in 3D space in an urban area. A global refinement method consists of two steps: virtual point correspondence simulation based on the center drift correction, and global estimation, shown in Fig. 5.5.

5.4.1. Virtual Point Correspondence Simulation

Different from point-to-point iterative closest point (**ICP**) [10] which realizes registration by finding point corresponding pairs and DeepVCP [46] which simulates virtual correspondences for key points in the source under a voxel-based convolutional neural network, in our research, we simulate virtual point correspondences for every point in source patches based on the predicted rotation matrix, the predicted translation vector and different center drifts, as the upper part of Fig. 5.5 shows. It is direct and saves time, without searching neighbors for each point.

For one patch P^{i}_{source} in the source list where there are N_{nn} patches, virtual corresponding points for this patch are computed by adding the center drift Δ^{i}_{center} for this patch to the predicted patch $P^{pred,i}_{template}$, to form a pair matrix p^{i}_{vir} consisting of N_{patch} pairs, as Eq. (5.17) shows:

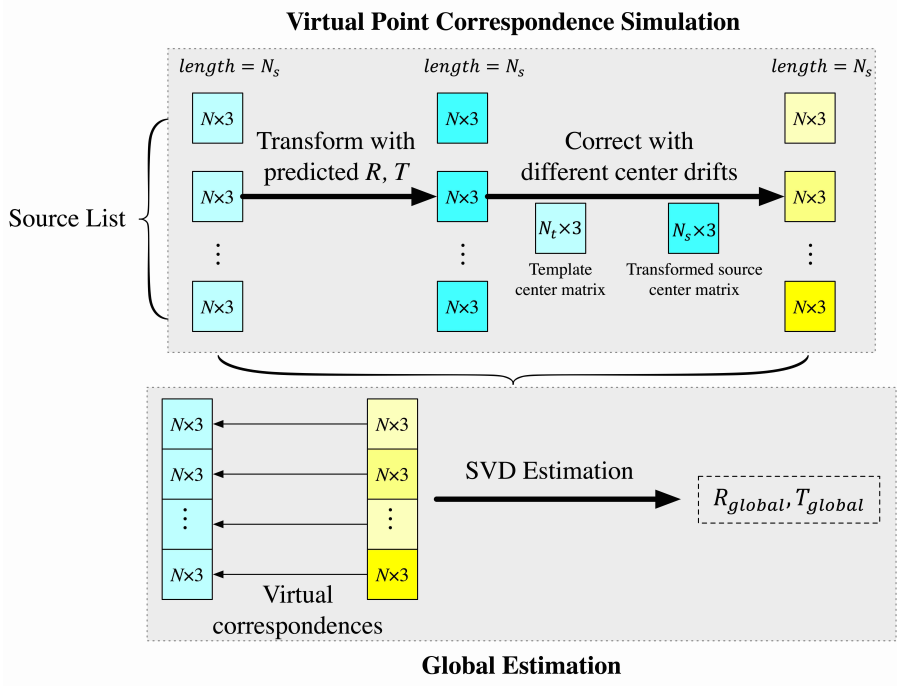


Figure 5.5: Virtual point correspondence simulation based on center drift correction and global estimation. Patches in the source list are transformed by predicted rotation matrix $\mathbf{R}$ and translation vector $\mathbf{T}$, then are added by different center drifts to generate virtual corresponding points. Source patch points and their correspondences are estimated by SVD decomposition to output an overall rotation matrix $\mathbf{R}_{global}$ and translation vector $\mathbf{T}_{global}$.

$$p_{vir}^{i} = \left[P_{source}^{i},\ P_{template}^{pred,i} + \Delta_{center}^{i} \right],\ 1 \le i \le N_{nn} \tag{5.17}$$

Then we get $N_{nn} \times N_{patch}$ point corresponding pairs. For each patch P_{source}^{i}, the predicted rotation matrix is the same as $\mathbf{R}_{pred}$, while the predicted translation vector varies, which depends on the center drift, $\mathbf{T}_{pred} + \Delta_{center}^{i}$. We have N_{nn} different translation vectors after virtual point correspondence simulation, thus a global estimation needs to be applied to output a global transformation matrix for all point pairs.

5.4.2. Global Estimation

After generating virtual correspondences, we apply a global estimation for all point corresponding pairs from N_{nn} different patches, with $N_{nn} = 10$ the length of input source or template list. Based on SVD decomposition for 3D point least square estimation [5], which estimates a rotation matrix and a translation vector by SVD decomposition for a matrix of points and their correspondences, we retrieve a global rotation matrix and a global translation vector for all points from different source patches within the same batch, as the lower part of Fig. 5.5 shows. The algorithm is shown as Algorithm 6:

Algorithm 6: Global rotation matrix & translation vector estimation

input: A list of virtual corresponding pairs $\mathbf{p}_{vir} = \left[P_{source},\ P_{vircor} = P_{template}^{pred} + \Delta_{center} \right]$

begin

$P_{source}^{Re} = P_{source} - \mathbf{center}(P_{source})$

$P_{vircor}^{Re} = P_{vircor} - \mathbf{center}(P_{vircor})$

$\mathbf{\Psi} = \left(P_{source}^{Re}\right)\left(P_{vircor}^{Re}\right)^{T}$

$[\mathbf{U}, \mathbf{S}, \mathbf{V}] = SVD(\mathbf{\Psi})$

$\mathbf{R}_{global} = \mathbf{V}\mathbf{U}^{T}$

$\mathbf{T}_{global} = \mathbf{center}(P_{vircor}) - \mathbf{R}_{global} \cdot \mathbf{center}(P_{source})$

end

output: A global rotation matrix $\mathbf{R}_{global}$, and a global translation vector $\mathbf{T}_{global}$

Finally we realize pose refinement, by getting a transformation matrix $\begin{bmatrix} \mathbf{R}_{global} & \mathbf{T}_{global} \\ \mathbf{0} & 1 \end{bmatrix}$ for a corresponding pair $\left(\boldsymbol{S}'^{\,i}_{source}, \boldsymbol{S}'^{\,j}_{template}\right)$ which is the output of place recognition.

6

Results on Place Recognition and Pose Refinement

Section 6.1 introduces the setup of experiments on place recognition and pose refinement. Section 6.2 and 6.3 give place recognition and pose refinement results, as well as the evaluation of the patch based neural network and comparison to some traditional registration methods. Finally, the application scope of place recognition and pose refinement are discussed in Section 6.4.

6.1. Experimental Setup

Before processing experiments, the description of the dataset, evaluation criteria, implementation environment, and some key settings for both place recognition and pose refinement are given.

6.1.1. Dataset Description

In Chapter 3, our terrestrial laser scanning (**TLS**) system and mobile laser scanning (**MLS**) system are introduced. Additionally, we give details on the input of place recognition and pose refinement after data preprocessing.

Dataset for Place Recognition

5 typical **MLS** scenes and corresponding **TLS** scans are selected, covering most parts of the TU Delft campus. As Fig. 6.1 shows, we use number 1-5 to label **TLS** and **MLS** data, with (TLS_i, MLS_i) $(1 \leq i \leq 5)$ as a corresponding pair. Number of points and 2D dimension are shown in Table 6.1. Input **MLS** scenes have more points and larger coverage than corresponding **TLS** scans.

Table 6.1: Number of points of $\left(TLS_i, MLS_i\right)$, $(1 \leq i \leq 5)$, including horizontal dimension

i	MLS_i		TLS_i	
1	2,860,664	306 m × 224 m	1,305,056	229 m × 190 m
2	7,357,693	174 m × 553 m	1,324,103	223 m × 230 m
3	9,708,025	224 m × 735 m	1,329,581	225 m × 226 m
4	8,211,967	174 m × 664 m	1,890,049	193 m × 216 m
5	8,736,419	149 m × 524 m	1,625,616	227 m × 212 m

Dataset for Pose Refinement

We use two datasets: the TU Delft dataset consisting of corresponding pairs estimated from place recognition, connected to Table 6.1 and Fig. 6.1; and the Shanghai dataset consisting of 3 **MLS** scenes (each scene consists of ~1 million points following preprocessing introduced in Chapter 3) of an urban area in Shanghai, shown in Fig. 6.3. For the TU Delft dataset, source patches and template patches are generated from **TLS** scans

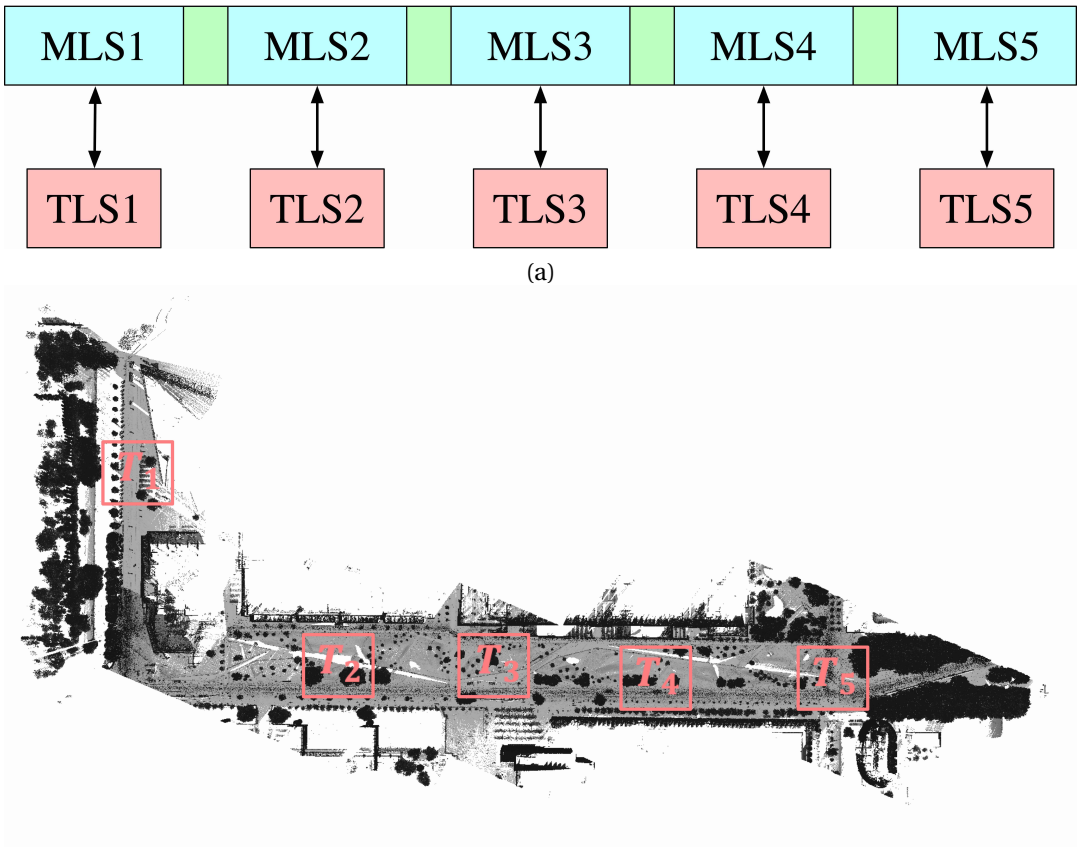


Figure 6.1: (a) Overview of 5 typical **MLS** scenes (cyan) and corresponding **TLS** scans (red) at TU Delft campus. Green represents overlap between **MLS** scenes. (b) Relative location of 5 **TLS** scans (T_i corresponds to TLS_i) in the **MLS** reference map

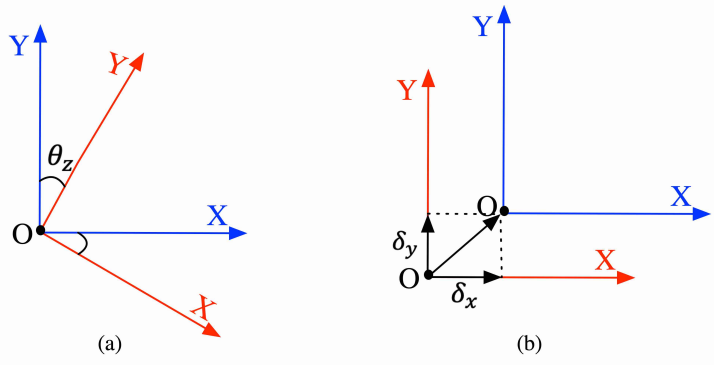


Figure 6.2: (a) A rotation range of $(0, \theta_z)$ w.r.t. the z axis between **TLS** (red) and **MLS** (cyan) (b) A horizontal translation range of $(0, \delta_x)$ and $(0, \delta_y)$ w.r.t. the x and the y axis, between **TLS** (red) and **MLS** (blue)

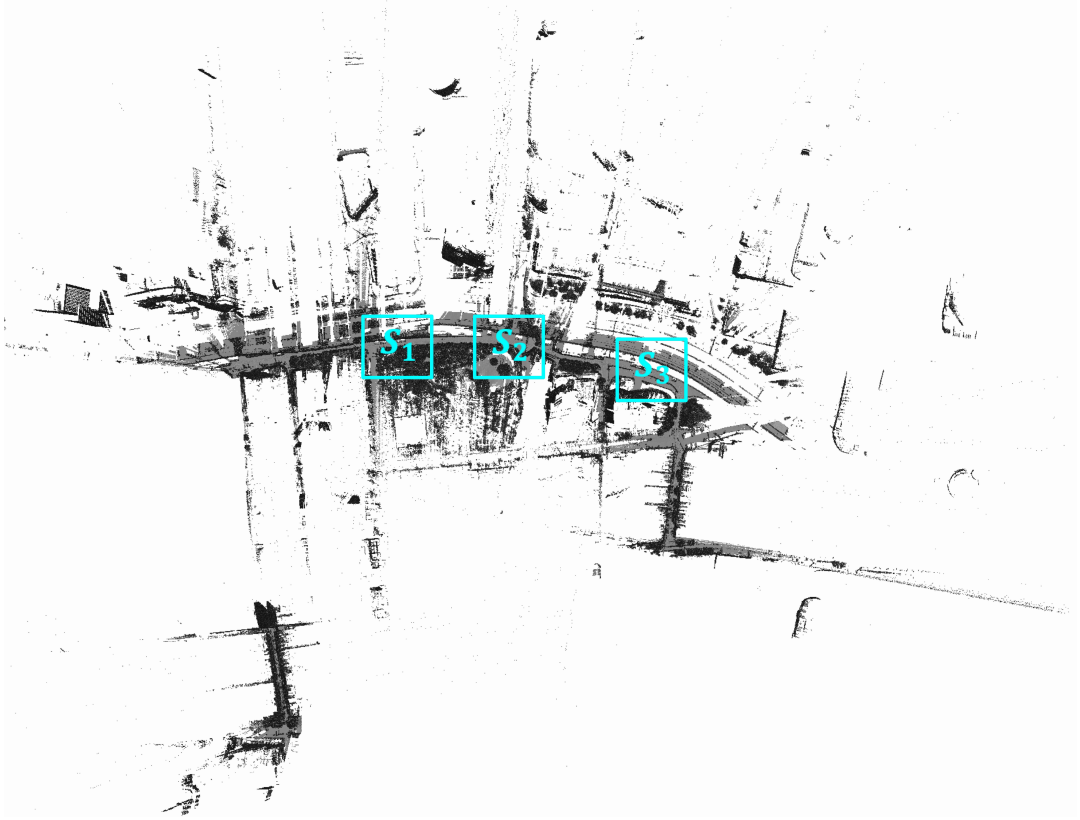


Figure 6.3: An aerial view of 3 **MLS** scenes (S_1, S_2 and S_3) of the Shanghai dataset

and corresponding **MLS** scenes, respectively. For the Shanghai dataset, both source and template patches are generated from the **MLS** point clouds. The TU Delft dataset uses both **TLS** and **MLS** point clouds, and will be used in every pose refinement experiment to give an overview of the localization process. The Shanghai dataset only uses **MLS** point clouds, and will only be used in evaluation and comparison from Section 6.3.5 to Section 6.3.7. Details of both datasets are shown in Table 6.2. Particularly, both test datasets are chosen from a completely new area that does not participate in training and validation process. There are some differences of the rotation and the translation range between these two datasets. A rotation range and a translation range are an angle range and an offset range, w.r.t. the (x, y, z) axis, respectively, shown in Fig. 6.2. A value within a rotation range or a translation range is called a rotation angle or a translation offset w.r.t. the certain axis, respectively. The Shanghai dataset has a larger rotation range w.r.t. the z axis and a larger horizontal translation range. These settings aim to evaluate pose refinement at a larger scale.

Table 6.2: Description of TU Delft and Shanghai dataset

Hyperparameter	TU Delft dataset	Shanghai dataset
Number of batches	3968 as training set 992 as holdout set 1600 as test set	2560 as training set 640 as holdout set 1000 as test set
Number of points per batch	10 (patches) × 256 (points)	10 (patches) × 256 (points)
Voxel patch size w.r.t. the (x, y, z) axis	(5 m, 5 m, 3 m)	(5 m, 5 m, 3 m)
Rotation range w.r.t. the (x, y, z) axis	$[(0,\pi/18),\ (0,\pi/18),\ (0,\pi/6)]$	$[(0,\pi/18),\ (0,\pi/18),\ (0,\pi)]$
Translation range w.r.t. the (x, y, z) axis	$[(0,30\ m),\ (0,30\ m),\ (0,10\ m]$	$[(0,50\ m),\ (0,50\ m),\ (0,10\ m]$
(α, β, γ) in Eq. (5.16)	(100, 0.1, 100) or (100, 100, 100)	(100, 0.1, 100) or (100, 100, 100)

6.1.2. Evaluation Criteria

During experiments, some results e.g. run time, number of extracted features and training loss, are computed. Specifically, there are four metrics: confusion matrix, precision-recall curve, mean and max pose error that are defined below and are used for evaluation under the test dataset.

Confusion Matrix w.r.t. Patch & Batch

A confusion matrix is a specific table that allows visualization of the performance of an algorithm [77]. There are four types of element in a confusion matrix: true positives (**TP**), false positives (**FP**), true negatives (**TN**) and false negatives (**FN**). As our registration neural network is patch based, with $N_{nn} = 10$ patches for each batch, we compute two different confusion matrices w.r.t. patch and batch, respectively.

After the patch correspondence search block (**CSB**) introduced in Section 5.2.3, as Eq. (5.5) shows, in a single batch we get N_{nn} pairs of $cor_i = (i, j)$ $(1 \le i \le N_{nn}, 1 \le j \le N_{nn})$ for the i^{th} patch in the source list and its most possible corresponding patch, the j^{th} patch in the template list. $\mathbf{TP}_p$, $\mathbf{FP}_p$, $\mathbf{TN}_p$ and $\mathbf{FN}_p$ w.r.t. the patch confusion matrix are defined in Eq. (6.1):

$$\begin{aligned} cor_i &= \mathbf{TP}_p \quad \textbf{if } \mathbf{C}(i,j) \ge r \ \& \ \mathbf{C}_{true}(i,j) = 1 \\ cor_i &= \mathbf{FP}_p \quad \textbf{if } \mathbf{C}(i,j) \ge r \ \& \ \mathbf{C}_{true}(i,j) = 0 \\ cor_i &= \mathbf{TN}_p \quad \textbf{if } \mathbf{C}(i,j) < r \ \& \ \mathbf{C}_{true}(i,j) = 0 \\ cor_i &= \mathbf{FN}_p \quad \textbf{if } \mathbf{C}(i,j) < r \ \& \ \mathbf{C}_{true}(i,j) = 1 \end{aligned} \tag{6.1}$$

where $r = 0.95$, $\mathbf{C}$ and $\mathbf{C}_{true}$ are the predicted and the true correspondence matrix defined in Section 5.2.3, respectively. We compute $\mathbf{TP}_p$, $\mathbf{FP}_p$, $\mathbf{TN}_p$ and $\mathbf{FN}_p$ for all B batches in experiments. Fig. 6.4 (a) gives an example of elements in a patch confusion matrix with 5 patches, with $cor(i,i)$ $(1 \le i \le 5)$ as a true corresponding pair. $\mathbf{TP}_p$, $\mathbf{FP}_p$, $\mathbf{TN}_p$ and $\mathbf{FN}_p$ describe the performance of the patch corresponding pair estimation based on the correspondence matrix, especially $\mathbf{TP}_p$ and $\mathbf{FP}_p$, since pairs predicted as positives are used to output pose parameters in the pose estimation block (**PEB**) introduced in Section 5.2.4.

Similarly, for a batch b_k $(1 \le k \le B)$, $\mathbf{TP}_b$, $\mathbf{FP}_b$, $\mathbf{TN}_b$ and $\mathbf{FN}_b$ w.r.t. the batch confusion matrix are defined in Eq. (6.2):

$$\begin{aligned} b_k &= \mathbf{TP}_b \quad \textbf{if } \mathbf{TP}_p + \mathbf{FP}_p \ge \frac{N_{nn}}{2} \ \& \ \frac{\mathbf{TP}_p}{\mathbf{TP}_p + \mathbf{FP}_p} \ge \epsilon \\ b_k &= \mathbf{FP}_b \quad \textbf{if } \mathbf{TP}_p + \mathbf{FP}_p \ge \frac{N_{nn}}{2} \ \& \ \frac{\mathbf{TP}_p}{\mathbf{TP}_p + \mathbf{FP}_p} < \epsilon \\ b_k &= \mathbf{TN}_b \quad \textbf{if } \mathbf{TP}_p + \mathbf{FP}_p < \frac{N_{nn}}{2} \ \& \ \frac{\mathbf{TP}_p}{\mathbf{TP}_p + \mathbf{FP}_p} < \epsilon \\ b_k &= \mathbf{FN}_b \quad \textbf{if } \mathbf{TP}_p + \mathbf{FP}_p < \frac{N_{nn}}{2} \ \& \ \frac{\mathbf{TP}_p}{\mathbf{TP}_p + \mathbf{FP}_p} \ge \epsilon \end{aligned} \tag{6.2}$$

where $\epsilon = 0.8$, $\mathbf{TP}_p$, $\mathbf{FP}_p$, $\mathbf{TN}_p$, $\mathbf{FN}_p$ are computed by patch pairs in b_k. Fig. 6.4 (b) gives an example of $\mathbf{TP}_b$, $\mathbf{FP}_b$, $\mathbf{TN}_b$ and $\mathbf{FN}_b$ for four different batches, with 10 patches for each batch. $\mathbf{TP}_b$, $\mathbf{FP}_b$, $\mathbf{TN}_b$ and $\mathbf{FN}_b$ describe the performance of results computed based on batches. For example, if the number of patch positives ($\mathbf{TP}_p + \mathbf{FP}_p$) in a batch is smaller than 5, we will not use this batch in **PEB**, because there are less patches to apply the global refinement. If there are enough patch positives ($\mathbf{TP}_p + \mathbf{FP}_p > 5$) but the rate $\frac{\mathbf{TP}_p}{\mathbf{TP}_p + \mathbf{FP}_p}$ is low (< 0.8) in a batch, many false patch positives ($\mathbf{FP}_p$), shown as wrong corresponding pairs, are used in **PEB**, then predicted pose parameters and the global prediction refinement are not reliable.

Precision-Recall Curve

Both precision and recall are computed from a confusion matrix, as Eq. (6.3) shows. Based on these 2 metrics, we draw a precision-recall (**PR**) curve to show the tradeoff between the precision and the recall under different thresholds. A high precision indicates there are less **FP** and a high recall relates to less **FN**, but it is hard to ensure a high level of both precision and recall in most circumstances, so we need to make a balance between

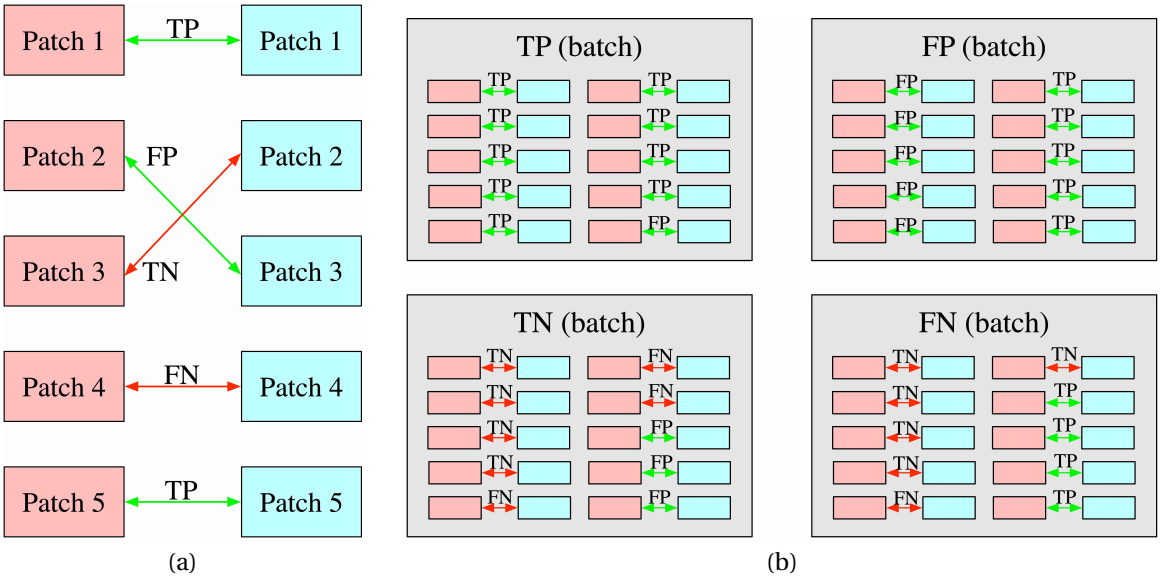


Figure 6.4: (a) True positives (**TP**), false positives (**FP**), true negatives (**TN**) and false negatives (**FN**) between source patches (red) and template patches (cyan). Red lines represent negatives and green lines represent positives. (b) **TP**, **FP**, **TN** and **FN** w.r.t. batch, based on **TP**, **FP**, **TN** and **FN** w.r.t. patch computed in a batch

precision and recall to choose a proper threshold. In order to well show the curve we plot a **(1-precision)-recall** curve in our experiments. We also call it **PR** curve in later experiments.

$$\begin{aligned} \textbf{Precision} &= \frac{\textbf{TP}}{\textbf{TP}+\textbf{FP}} \\ \textbf{Recall} &= \frac{\textbf{TP}}{\textbf{TP}+\textbf{FN}} \end{aligned} \tag{6.3}$$

In general, the patch precision relates to the success rate of the patch corresponding pair estimation, and the patch recall indicates the percentage of patch corresponding pairs that are correctly estimated. The batch precision relates to the accuracy of estimated pose parameters, and the batch recall indicates the percentage of reliable batches ($\frac{\textbf{TP}_p}{\textbf{TP}_p+\textbf{FP}_p} > 0.8$) that are used in experiments.

Mean Pose Error

The accuracy of localization is expressed as mean pose error. The mean pose error combines two errors: mean rotation error ϵ_r in degrees and mean translation error ϵ_t in meters. Both types are computed as a mean root mean square error (**RMSE**) between predicted values and true values, rotation angles $(\theta_x, \theta_y, \theta_z)$ and translation offsets $(\delta_x, \delta_y, \delta_z)$, w.r.t. the (x, y, z) axis in Eq. (2.1) over all batches, respectively, as Eq. (6.4) shows:

$$\begin{aligned} \epsilon_r &= \frac{1}{B}\sum_{i=1}^{B} \textbf{RMSE}\left(\boldsymbol{\theta}^i_{pred}, \boldsymbol{\theta}^i_{true}\right) \\ \epsilon_r &= \frac{1}{B}\sum_{i=1}^{B} \textbf{RMSE}\left(\boldsymbol{\delta}^i_{pred}, \boldsymbol{\delta}^i_{true}\right) \end{aligned} \tag{6.4}$$

where $\boldsymbol{\theta}_{pred}$ and $\boldsymbol{\theta}_{true}$ are the predicted and the true $(\theta_x, \theta_y, \theta_z)$, respectively; $\boldsymbol{\delta}_{pred}$ and $\boldsymbol{\delta}_{true}$ are the predicted and the true $(\delta_x, \delta_y, \delta_z)$, respectively; B is the number of batches.

Max Pose Error

Similar to the mean pose error, the max rotation error η_r and the max translation error η_t are defined in Eq. (6.5):

$$\eta_r = \max_{1 \le i \le B} \mathbf{RMSE}\left(\boldsymbol{\theta}^i_{pred}, \boldsymbol{\theta}^i_{true}\right)$$
$$\eta_r = \max_{1 \le i \le B} \mathbf{RMSE}\left(\boldsymbol{\delta}^i_{pred}, \boldsymbol{\delta}^i_{true}\right) \tag{6.5}$$

where $\boldsymbol{\theta}_{pred}$ and $\boldsymbol{\theta}_{true}$ are the predicted and the true $(\theta_x, \theta_y, \theta_z)$, respectively; $\boldsymbol{\delta}_{pred}$ and $\boldsymbol{\delta}_{true}$ are the predicted and the true $(\delta_x, \delta_y, \delta_z)$, respectively; B is the number of batches.

6.1.3. Implementation Environment

The training process is ran at the TU Delft high performance cluster (**HPC**) using a CentOS 7 Linux distribution, with 1080Ti geographic processing unit and Pyhon 3.7.7, TensorFlow 2.3.0, Open3D 0.9.0. Other experiments are processed under Mac OS Catalina 10.15.6, with 2.9 GHz Quad-Core Intel Core i7 processor and 16 GB 2133 MHz LPDDR3 memory. The compiler is Jupyter Notebook 6.0.3 for python3.

6.1.4. Experiment Settings

Some parameters in Chapter 4 and Chapter 5 were introduced. This section addresses the settings that are used for place recognition and pose refinement.

Place Recognition Settings

Experiments of place recognition are processed with or without line feature extraction for **TLS** scans and **MLS** scenes following the cylinder feature extraction method in Section 4.3.3, to check the contribution of line features during corresponding pair estimation introduced in Section 4.4. Pair candidates are generated based on computed similarities ρ and mean distances μ defined in Eq. (4.17) and Eq. (4.18) in Section 4.4.3. Results of place recognition are decided following the decision making strategy in Section 4.4.4.

Pose Refinement Settings

Both the TU Delft dataset and the Shanghai dataset are trained for 3000 epochs, using two types of loss functions defined in Eq. (5.16) in Section 5.3.3, with different balancing factors as shown in Table 6.2. During training, batch normalization which applies recentering and rescaling for input layers is used in **MLP**s in block **A** – the feature extraction block (**FEB**) introduced in Section 5.2.2 to make neural networks faster and more reliable. A dropout rate of (0.3, 0.3, 0.1, 0.0, 0.0, 0.0) is applied in **PEB** to avoid overfitting by randomly assigning zeros to input layers.

6.2. Place Recognition Results

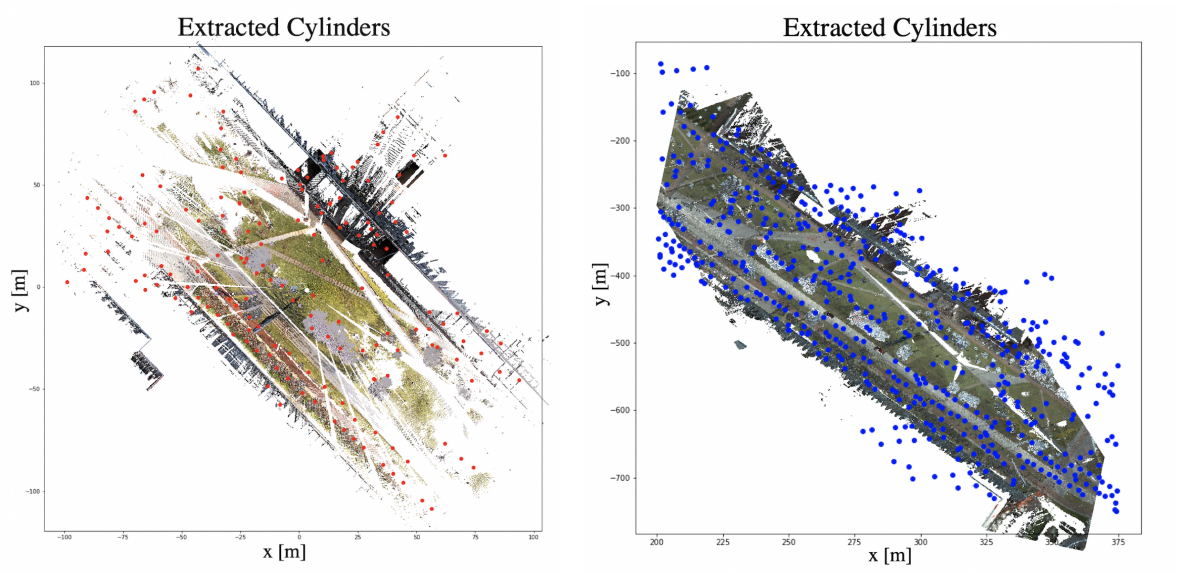
Extracted features and corresponding pair estimation are shown in results of place recognition in Section 6.2.1 and Section 6.2.2, respectively. Next an overview of intermediate results during localization, place recognition results is visualized in Section 6.2.3.

6.2.1. Feature Extraction Results

Following the 9-step feature extraction method in Section 4.3.3, we extract different numbers of cylinders. The run time of line and cylinder extraction, the number of extracted cylinders for each **TLS** scan and **MLS** scene are shown in Table 6.3. A comparison to Table 6.1 shows that about 100 cylinders are extracted per million points from **TLS** or **MLS** point clouds at 0.1 m voxel downsampling. We visualize the layout of extracted cylinders and they well represent corresponding **TLS** or **MLS** point clouds, shown in Fig. 6.5. It is more time consuming to extract features from the **MLS** point clouds due to their large number of points. Run time per million points of cylinder feature extraction is longer in **TLS** scans, about 10 seconds compared to 5 seconds in **MLS** scenes. The reason for this longer run time is that there is less information of the facades but higher point density at canopies and terrain in **TLS** scans, as Fig 6.6 shows.

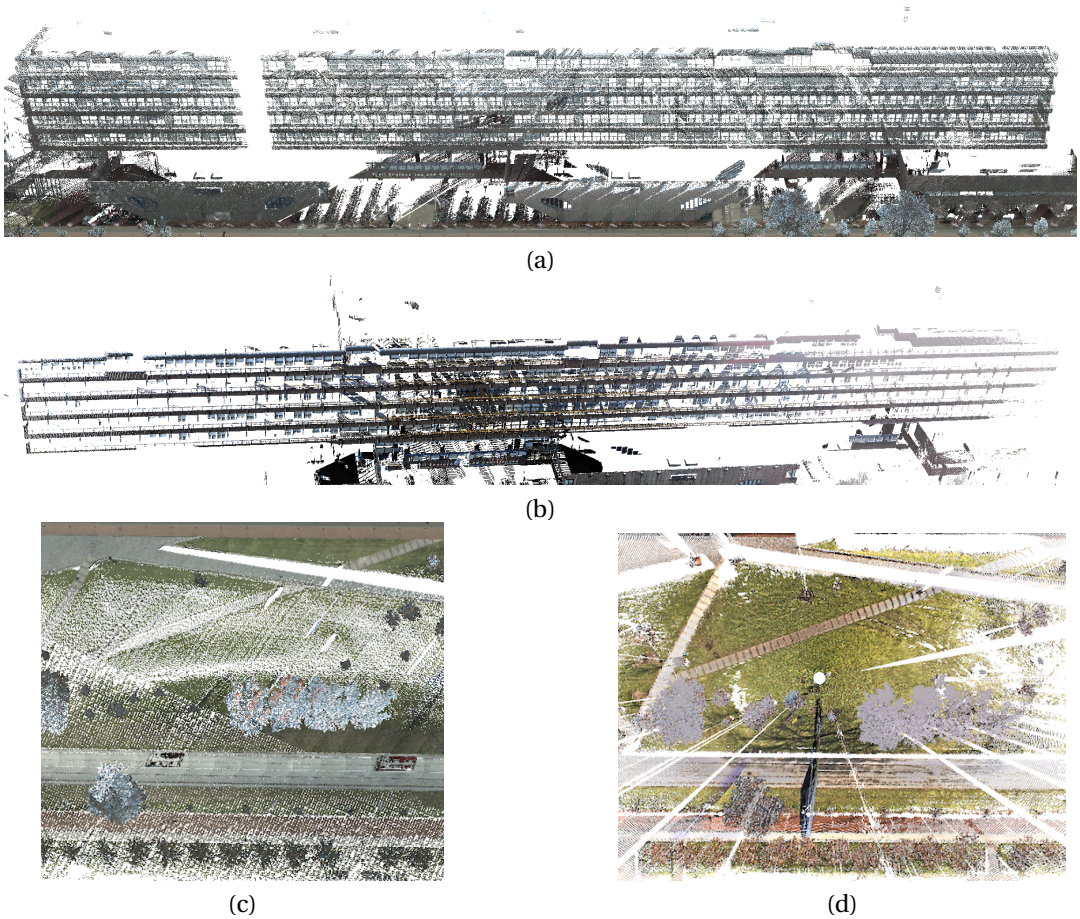
6.2.2. Recognized Results based on Extracted Features

Based on the Gaussian mixture model (**GMM**) probabilistic estimation in Section 4.4, cylinders of **TLS** scans are transformed by a rotation matrix **R** and translation vector **T** following Algorithm 4 in Section 4.4.2. Next we compute the similarity and the mean distance for each pair of **TLS** and **MLS** point clouds based on Eq. (4.17) and Eq. (4.18). The results are shown in Table 6.4. There are red outliers which are below our threshold



(a) (b)

Figure 6.5: An aerial view of extracted cylinders in (a) a **TLS** scan (red) (b) a **MLS** scene (blue)



(a)

(b)

(c) (d)

Figure 6.6: A facade and a certain area in **MLS** point cloud and corresponding **TLS** point cloud at the same scale. (a) A **MLS** facade (b) A **TLS** facade (c) A **MLS** area (d) A **TLS** area

Table 6.3: Run time of line & cylinder feature extraction and number of extracted cylinders in 5 **TLS** scans and corresponding **MLS** scenes

Dataset		Line extraction time	Cylinder extraction time	No. of extracted cylinders
MLS	MLS_1	$9.5s$	$25.9s$	298
	MLS_2	$35.8s$	$41.5s$	522
	MLS_3	$47.0s$	$53.1s$	641
	MLS_4	$41.0s$	$40.0s$	498
	MLS_5	$47.4s$	$45.3s$	467
TLS	TLS_1	$6.3s$	$12.4s$	191
	TLS_2	$8.5s$	$11.6s$	159
	TLS_3	$5.4s$	$11.0s$	235
	TLS_4	$5.9s$	$12.2s$	168
	TLS_5	$5.0s$	$12.6s$	242

Table 6.4: Computed similarity ρ & mean distance μ between 5 **TLS** static scans and 5 **MLS** scenes. Red represents similarities or mean distances below our threshold given in Section 4.4.4; blue means there are multiple possible matching pairs (computed similarities are close) between **TLS** and **MLS** data; green indicates a dominant solution (there is a computed similarity which is obviously larger than others) between **TLS** and **MLS** data.

(ρ, μ)	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	(0.73, 7.1)	(0.70, 7.1)	(0.69, 7.1)	(0.66, 8.7)	(0.74, 6.9)
TLS_2	(0.45, 16.2)	(0.60, 10.6)	(0.60, 10.1)	(0.51, 18.6)	(0.56, 11.3)
TLS_3	(0.62, 10.5)	(0.63, 8.8)	(0.73, 7.5)	(0.63, 11.6)	(0.63, 8.5)
TLS_4	(0.69, 7.2)	(0.72, 6.7)	(0.77, 5.9)	(0.83, 5.5)	(0.82, 5.3)
TLS_5	(0.63, 8.9)	(0.68, 7.7)	(0.69, 7.6)	(0.64, 8.9)	(0.79, 6.1)

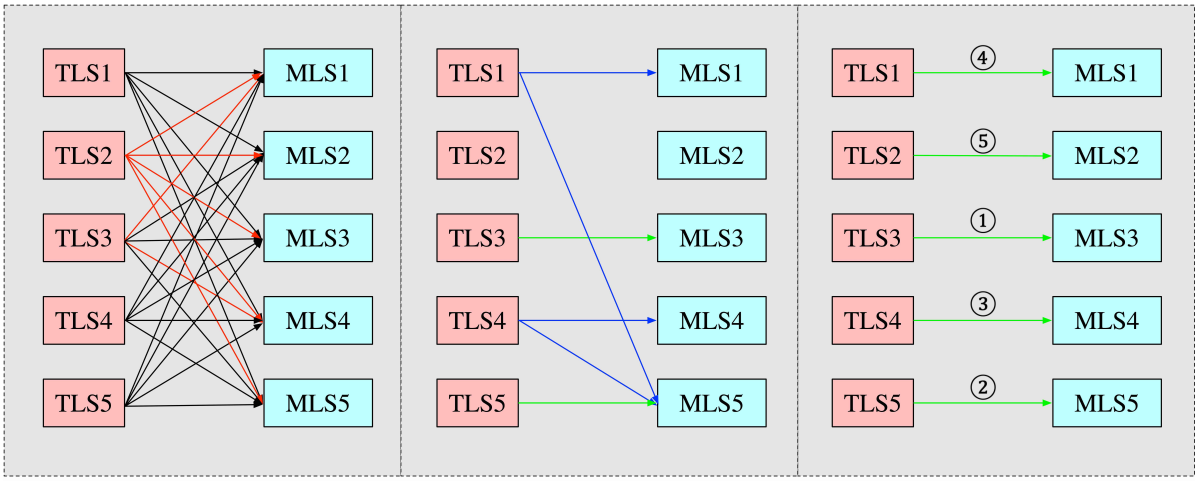
of similarity (0.6) and mean distance (10 m) e.g. (TLS_2, MLS_1), blue ambiguities which have more than one matching pair e.g. (TLS_1, MLS_1) and (TLS_1, MLS_5), green dominant solutions that have much larger similarity than other pairs e.g. (TLS_5, MLS_5). Due to the fact that our cylinders are coarsely extracted and the additional temporal change between **TLS** and **MLS** point clouds, only 2 corresponding pairs are dominant solutions, thus a decision making strategy needs to be applied to extract as many correct corresponding pairs as possible.

The decision making process is shown in Fig. 6.7, following Algorithm 5 in Section 4.4.4. In general, red outliers are removed, ambiguities and dominant solutions are found. Next we find the most dominant solution step by step until all possible pairs are decided. Details of the decision making process is shown in Table A.2 in Appendix A.

Line features are only used for filtering in step 8 as explained in Section 4.3.3. As it is relatively time consuming, especially for the **MLS** point clouds, we also included an experiment without line feature extraction. The computed similarities and mean distances are shown in Table A.1 in Appendix A, where the dominant solution (TLS_1, MLS_5) is a wrong corresponding pair, and (TLS_3, MLS_3) which is a dominant solution in Table 6.4 becomes ambiguous in Table A.1. Fortunately, still we retrieve all true corresponding pairs under the decision making strategy. Details are shown in Table. A.3 in Appendix A. It is a tradeoff between the accuracy and the efficiency to decide whether to apply line feature extraction. If extracted cylinders are reliable enough, we can skip line feature extraction to save time. In general, line features help to achieve a more reliable decisions.

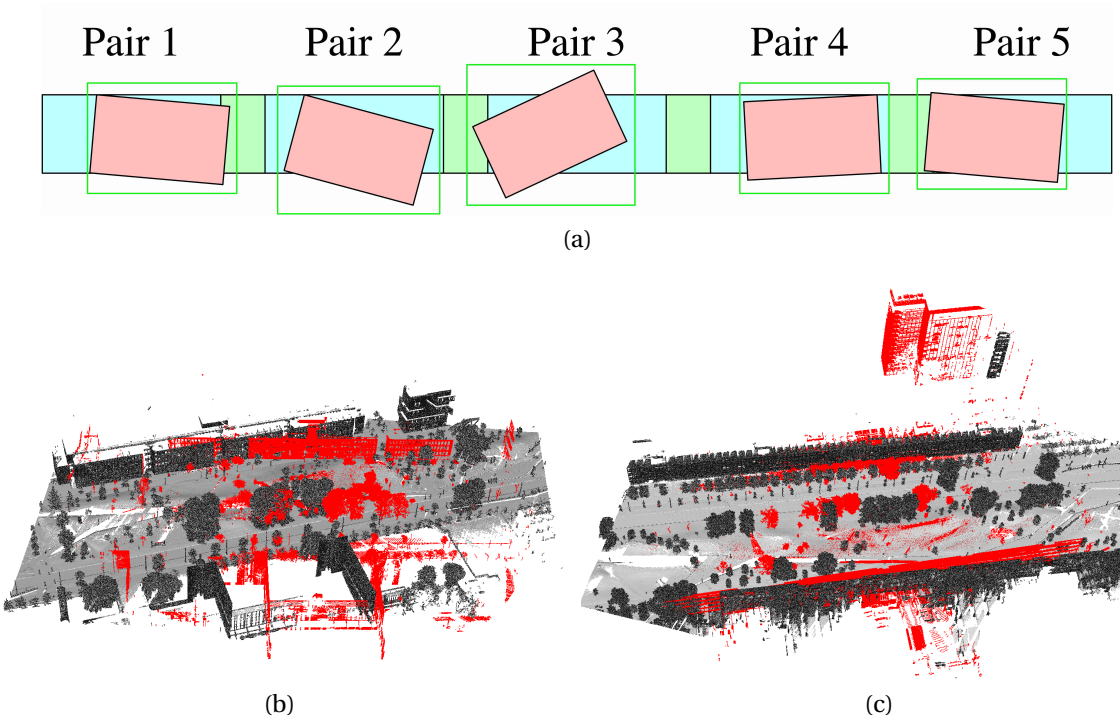
6.2.3. Overview of Place Recognition Results

Based on Algorithm 4 in Section 4.4.2, a rotation matrix $\mathbf{R}_3$ and a translation vector $\mathbf{T}_3$ are retrieved, and we apply them to transform **TLS** scans if corresponding pairs are correctly retrieved following Algorithm 5 in Section 4.4.4. Overview of successful corresponding pairs of transformed **TLS** scans and **MLS** scenes are shown in Fig. 6.8. The success rate of place recognition is **100%** in our experiments. Initial transformation varies in different corresponding pairs, depending on results of **GMM** probabilistic estimation. Fig. 6.8 (c) show a satisfying result with a small rotation angle (~ 5 degrees) between the transformed **TLS** scan and corresponding **MLS** scene w.r.t. the z axis, while (b) has a larger rotation angle (~ 25 degrees). More results



(a) Compute similarities & mean distances (b) Remove outliers (red), find ambiguities (blue) & dominant solutions (green) (c) Decision making order

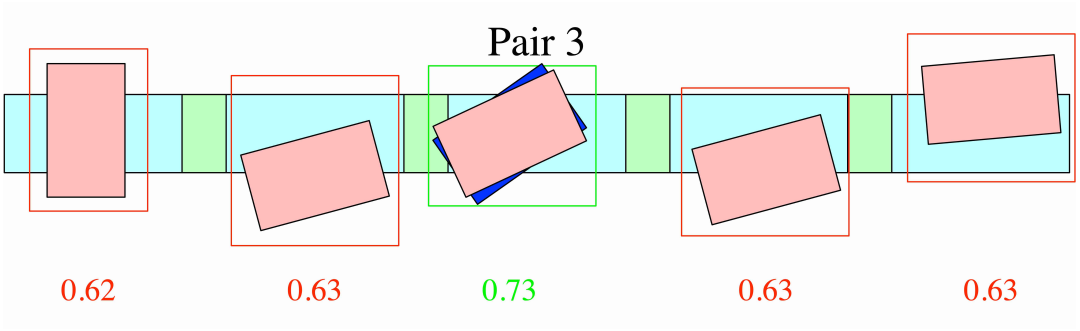
Figure 6.7: Overview of the decision making strategy based on computed similarities and mean distances in Table 6.4



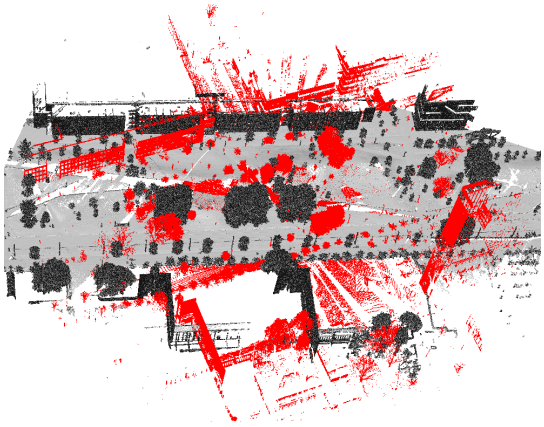
(a)

(b) (c)

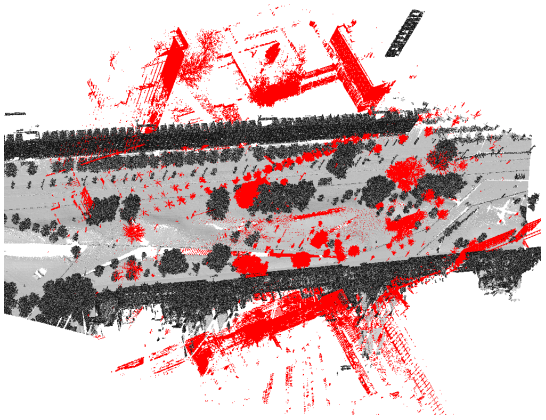
Figure 6.8: (a) 5 successful corresponding pairs (green box) of transformed **TLS** point clouds (red) and **MLS** point clouds (cyan) estimated by the decision making strategy, connected to Fig. 6.1 where pair i represents (TLS_i, MLS_i) $(1 \leq i \leq 5)$. More results are shown in Fig. A.2 in Appendix A. (b) Transformed visualization of pair 2 (TLS_2 (red), MLS_2 (grey)) where the rotation angle w.r.t. the z axis between **TLS** and **MLS** is about 25 degrees. (c) Transformed visualization of pair 4 (TLS_4 (red), MLS_4 (grey)) where the rotation angle w.r.t. the z axis between **TLS** and **MLS** is about 5 degrees.



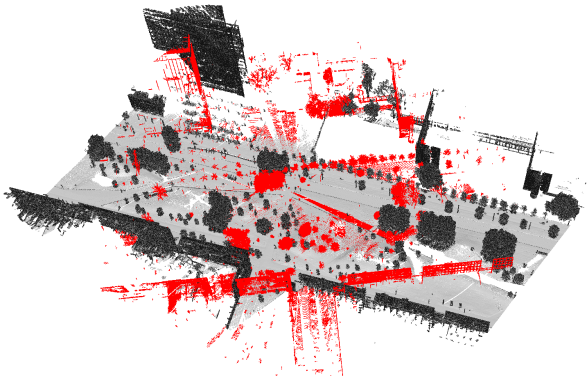
(a)



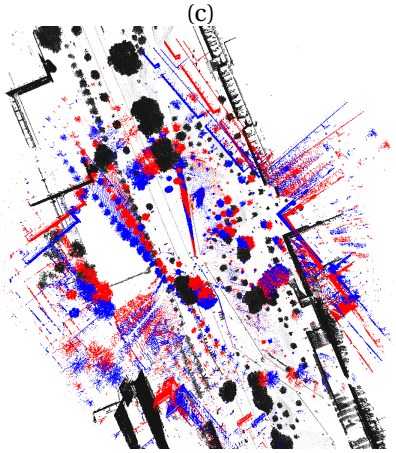
(b)



(c)



(d)



(e)

Figure 6.9: (a) 5 **MLS** candidates for transformed TLS_3, with 4 not-matching pairs (red box) and 1 matching pair as the dominant solution (green box) with 0.73 similarity, connected to Table 6.4. The transformed TLS_3 of the matching pair without line feature extraction connected to Table A.1 in Appendix A is also shown in blue. (b) Transformed visualization of a not-matching pair (TLS_3 (red), MLS_2 (grey)) with 0.63 similarity. (c) Transformed visualization of a not-matching pair (TLS_3 (red), MLS_4 (grey)) with 0.63 similarity. (d) Transformed visualization of the true matching pair 3 (TLS_3 (red), MLS_3 (grey)) with 0.73 similarity. (e) An aerial view of the true matching pair 3 (TLS_3, MLS_3) with (red) and without (blue) line feature extraction. ~ 4 degrees of the rotation angle w.r.t. the z axis are improved for transformed TLS_3 with line feature extraction.

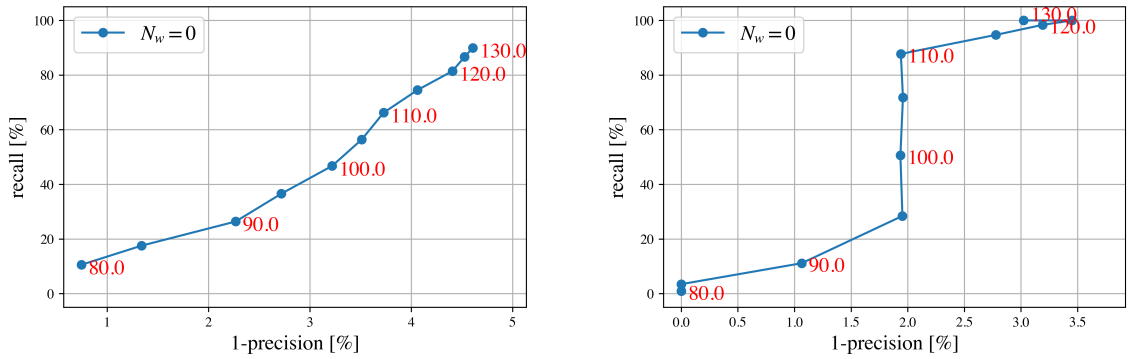


Figure 6.10: Precision-recall (**PR**) curves computed for the TU Delft validation holdout dataset. Red numbers represent the threshold of the L_2 feature distance in Eq. (5.4) in Section 5.2.3. $N_w = 0$ indicates source patches and template patches are one-to-one corresponding. (a) **PR** curve w.r.t. patch (b) **PR** curve w.r.t. batch

are shown in Fig. A.2 in Appendix A.

The way of making decision is based on the defined similarity. Basically the similarity describes the degree of overlap between transformed **TLS** scans and **MLS** scenes. For example, Fig. 6.9 (a) shows an overview of matching between TLS_3 and 5 **MLS** scenes; (b) and (c) show less degree of overlap with lower computed similarities than the matching pair in (d). Fig. 6.9 (e) indicates an improvement of line feature extraction for the rotation angle (~ 4 degrees) between the transformed **TLS** scan and corresponding **MLS** scene w.r.t. the z axis.

6.3. Pose Refinement Results

The training results and the performance of feature distance thresholds in Eq. (5.4) in Section 5.2.3 using the validation holdout dataset are shown in Section 6.3.1 and Section 6.3.2, respectively. An overview of final results during localization, pose refinement results based on the trained and tuned neural network is visualized in Section 6.3.3. Next the patch based registration neural network is evaluated by 2 test datasets. Details of the testing process are visualized in Section 6.3.4 to show the availability of our neural network. Different blocks are evaluated by the test dataset in Section 6.3.5 and Section 6.3.6 to indicate the reliability and robustness. Quantitative results of the test dataset for our neural network in comparison to some traditional registration methods are shown in Section 6.3.7 and Section 6.3.8, to validate improvements of our neural network. Section 6.3.9 gives an ablation study to show the contribution of certain blocks in our neural network.

6.3.1. Training of the TU Delft & the Shanghai Dataset

After training 3000 epochs of both the TU Delft and the Shanghai dataset, loss in Eq. (5.14) is 392, 508 under balancing factors $\alpha = 100$, $\beta = 1$, $\gamma = 100$ when $\textbf{Loss}_{cob} = \textbf{Loss}_{center}$ in Eq. 5.16 in Section 5.3.3, respectively; and 470, 462 under balancing factors $\alpha = 100$, $\beta = 100$, $\gamma = 100$ when $\textbf{Loss}_{cob} = \textbf{Loss}_{transform}$, respectively. All training losses are small enough. Training an epoch costs about 30 seconds for the TU Delft dataset and 10 seconds for the Shanghai dataset.

6.3.2. Performance of Thresholds in the Correspondence Search Block (CSB)

As we choose a loose way for feature extraction in **CSB** in Section 5.2.3, different feature distance thresholds of T_{L_2} in Eq. (5.4) in Section 5.2.3 corresponds to different recall and precision values in the **PR** curve, depending on training epochs and hyperparameters. This experiment assumes patches are one-to-one corresponding between source and template list. We use the TU Delft validation holdout set to show the performance of different T_{L_2} w.r.t. patch and batch, as Fig. 6.10 shows. Sub-figure (a) shows an approximate lineal trend under different thresholds between patch precision and recall: a high precision corresponds to a low recall. In other words, a small T_{L_2} leads to a high precision but a low recall. In sub-figure (b), there is a sharp jump in recall for four thresholds 95.0, 100.0, 105.0 and 110.0, which indicates these four thresholds do not affect batch precision. In general, it is a tradeoff between precision and recall to select a proper T_{L_2}, and we prioritize precision but make sure both patch and batch recall are higher than 40%. Finally, we choose $T_{L_2} = 100.0$ as

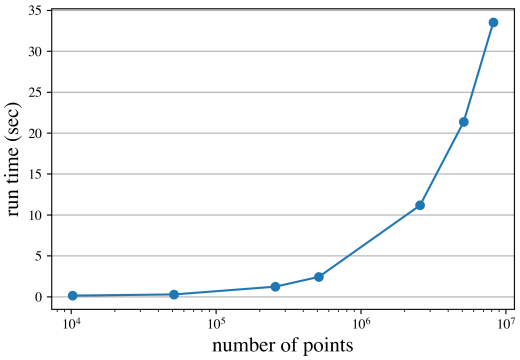


Figure 6.11: Run time of the patch correspondence search block (**CSB**) in Section 5.2.3 under different number of points processed. Processing a million points takes about 5 seconds in our experiments.

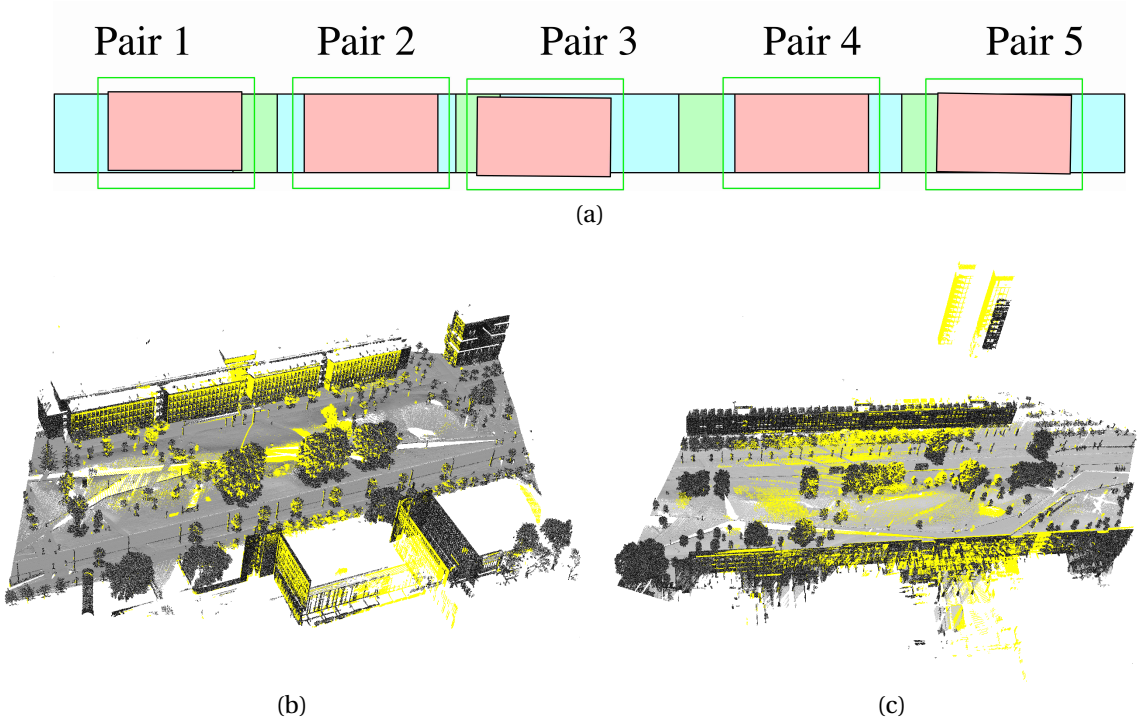


Figure 6.12: (a) Pose refinement results of 5 corresponding pairs of place recognition results, connected to Fig. 6.8 (a). More results are shown in Fig. A.3 in Appendix A. (b) Visualization of the pose refinement result for pair 2 (TLS_2 (yellow), MLS_2 (grey)) where the **TLS** scan is well aligned to corresponding **MLS** scene (almost no rotation angle w.r.t. the z axis and no horizontal translation offset). (c) Visualization of the pose refinement result for pair 4 (TLS_4 (yellow), MLS_4 (grey)) where the rotation angle w.r.t. the z axis between the **TLS** scan and corresponding **MLS** scene is small (< 1 degree), and the horizontal translation offset is smaller than 1 m.

the feature distance threshold, since this results in a recall w.r.t. patch or batch of nearly 50%, and a precision above 95% in the validation holdout set.

The run time of **CSB** w.r.t. different numbers of points is shown in Fig. 6.11. It is the most time consuming process in pose refinement. Processing the TU Delft test dataset costs about 34 seconds for more than 8 million points, with 10 patches in a single batch. The computational complexity is $O\left(m^2\right)$ w.r.t. m the number of patches.

6.3.3. Overview of Pose Refinement Results

After training and validation, we estimate pose parameters for each corresponding pair of place recognition results based on a trained neural network for pose refinement. We transform **TLS** scans of place recognition results by a transformation matrix computed by the extracted pose parameters. Overview of pose refinement results are shown in Fig. 6.12 (More results are shown in Fig. A.3 in Appendix A), connected to place recognition results in Fig. 6.8. All corresponding pairs have accurate results (the rotation angle w.r.t. the z axis is smaller than 1 degree and the horizontal translation offset is smaller than 1 m) after pose refinement in comparison to place recognition results. It proves the potential of our registration neural network given an initial alignment of place recognition, where point clouds from **TLS** and **MLS** are close. Although **TLS** scans and **MLS** scenes are collected at different times and contain a lot of points of planes e.g. grounds, facades, we can realize pose refinement as long as patch corresponding pairs are correctly estimated, where the rotation angle w.r.t. the z axis between transformed **TLS** scans and corresponding **MLS** scenes is smaller than 30 degrees and translation offsets are not obvious (smaller than 5 meters) after place recognition.

6.3.4. Time Series of Test

For both the TU Delft and the Shanghai test dataset, the rotation and the translation range are more complicated and more flexible than place recognition results, which is a more general case without using any previous results. Fig. 6.13 shows an example of a time series of tests in different epochs with and without global refinement, together with ground truth as the reference. It indicates that for a single patch, after prediction the source is not well aligned to the template (rotation angle is obvious), but the global refinement improves the prediction a lot (almost no rotation angle). More experiments using the test dataset to evaluate different blocks of the patch-based registration neural network are given in Section 6.3.5 and Section 6.3.6. Quantitative results of the test dataset together with some traditional registration methods are shown in Section 6.3.7 and Section 6.3.8.

6.3.5. Evaluation of the Correspondence Search Block (CSB)

For the TU Delft validation holdout set, the precision and recall w.r.t. batch or patch are good (precision > 95% and recall ~ 50 %) if we select $T_{L_2} = 100$. In order to evaluate the performance of precision and recall based on the TU Delft test dataset, apart from the case of one-to-one corresponding between source and template patches, for each template batch we randomly add (2, 4, 6, 8, 10) patches from other template batches. These random patches do not have corresponding source patches in a single batch. They are errors during corresponding pair estimation and we see their impact on **CSB** in our neural network. As Fig. 6.14 shows, we randomly add 2 patches to a batch with 5 patches, then the source patch 2 is incorrectly matched to one of the random patches because of a lower feature distance of the wrong pair than the true pair. Results of **PR** curves for different numbers of random patch are shown in Fig. 6.15 for the TU Delft test dataset. The precision w.r.t. patch and batch at $T_{L_2} = 100$ decrease a lot, ~ 10% and ~ 18% decrease in comparison to the TU Delft validation holdout set, respectively. Still the recall is above 40%. Both patch and batch **PR** curves for different numbers of random patch show an approximate lineal trend between precision and recall. A randomly added patch leads to a decrease of about 1.1% in patch precision and about 2.2% in batch precision. This decrease is caused by the similarity between a random patch and a true corresponding template patch e.g. the similar appearance of the ground truth. The recall is less affected by random patches. In an urban area, there are many similar patches generated from canopies, buildings and roads. In order to decrease the impact of random patches on patch and batch precision, we need to avoid similar patches to be appeared in the same batch. For the Shanghai test dataset, both precision and recall are above 90% w.r.t. patch or batch at proper T_{L_2} (e.g. $T_{L_2} = 40$) as shown in Fig. A.1 in Appendix A, which indicates that point clouds collected by the same system (**MLS** system) are less noisy thus the trained network is more reliable.

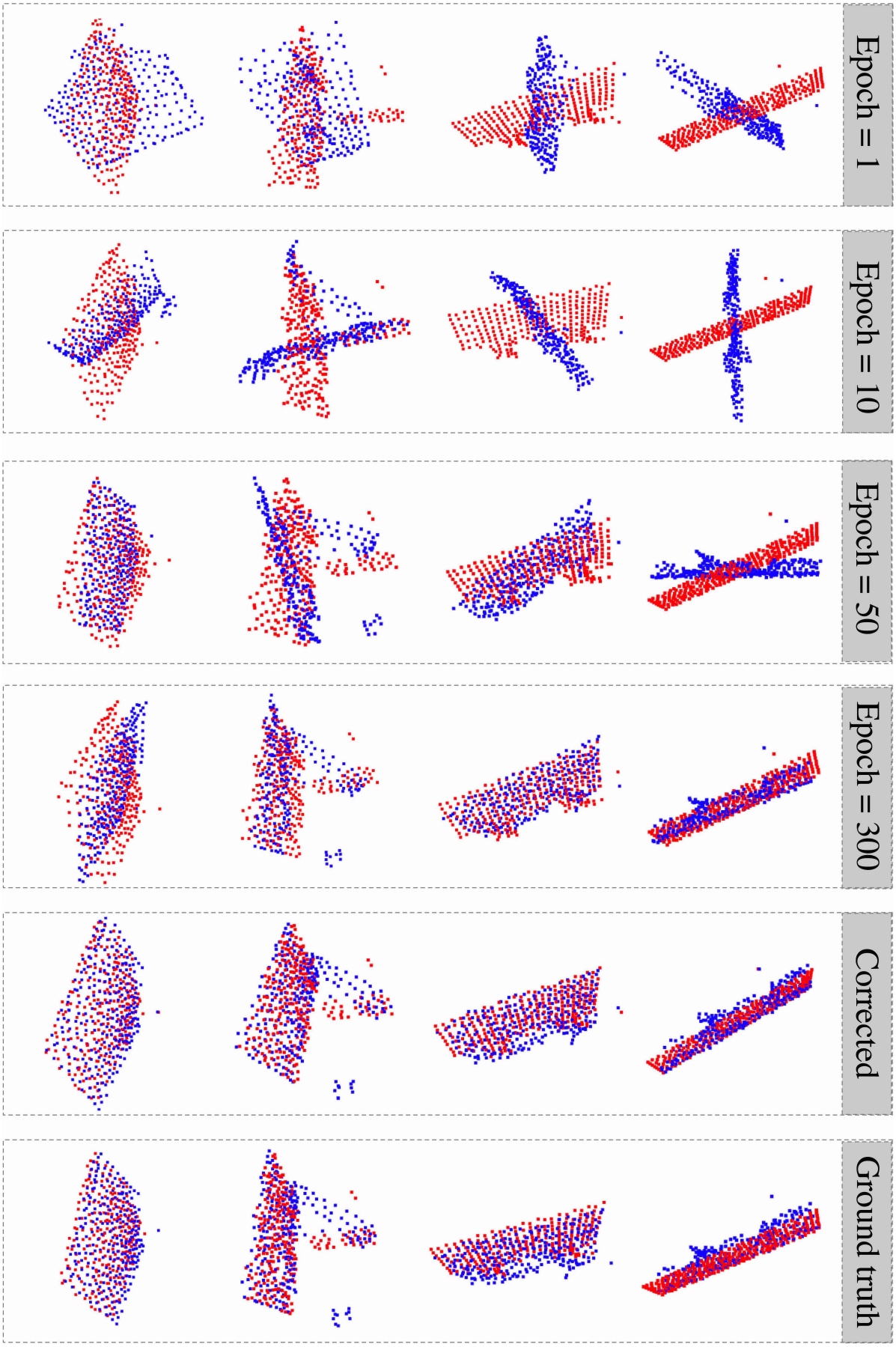


Figure 6.13: Outputs of test in 4 different epochs (1, 10, 50, 300) for four example patches, with corrected results after global refinement over these 4 patches, and ground truth. During test, after transforming source patches with the predicted rotation matrix and the predicted translation vector, the transformed source patch (red) and corresponding template patch (blue) is getting closer with the increase of epoch. In epoch 300, there is small offset between transformed source patches and corresponding template patches. The global refinement makes results almost the same as ground truth.

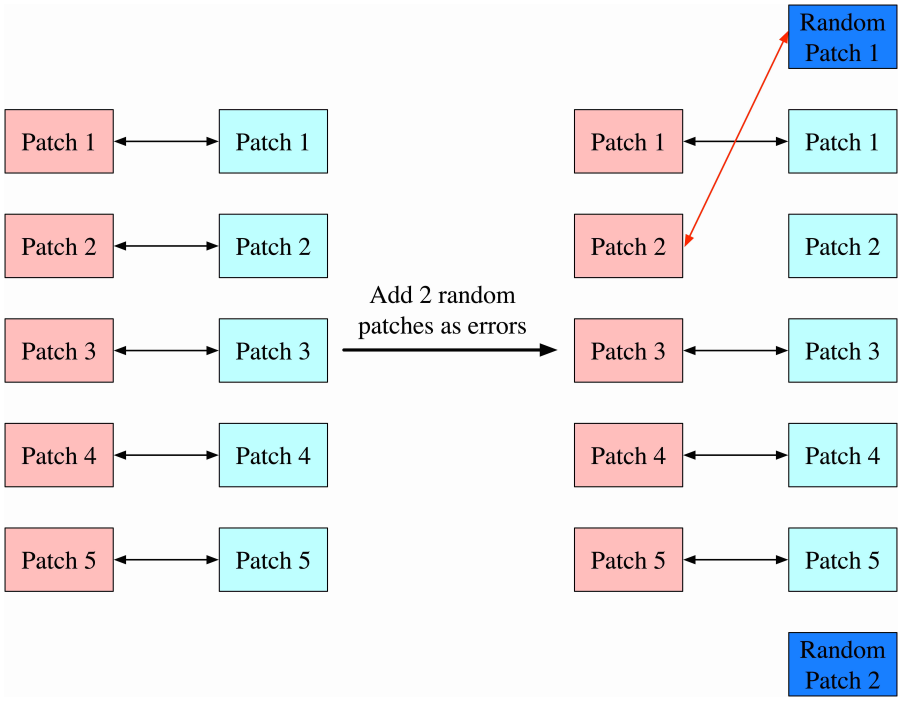


Figure 6.14: An example of adding 2 random patches as errors to a batch with 5 patches. The true corresponding pair of source patch 2 (red) and template patch 2 (cyan) becomes a wrong corresponding pair of source patch 2 (red) and random patch 1 (blue) because of a lower L_2 feature distance between source patch 2 and random patch 1.

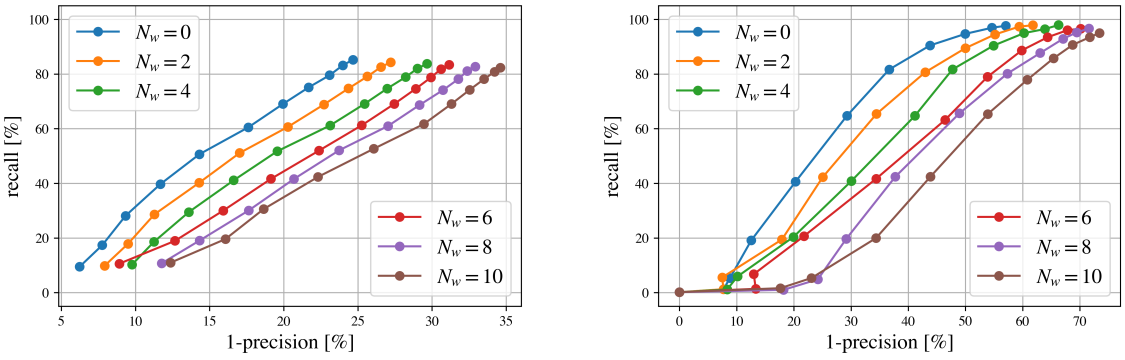


Figure 6.15: Precision-recall (**PR**) curves computed for the TU Delft test dataset from $T_{L_2} = 80$ to $T_{L_2} = 130$ (bottom to top), with N_w random patches added. $N_w = 0$ indicates source patches and template patches are one-to-one corresponding. (a) **PR** curves w.r.t. patch (b) **PR** curves w.r.t. batch

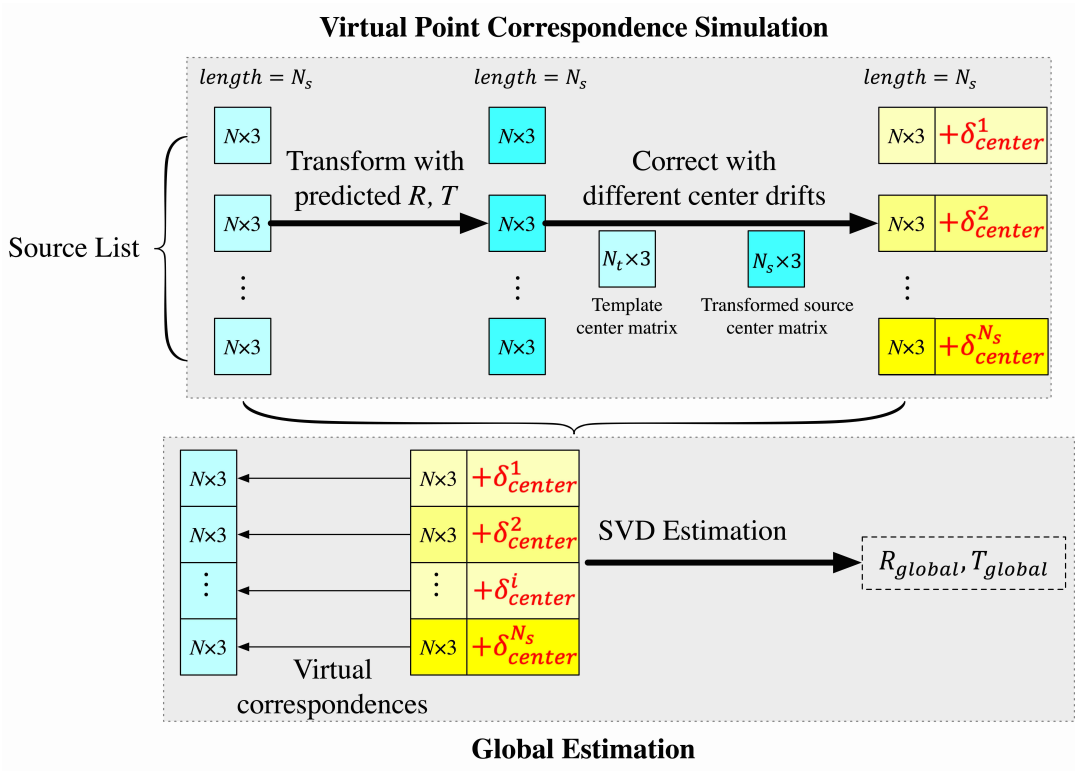


Figure 6.16: Simulated errors expressed as center offsets (red) for source patches during the virtual point correspondence simulation in the global refinement

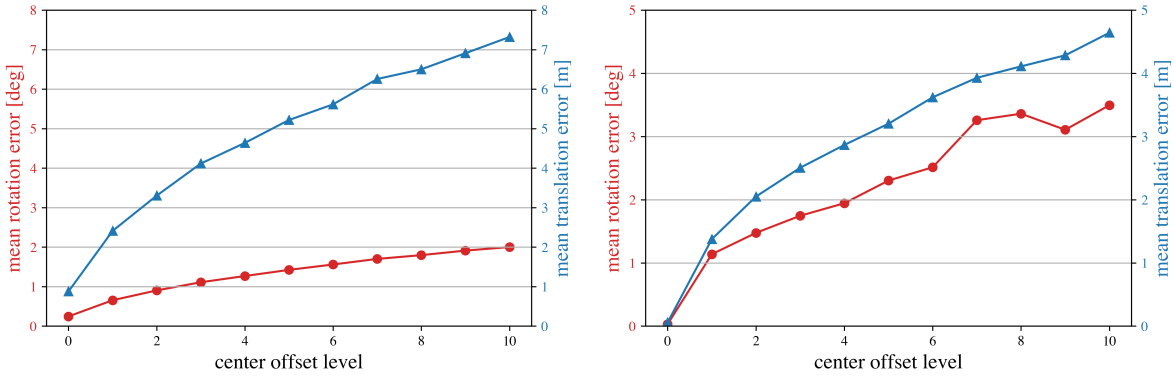


Figure 6.17: Mean error of rotation and translation for different center offset levels for (a) the TU Delft test dataset (b) the Shanghai test dataset

6.3.6. Evaluation of the Pose Estimation Block (PEB) & Global Prediction Refinement

CSB estimates corresponding pairs between source and template patches. In this experiment we assume there are 10 successfully estimated corresponding pairs. We cannot add errors to **PEB** as it is pre-trained. Errors are simulated during the virtual point correspondence simulation in the global refinement introduced in Section 5.4, by which we evaluate the impact of errors on the global refinement for predicted results of **PEB**. As Fig. 6.16 shows, the list of n center offsets $\boldsymbol{\delta}^n_{center}$ for n source patches is defined in Eq. (6.6):

$$\begin{aligned} \boldsymbol{\delta}^n_{center} &= [\delta^1_{center},\ \delta^2_{center},\ ...\delta^n_{center}] \quad 0 \le n \le N_s \\ \delta^i_{center} &= \left(\Delta c^i_x,\ \Delta c^i_y,\ \Delta c^i_z\right) \quad 1 \le i \le n \end{aligned} \tag{6.6}$$

where $0 \le \Delta c^i_x \le 10$ m, $0 \le \Delta c^i_y \le 10$ m, $0 \le \Delta c^i_z \le 6$ m and N_s is the number of patches in the source list.

The length of the center offset list n is shown as the n center offset level as well. The mean pose error of 0 to 10 center offset level are shown in Fig. 6.17. The Shanghai test dataset have a smaller mean translation error but a larger mean rotation error than the TU Delft test dataset at the same center offset level. During the global refinement process, the correction of center drift strongly impacts the improvement of predicted results. In this experiment, we prioritize on the mean translation error, which indicates a neural network that is more robust to the mean translation error is more reliable during the global refinement. The rate between maximum mean rotation error and the rotation range w.r.t. the z axis is about $\frac{1}{15}$ and $\frac{1}{45}$ for the TU Delft and the Shanghai test dataset, respectively. This rate indicates that the fluctuation of the Shanghai test dataset is smaller than the TU Delft test dataset. The larger mean rotation error of the Shanghai test dataset is possibly caused by the larger rotation range shown in Table 6.2. Overall, each increase of center offset leads to an increase of about 0.65 m and 0.45 m of the mean translation error, and 0.2 deg and 0.35 deg of the mean rotation error for the TU Delft test dataset and the Shanghai test dataset, respectively. The Shanghai test dataset has a better performance during the evaluation of **PEB** & the global refinement, as it always has the smaller mean translation error. The Shanghai dataset only uses **MLS** point clouds, so source and template patches are less noisy. If we want to improve the performance of **PEB** & the global refinement for the TU Delft test dataset, input patches need to be filtered to make them less noisy and close between the source and the template.

Table 6.5: Parameter selection for traditional registration methods. ICP_P2Po: point to point iterative closest point (**ICP**) registration; ICP_P2Pl: point to plane **ICP** registration; FPFH+RANSAC: random sample consensus (**RANSAC**) registration based on fast point feature histogram (**FPFH**) matching; CPD: rigid registration based on Gaussian mixture model (**GMM**) probabilistic estimation.

Method	Setting
ICP_P2Po & ICP_P2Pl	downsampling voxel size: 1.0 m source batch size: ~35,000 points template batch size: ~65,000 points maximum correspondence points-pair distance: 5.0 m maximum number of iterations: 20000 number of neighbors for normal computation: 10 relative root mean square error (**RMSE**): 1.0×10^{-6}
FPFH+RANSAC	number of neighbors for **FPFH** feature extraction: 30 number of correspondences to fit **RANSAC**: 4
CPD	source batch size: 2560 points template batch size: 2560 points relative difference Δ of the objective function in Algorithm 4: 0.1 maximum number of iterations: 20

6.3.7. Error Analysis of Registration Neural Networks & Traditional Methods

We choose four classical registration methods varying from point-based, plane based, feature based and probability based, with settings shown in Table 6.5. We also include two types of our network corresponding to two sets of balancing factors (α, β, γ) in Table 6.2, to give results with and without global refinement, shown as predicted network A/B, corrected network A/B, respectively.

Table 6.6: Error analysis of the TU Delft test dataset

Method	Mean rotation error ϵ_r (deg)	Mean translation error ϵ_t (m)	Max rotation error η_r (deg)	Max translation error η_t (m)	Run time on 8×10^6 points (sec)
ICP_P2Po	0.40	1.94	**0.40**	**1.95**	380.16
ICP_P2Pl	0.34	1.76	**0.35**	**1.76**	179.27
FPFH+RANSAC	/	/	/	/	922.76
CPD	**0.27**	**0.95**	1.09	5.02	> 1000
Predicted Network A	8.16	9.40	8.81	36.02	**26.00**
Predicted Network B	6.66	13.66	7.43	24.94	**26.33**
Corrected Network A	**0.25**	**0.88**	1.26	4.07	**36.97**
Corrected Network B	**0.24**	**0.88**	1.26	4.06	**35.98**

We compute five metrics: the mean and maximum error in rotation and translation, as well as the run time of processing 8 million points for the test dataset, shown in Table 6.6. Corrected Network A & B have the lowest mean rotation (0.24 deg) and translation (0.88 m) error, with satisfying run time (~4.5 seconds per million points). **CPD** also has small mean pose error (0.27 deg in rotation and 0.95 m in translation) but it is time consuming (> 100 seconds per million points). It is a method that is properly applied for a small sample e.g. cylinders of a **TLS** scan and corresponding **MLS** scene during place recognition. Two **ICP** methods perform well at the max pose error (0.35 deg in rotation and 1.76 m in translation). Point to plane **ICP** registration saves a lot time compared to point to point **ICP** registration (~ 200 seconds are saved during processing 8 million points). In terms of the failure of **FPFH+RANSAC**, it is due to the fact that our voxel cell of downsampling (1.0 m) is too large to extract **FPFH** features, then extracted **FPFH** features are not reliable to do registration. **FPFH+RANSAC** is time consuming (> 100 seconds per million points), so there is no need to repeat this experiment using dense point clouds e.g. point clouds at a 0.05 m voxel cell. The **FPFH** feature extraction also indicates our cylinder feature extraction in place recognition saves time (~ 15 seconds per million points). Although Predicted Network A/B is the most efficient method (26 seconds of processing 8 million points), the mean pose error is relatively large (> 5 deg in rotation and > 5 m in translation) compared to other methods. This large pose error also indicates the point-based neural network is hard to achieve a satisfying result (the mean pose error of < 1 deg in rotation and < 1 m in translation) for point cloud registration at a large scale e.g. an urban environment. Overall, the **ICP** registration performs better at the max pose error, and a corrected neural network has a better result of the mean pose error, with high efficiency. For the Shanghai test dataset, the mean and the max pose error are all small for our neural network (0.03 deg of rotation and 0.06 m of translation in the mean pose error, 0.14 deg of rotation and 0.34 m of translation in the max pose error), but **ICP** and **CPD** registration methods cannot work because of the large rotation range w.r.t. the z axis (0 to π), as shown in Table 6.2 and Table A.4 in Appendix A.

6.3.8. Comparison between Patch-based Neural Network and ICP w.r.t. Initial Poses

An initial transformation is applied in place recognition as shown in Fig. 6.8. The patch based neural network works well on results of a small rotation angle (< 30 degrees) w.r.t. the z axis between transformed **TLS** scans and corresponding **MLS** scenes, as Fig. 6.12 shows. In order to test our neural network at a larger rotation angle w.r.t. the z axis, we design an experiment of the mean pose error vs. different initial angles. An initial angle is the median value of a certain rotation range w.r.t. z axis between **TLS** and **MLS**. A list of initial angles $\boldsymbol{\psi}_t^{0,\lambda}$ are generated sequentially by a rotation range from 0 to λ, uniformly divided into t bins, as shown in Eq. (6.7). Fig. 6.18 gives an example of two initial angles where $\lambda = 2\pi$ and $t = 8$.

$$\begin{aligned} \boldsymbol{\psi}_t^{0,\lambda} &= [\psi_1,\ \psi_2,\ ...\psi_t] \\ \psi_i &= (2i-1)\cdot\lambda/2t \quad 1 \le i \le t \end{aligned} \tag{6.7}$$

For the TU Delft test dataset, we create 16 uniform bins under the rotation range of $(0, \pi/6)$ w.r.t. the z axis, with 16 initial angles computed. Batches are reorganized based on the rotation angle w.r.t. the z axis between **TLS** and **MLS**, and there are about 100 test batches within each bin. A point-to-point iterative closest point (**ICP**) registration method ICP_P2Po is applied as the comparison. Results are shown in Fig. 6.19. The standard deviation (**STD**) are computed based on these results to show the stability of our neural network

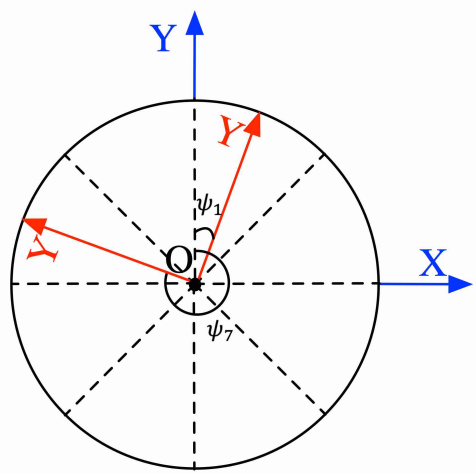


Figure 6.18: An example of two initial angles between **TLS** (red) and **MLS** (blue) under a rotation range from 0 to 2π w.r.t. the z axis, uniformly divided into 8 bins. $\psi_1 = \pi/8$, $\psi_7 = 13\pi/8$. (Simply only the y axis is shown for two different **TLS** coordinate systems)

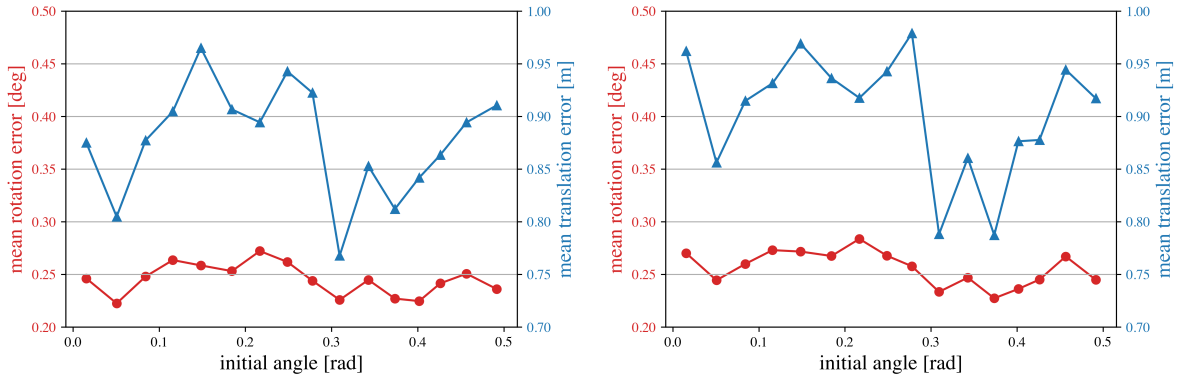


Figure 6.19: Mean error of rotation and translation for different initial angles for the TU Delft test dataset, with 16 uniform bins over $(0, \frac{\pi}{6})$ w.r.t. the z axis. (a) Results of our network, the standard deviation (**STD**) of the mean pose error is 0.014 deg in rotation and 0.050 m in translation, respectively. (b) Results of ICP_P2Po, the **STD** of the mean pose error is 0.016 deg in rotation and 0.057 m in translation, respectively.

and ICP_P2Po. The **STD** of the mean pose error for 16 initial angles for our network and for ICP_P2Po are similar, 0.014 deg vs. 0.016 deg in rotation and 0.050 m vs. 0.057 m in translation, respectively. These two closeby **STD**s indicate that under small rotation angles (smaller than 30 degrees), both our neural network and ICP_P2Po have the stable performance, with no more than 1.0 m of the mean translation error and no more than 0.3 deg of the mean rotation error.

For the Shanghai test dataset, the rotation range w.r.t. the z axis is $(0, \pi)$ shown in Table 6.2. We create 10 uniform bins and repeat the experiment, and results are shown in Fig. 6.20. The **STD** of the mean pose error for our network is still small, 0.0012 deg in rotation and 0.0042 m in translation, which indicates that results of our neural network are stable, with no more than 0.07 m of the mean translation error and no more than 0.03 deg of the mean rotation error. But results of ICP_P2Po fluctuate a lot in different initial angles. **STD**s are 0.020 deg in rotation and 0.043 in translation, more than 10 times larger than **STD**s of our network. The mean pose error peaks at the initial angle of $\sim \pi/2$, with about 0.080 deg in rotation and 0.18 m in translation. Overall, results for the TU Delft and for the Shanghai test dataset prove that our neural network also works stably at a large rotation range e.g. $(0, \pi)$, while the **ICP** method for registration only works stably at a small rotation range e.g. $(0, \pi/6)$ and is more sensitive with the increase of ψ, the initial angle, especially when $\psi > \pi/6$. Results from these 2 test datasets in Table 6.6, Table A.4 in Appendix A and Fig. 6.20 also prove that our neural network works at a large rotation range (0 to π) with or without reorganization under different initial angles, but **ICP** registration methods need reliable prior information (an initial transformation) to make the source close to the template, similar to what we did in place recognition, otherwise these registration methods do

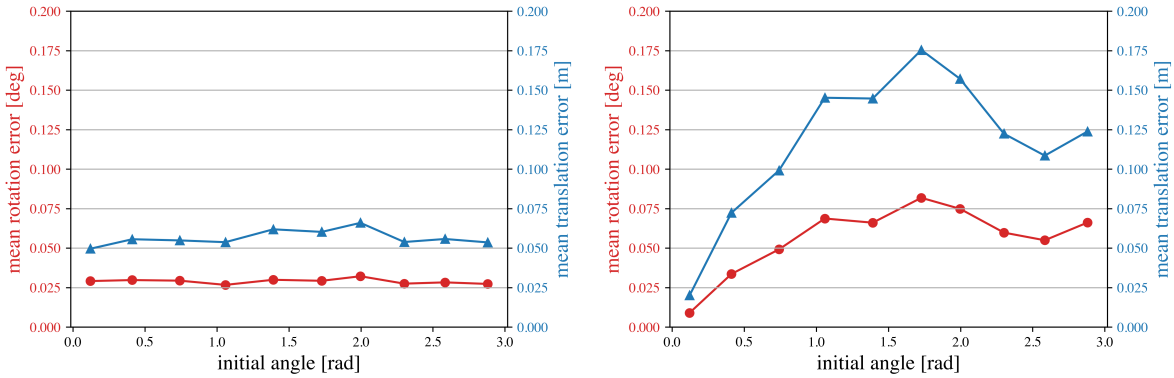


Figure 6.20: Mean error of rotation and translation for different initial angles for the Shanghai test dataset, with 10 uniform bins over $(0, \pi)$ w.r.t. the z axis. (a) Results of our network, the standard deviation (**STD**) of the mean pose error is 0.0012 deg in rotation and 0.0042 m in translation, respectively. (b) Results of ICP_P2Po, the **STD** of the mean pose error is 0.020 deg in rotation and 0.043 m in translation, respectively.

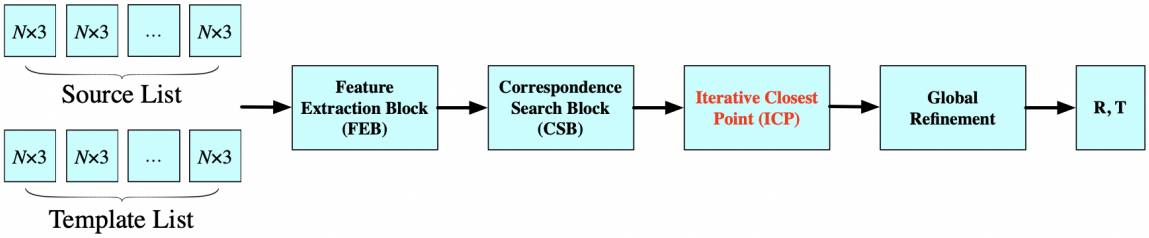


Figure 6.21: The ablation study of the contribution of **CSB** and global refinement, where **PEB** is replaced by a point-to-point **ICP** registration method (red).

not work well, even cannot work.

6.3.9. Ablation Study

We design an ablation study to show the contribution of **CSB** and global refinement. As Fig. 6.21 shows, for each source patch, we assume its correspondence in the template is correctly estimated in **CSB**. Instead of applying **PEB**, we use ICP_P2Po to realize registration between each patch corresponding pair. Global refinement is applied over all pairs finally. The result of this ablation study is shown in Table 6.7, where both mean and max pose error are close to Corrected Network A, which proves the contribution of **CSB** and global refinement. Still it takes more time than our network, it is faster than the ICP_P2Po method in Table 6.6.

Table 6.7: Contribution of the correspondence search block (**CSB**) and global refinement for the traditional registration method ICP_P2Po. Each source batch or template batch consists of 10 patches and 2560 points. Maximum correspondence points-pair distance during ICP_P2Po is adjusted to 0.2 m. Other settings are the same as Table 6.5.

Method	Mean rotation error ϵ_r (deg)	Mean translation error ϵ_t (m)	Max rotation error η_r (deg)	Max translation error η_t (m)	Run time on 8×10^6 points (sec)
ICP_P2Po	0.26	0.90	1.49	4.08	84.39
Corrected Network A	0.25	0.88	1.26	4.07	**36.97**

6.4. Application Scope

Both place recognition and pose refinement have a proper scope under certain circumstances. In general, if we have a reliable reference **MLS** map and several **TLS** scans of good quality, the 3D point cloud localization framework can work well. Place recognition is possibly applied in other scenarios e.g. an indoor environment

as long as there are object features that can be efficiently extracted (within 50 seconds per million points). Pose refinement is reliable when patch corresponding pairs are accurately estimated (precision larger than 80%).

6.4.1. Place Recognition Scope

In detail, there are some characteristics, preconditions and limitations for place recognition:

1. **One-to-many solutions**. As there are several **TLS** scans and typical **MLS** scenes, for each scan and each scene we always compute a similarity and a mean distance based on **GMM** probabilistic estimation. Due to the characteristics of the urban environment, there are similarities in different areas. The probabilistic estimation may fail if there are two or more similar scenes e.g. ambiguities in Table 6.4. A decision making strategy is needed to decrease the risk of failure during corresponding pair estimation;

2. **Dominant pair(s)**. During decision making, some true corresponding pairs are reliable enough to enable us to decide them in priority. Then we gradually decide solutions for other ambiguous pairs. For example, the computed similarity of (TLS_5, MLS_5) is much larger than other pairs for TLS_5 in our experiment, thus a reliable result is extracted;

3. **Efficiency prioritization**. In general our place recognition method based on cylinders is more efficient (15 seconds per million points) in comparison to other place recognition methods based on fine features e.g. point-based features (more than 100 seconds). It is relatively time consuming to extract features from the **MLS** point clouds, with maximum more than 5 times longer than extracting features from **TLS** data. Fortunately, in reality **MLS** data is usually acquired before **TLS** data. We can apply feature extraction for **MLS** scenes in advance. A feature extraction method e.g. cylinder feature extraction on a segmented map for **MLS** data ensures more reliable results without merging or filtering. In some small scenarios e.g. an indoor environment, it is more efficient to process **MLS** data due to a smaller number of points than urban point clouds.

6.4.2. Pose Refinement Scope

Characteristics, preconditions and limitations of pose refinement are discussed as following:

1. **Rigid point clouds**. In our research we assume registration between **TLS** and **MLS** point clouds is rigid, so point clouds should be collected under a good weather condition. Overlap between **TLS** and **MLS** data should be large enough (larger than 50%) to enable registration.

2. **Tuning of the correspondence search block (CSB)**. When we choose different datasets or hyperparameters for training, the proper threshold T_{L_2} in **CSB** is various. As a result, it is a prerequisite to set aside a validation holdout set for tuning **CSB** after training, which costs extra time (several seconds) but is useful.

3. **Dominant patch(es) within a batch**. Some patches in a single batch are well kept for appearance in both **TLS** and **MLS** data. In **CSB**, these patch pairs have a low feature distance ($T_{L_2} < 50$). In the pose estimation block (**PEB**) & global refinement, pose parameters are accurately retrieved. Manual processing is also helpful to ensure more patches are dominant. For example, when we generate a list of patches alongside a road, patches will be similar. If we manually separate them in different batches, similar patches are avoided to appear in the same batch, thus the precision of **CSB** increases.

4. **PEB not accurate enough**. From Table 6.6 we find the output of our neural network without global refinement is still unreliable (several degrees/meters offset), showing the limitation of current point-based networks. If we want to extend the patch-based neural network to a wider scale, still it is important to improve the performance of prediction without global refinement, since the virtual point correspondence simulation highly depends on the predicted transformation matrix, as Eq. (5.17) shows.

5. **Robust neural network**. Our patch-based registration neural network works at different initial angles (0 to π) without any prior information. Different blocks inside our neural network are reliable if there are not many errors (1 or 2 wrong patches or center offsets added). It indicates the potential that if we do not apply an initial transformation in place recognition (only corresponding pairs are estimated), still we can realize pose refinement based on our neural network together with global refinement.

7

Conclusions & Recommendations

This chapter firstly answers the research questions as introduced in Chapter 1 in Section 7.1. Then recommendations for future work are provided for both place recognition and pose refinement in Section 7.2.

7.1. Conclusions

In Section 1.6, the main research question is introduced:

How to realize localization in the urban environment, by means of an innovative registration neural network and by traditional methods?

In summary, we divide point cloud localization in the urban environment between static terrestrial laser scanning (**TLS**) scans and a mobile laser scanning (**MLS**) reference map into two steps: **place recognition** and **pose refinement**, based on the characteristic of the larger coverage of the **MLS** reference map in comparison to the static **TLS** scans, and the prerequisite of large overlap between the source and template in point cloud registration. For **place recognition**, we realize it by firstly designing a cylinder-like object feature descriptor to efficiently represent both **TLS** and **MLS** point clouds by several hundreds of cylinders rather than millions of points, then applying probabilistic estimation based on a Gaussian mixture model (**GMM**) to form corresponding pairs between **TLS** scans and **MLS** scenes. A transformation is applied to align a **TLS** scan in a loose sense to its corresponding **MLS** scene. **Pose refinement** is realized starting from the results of **place recognition**, by designing a neural network consisting of 3 blocks for feature extraction, correspondence search and pose estimation (registration) respectively, to predict 6 pose parameters ($(\theta_x,\ \theta_y,\ \theta_z,\ \delta_x,\ \delta_y,\ \delta_z)$ in Eq. (2.1)) initially, and tune the predicted results with a global refinement process based on points in the source and their virtual corresponding points in the template. Six sub-questions are proposed to give details for the main question.

1. **How to organize the input data of large size?**
   In Chapter 3 we introduced the overall workflow of our research, addressing preprocessing methods and different data structures w.r.t. different purposes, as Fig. 3.1 shows. In order to deal with the large amount of input data, **TLS** and **MLS** point clouds are firstly preprocessed by voxel downsampling and statistical outlier removal. Particularly, the **MLS** point clouds are manually divided into several typical scenes. Then both **TLS** and **MLS** point clouds have identical point density. In place recognition, several resampling methods based on certain data structures and corresponding algorithms are applied to extract features from a **TLS** scan or a **MLS** scene, representing millions of points with hundreds of cylinders finally. After place recognition, **TLS** scans and their corresponding **MLS** scenes have smaller size with large overlap. In pose refinement, patches are generated from **TLS** scans and their corresponding **MLS** scenes, and we use a certain number of patches for a single batch during training. For a single batch we predict a set of pose parameters under a pre-trained network, then tune it by global refinement.

2. **What is the workflow of place recognition and pose refinement?**
   The workflow for place recognition and pose refinement has been described in Chapter 4 and Chapter 5, respectively. In place recognition, there are two main processes: feature extraction and corresponding

pair estimation. During the feature extraction process, we design a reliable feature descriptor tailored to the characteristics of the urban environment. The feature descriptor is designed in steps. During the corresponding pair estimation process, we firstly compute a metric based on extracted features to describe the similarity between a **TLS** scan and a **MLS** scene. Then the metric is used to form corresponding pairs between **TLS** scans and **MLS** scenes, under a decision making strategy. In pose refinement, there are also two main processes: neural network design and global refinement. The neural network is designed to contain several blocks for different purposes e.g. feature extraction, pose estimation. Next we select proper hyperparameters to do training, validation and testing based on the defined loss function. During the global refinement, we design a global correction method for predicted results of the neural network, to improve the accuracy of estimated pose parameters.

3. **How to realize place recognition in a relatively simple way?**
   This question is answered in Chapter 4. We choose cylinder-like objects as our feature descriptor and a **GMM** probabilistic framework to form corresponding pairs between **TLS** scans and **MLS** scenes. A cylinder-like object is stable and general in the urban environment as long as the terrain does not change a lot. A cylinder-like object is easy to retrieve starting from a simple circle equation in 2D space. The impact of different types of error is mitigated by several processing methods. A **TLS** scan or a **MLS** scene usually contains several hundreds of cylinder-like objects. This number is much smaller than the number of points (several millions), so it enables us to apply computationally less efficient estimation methods. A **GMM** probabilistic framework is suitable to process data of a small size, especially when we do not know which part of the given data belongs to which part in the whole data population. The given data here consists of our **TLS** scans and the overall data population consists of all **MLS** scenes. The cylinder-like object descriptor is coarsely retrieved, but well preserves the information of **TLS** scans and **MLS** scenes. As a result, the **GMM** probabilistic framework is a good choice to realize place recognition based on extracted cylinder features, especially when followed by the designed proper decision making strategy to improve the robustness.

4. **How to design our registration neural network and select proper hyperparameters during pose refinement?**
   Chapter 5 gives details on the structure of our registration neural network. We design a patch based registration neural network, still with a point-based neural network as kernel. The whole patch based registration neural network consists of three main blocks: the feature extraction block (**FEB**), the correspondence search block (**CSB**), and the pose estimation (registration) block (**PEB**). Particularly, **FEB** retrieves two types of feature, invariant feature and general feature, as the input of **CSB** and **PEB**, respectively. **CSB** is designed to form patch corresponding pairs before **PEB**. **PEB** predicts initial results of pose parameters. The patch based neural network enables point cloud registration at a large scale, but leads to less accurate output (several degrees/meters offset). We apply a global refinement method by simulating virtual corresponding points for patches from **TLS** data, based on predicted results from **PEB**, then by globally estimating pose parameters based on points from different patches and their virtual corresponding points. Deep learning process consists of training, validation, test under defined loss function definition. We choose proper hyperparameters based on different pre-trained networks under the defined loss function and the validation dataset.

5. **What is the performance of place recognition and pose refinement?**
   In Chapter 4 and Chapter 5 we gave examples of place recognition and pose refinement. In Chapter 6 we quantitatively assessed the performance of both place recognition and pose refinement, as well as visualized the output. In terms of place recognition, extracted cylinder-like objects cover and well represent the area of **TLS** or **MLS** point clouds. Feature extraction is the most time consuming process during place recognition (about 15 seconds per million points), especially for **MLS** data, but we efficiently use about 100 cylinders per million points for later processing rather than original data of millions of points. Under a decision making strategy based on computed similarities and mean distances, all of 5 corresponding pairs between **TLS** scans and **MLS** scenes are accurately retrieved. Both ambiguities and outliers are removed. An initial transformation is applied for each corresponding pair to make the source and the template close (rotation angle smaller than 30 degrees). In pose refinement, patches under a voxel cell (5 m, 5 m, 3 m) are created from place recognition results as the input of the neural network for registration. After the training process, **CSB** in the neural network is necessarily tuned by the validation holdout set to decide the proper threshold e.g. 100 in our experiments based on precision-recall (**PR**) curves. Random wrong patches impact on precision of the **CSB**, leading to 1% to 2 % decrease per added wrong patch when we use the TU

Delft test dataset. There is no influence on recall caused by wrong patches. The initial output of the patch based neural network is not good (several degrees/meters offset), but the global refinement process improves the accuracy remarkably, from ~ 1 deg to ~ 0.1 deg in the mean rotation error and ~ 1 m to ~ 0.1 m in the mean translation error. Results of place recognition are well tuned by pose refinement (the rotation angle is smaller than 1 deg and the translation offset is smaller than 1 m). The impact of center offset on the global refinement depends on the quality of input data, less noisy data e.g. the Shanghai test dataset that only uses **MLS** point clouds will train a more robust network.

6. **What are improvements of our neural network in comparison with current neural networks and traditional methods?**
   During the evaluation and analysis of pose refinement in Chapter 6, we compared results of our neural network with the current point-based neural network, and some traditional methods e.g. iterative closest point (**ICP**) and coherent point drift (**CPD**). For the current point-based neural network, results are not good with nearly 10 deg and 10 m of the mean pose error in rotation and translation. Results of a point-based neural network cannot be improved by the global refinement, because a point-based neural network focuses on a small scale, mostly inputting one single simple object in the source and template within a batch. The source and the template always have the same class. The patch based neural network extends the point-based neural network at a small scale to a larger scale, which enables to input several patches at one batch and to output more accurate pose parameters (~0.1 deg and ~0.1 m of the mean pose error in rotation and translation) tuned by the global refinement. The appearance of input patches is flexible, which saves a lot of time to label input data before training. In comparison to traditional registration methods, applying a neural network is much more efficient (~ 35 seconds vs. more than 100 seconds for processing 8 million points) and does not need prior information (initial transformation) compared with **ICP**. A neural network can work on sparse point clouds (0.5 m voxel size) while traditional registration methods e.g. **RANSAC+FPFH** needs dense point clouds (0.05 m voxel size) for feature extraction. The accuracy of our patch based registration neural network is also better in terms of the mean pose error, with 0.25 deg in rotation and 0.88 m in translation in comparison to 0.27 deg to 0.40 deg in rotation and 0.95 m to 1.94 m in translation of some traditional registration methods e.g. **ICP** and **CPD**, when the rotation range is not large (smaller than 30 degrees). When the rotation range is large (from 0 to π), only our neural network can work without an initial transformation.

In general, we incorporate traditional methods, a hand-crafted feature descriptor and **GMM** probabilistic framework, with a novel registration neural network, patch based registration neural network to realize point cloud localization in the urban environment. Our **place recognition** method is efficient based on the common and stable cylinder-like object in the urban environment, and works on point clouds at different times (2016 and 2020) collected from different systems (**TLS** and **MLS**). During **pose refinement**, compared with traditional registration methods, the newly proposed neural network improves the accuracy from the aspect of the mean pose error, the efficiency based on the run time and the application scope from the aspect of the initial pose and point density. Our innovative registration neural network also extends the applications scope of the current point-based registration neural network to a larger scale.

7.2. Recommendations

There are several methods proposed in our research that can be further investigated and probably improved, divided from the aspect of **place recognition** and **pose refinement**.

7.2.1. Place Recognition

Fine Feature Descriptor

In our research, place recognition starts from point clouds without any processing based on only 3D coordinate information. Although we apply a 9-step feature extraction to design a cylinder-like object descriptor, this descriptor is coarse. In order to remove the impact of canopies, pillars on buildings and new trees, we apply merging and filtering several times, leading to some differences between filtered cylinder-like objects and the ground truth. Thus the **GMM** probabilistic framework and a decision making strategy are used. If a more fine feature descriptor can be extracted, e.g. cylinder-like objects are extracted from a well segmented map, then extracted cylinder-like objects are the same representation as the real world. We do not need to remove different types of error and the feature descriptor will be more reliable.

Complementary Information for Extracted Features
During the corresponding pair estimation in place recognition, we only input 2D coordinate information of extracted cylinder-like objects to a **GMM** probabilistic framework. Apart from 2D coordinates, if we can extract other reliable information, e.g. topological relationships to be attached for each cylinder-like object, corresponding pairs will be more reliable. Due to the small number of extracted cylinder-like objects, it is possible to apply probabilistic estimation by inputting data with more than a high (>10) dimension.

7.2.2. Pose Refinement

Threshold of Patch Correspondence Search
We apply a loose way of invariant feature extraction without T-net transformation before **CSB**. Thus we choose L_2 feature distance and **Softmax** function to output a probability for each patch pair of **TLS** and **MLS** data. After training, a threshold of feature distance is tuned by validation holdout set. One disadvantage of this tuning method is threshold variance. Depending on different input datasets and hyperparameter selections, the threshold is various and always needs to be tuned after training. If we can find some normalized metrics that are distinguishable enough to output a probability for each corresponding pair at a small value scale instead of the large scale of computed L_2 feature distance, we might be able to set a fixed threshold or scheduled threshold during training, to make the neural network more generally used.

Patch-based Network Tuning
In our experiments, wrong patch affects the precision of **CSB**, and center offset impacts the accuracy of pose parameters. The output of **PEB** is not good (several degrees/meters offset) even we assume all patch corresponding pairs are correct. It is hard to find out which part of the whole neural network goes wrong or needs to be improved, but there are some possible ways to improve the network from the aspect of input data and complementary information. In our research, input data is irregular patches with random appearance. One possible way is to attach segmentation information for each patch, then different segments will not be considered during correspondence search, thus to improve the precision of **CSB**. Though input patches are random in appearance, we can find a resampling method to make resampled patches more distinct in **FEB**, which is another possible way to improve the performance of **CSB**. As for the improvement of output of **PEB**, it is possible to add some complementary information e.g. features computed by traditional methods, to learned features, which can partially control the training process instead of a complete black box.

In summary, it is worth further study to make the 3D point cloud localization framework automatize in practice. In a constructed **MLS** reference map, preprocessing is applied and features are extracted in advance. When we collect several **TLS** scans (the number of scans should be larger than 3 in order to apply a global estimation), the same processing method is applied and place recognition is realized based on extracted features and a decision making strategy. **TLS** scans are coarsely aligned to corresponding areas in the **MLS** reference map. Next point clouds with overlap are resampled to general source and template patches from **TLS** and **MLS**, respectively. Patches are inputted to a trained neural network to predict 6 pose parameters, and the global refinement process tunes the prediction results to help us realize pose refinement. Based on place recognition and pose refinement results, we extract an overall transformation matrix between a **TLS** scan and the **MLS** reference map, and **TLS** point clouds can be well aligned to a big **MLS** point cloud finally.

A

Extra Experiments

This appendix gives some detailed results of experiments in Chapter 6.

Table A.1: Computed similarity ρ & mean distance μ **without line feature extraction** between 5 **TLS** static scans and 5 **MLS** scenes. Red represents similarities or mean distances below our threshold given in Section 4.4.4; blue means there are multiple possible matching pairs (computed similarities are close) between **TLS** and **MLS** data; green indicates a dominant solution (there is a computed similarity which is obviously larger than others) between **TLS** and **MLS** data.

(ρ, μ)	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	(0.68, 7.5)	(0.65, 7.4)	(0.69, 7.2)	(0.69, 9.1)	(0.79, 6.5)
TLS_2	(0.50, 14.0)	(0.62, 10.3)	(0.61, 9.8)	(0.51, 17.3)	(0.57, 10.4)
TLS_3	(0.61, 11.3)	(0.63, 8.8)	(0.69, 7.8)	(0.59, 12.0)	(0.69, 8.0)
TLS_4	(0.68, 7.1)	(0.69, 6.8)	(0.76, 5.9)	(0.85, 5.2)	(0.81, 5.2)
TLS_5	(0.68, 8.7)	(0.67, 7.8)	(0.68, 7.8)	(0.68, 8.6)	(0.78, 6.2)

Table A.2: Decision making process consisting of 6 steps based on Table 6.4, with the dominance Δ for each iteration in Algorithm 5 in Section 4.4.4, where a green value indicates a dominant solution and '$-$' indicates a pair that has been decided. x, 0 and 1 indicate an undecided pair, a not-matching pair and a matching pair, respectively.

Decision	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	x	x	x	x	x
TLS_2	0	0	0	0	0
TLS_3	0	x	x	0	x
TLS_4	x	x	x	x	x
TLS_5	x	x	x	x	x

(a) Initialization: outlier removal

Decision	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	x	x	0	x	0
TLS_2	0	0	0	0	0
TLS_3	0	0	1	0	0
TLS_4	x	x	0	x	0
TLS_5	0	0	0	0	1

(c) 2^{nd} decision making, $\mathbf{\Delta}_2$ = (0.005, 0, $-$, 0.006, 0.111)

Decision	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	x	x	0	x	x
TLS_2	0	0	0	0	0
TLS_3	0	0	1	0	0
TLS_4	x	x	0	x	x
TLS_5	x	x	0	x	x

(b) 1^{st} decision making, $\mathbf{\Delta}_1$ = (0.005, 0, 0.102, 0.006, 0.095)

Decision	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	x	x	0	0	0
TLS_2	0	0	0	0	0
TLS_3	0	0	1	0	0
TLS_4	0	0	0	1	0
TLS_5	0	0	0	0	1

(d) 3^{rd} decision making, $\mathbf{\Delta}_3$ = (0.037, 0, $-$, 0.107, $-$)

Decision	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	1	0	0	0	0
TLS_2	0	0	0	0	0
TLS_3	0	0	1	0	0
TLS_4	0	0	0	1	0
TLS_5	0	0	0	0	1

(e) 4^{th} decision making, $\mathbf{\Delta}_4$ = (0.037, 0, −, −, −)

Decision	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	1	0	0	0	0
TLS_2	0	1	0	0	0
TLS_3	0	0	1	0	0
TLS_4	0	0	0	1	0
TLS_5	0	0	0	0	1

(f) Correction based on prior information

Table A.3: Decision making process consisting of 6 steps based on Table A.1, with the dominance Δ for each iteration in Algorithm 5 in Section 4.4.4, where a green value indicates a dominant solution and '−' indicates a pair that has been decided. x, 0 and 1 indicate an undecided pair, a not-matching pair and a matching pair, respectively.

Decision	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	x	x	x	x	x
TLS_2	0	0	x	0	0
TLS_3	0	x	x	0	x
TLS_4	x	x	x	x	x
TLS_5	x	x	x	x	x

(a) Initialization: outlier removal

Decision	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	x	x	x	x	0
TLS_2	0	0	x	0	0
TLS_3	0	x	x	0	0
TLS_4	x	x	x	x	0
TLS_5	0	0	0	0	1

(b) 1^{st} decision making, $\mathbf{\Delta}_1$ = (0.096, 0, 0.004, 0.039, 0.097)

Decision	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	x	x	x	0	0
TLS_2	0	0	x	0	0
TLS_3	0	x	x	0	0
TLS_4	0	0	0	1	0
TLS_5	0	0	0	0	1

(c) 2^{nd} decision making, $\mathbf{\Delta}_2$ = (0, 0, 0.061, 0.088, −)

Decision	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	x	x	0	0	0
TLS_2	0	0	0	0	0
TLS_3	0	0	1	0	0
TLS_4	0	0	0	1	0
TLS_5	0	0	0	0	1

(d) 3^{rd} decision making, $\mathbf{\Delta}_3$ = (0.01, 0, 0.061, −, −)

Decision	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	1	0	0	0	0
TLS_2	0	0	0	0	0
TLS_3	0	0	1	0	0
TLS_4	0	0	0	1	0
TLS_5	0	0	0	0	1

(e) 4^{th} decision making, $\mathbf{\Delta}_4$ = (0.035, 0, −, −, −)

Decision	MLS_1	MLS_2	MLS_3	MLS_4	MLS_5
TLS_1	1	0	0	0	0
TLS_2	0	1	0	0	0
TLS_3	0	0	1	0	0
TLS_4	0	0	0	1	0
TLS_5	0	0	0	0	1

(f) Correction based on prior information

Table A.4: Error analysis of the Shanghai test dataset

Method	Mean rotation error ϵ_r (deg)	Mean translation error ϵ_t (m)	Max rotation error η_r (deg)	Max translation error η_t (m)	Run time on 5×10^6 points (sec)
ICP_P2Po	/	/	/	/	>1000
ICP_P2Pl	/	/	/	/	>1000
CPD	/	/	/	/	>1000
Predicted Network A	16.15	21.23	65.98	94.97	**17.26**
Predicted Network B	23.85	22.35	66.12	41.11	**16.97**
Corrected Network A	**0.03**	**0.06**	**0.14**	**0.34**	**23.42**
Corrected Network B	**0.03**	**0.07**	**0.18**	**0.36**	**23.72**

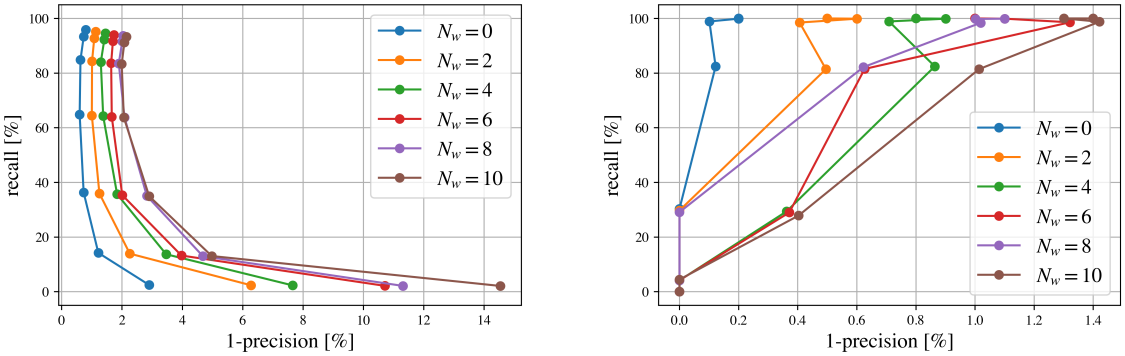


Figure A.1: Precision-recall (**PR**) curves computed for the Shanghai test dataset from $T_{L_2} = 10$ to $T_{L_2} = 40$ (bottom to top), with N_w random patches added. $N_w = 0$ indicates source patches and template patches are one-to-one corresponding. (a) **PR** curves w.r.t. patch (b) **PR** curves w.r.t. batch

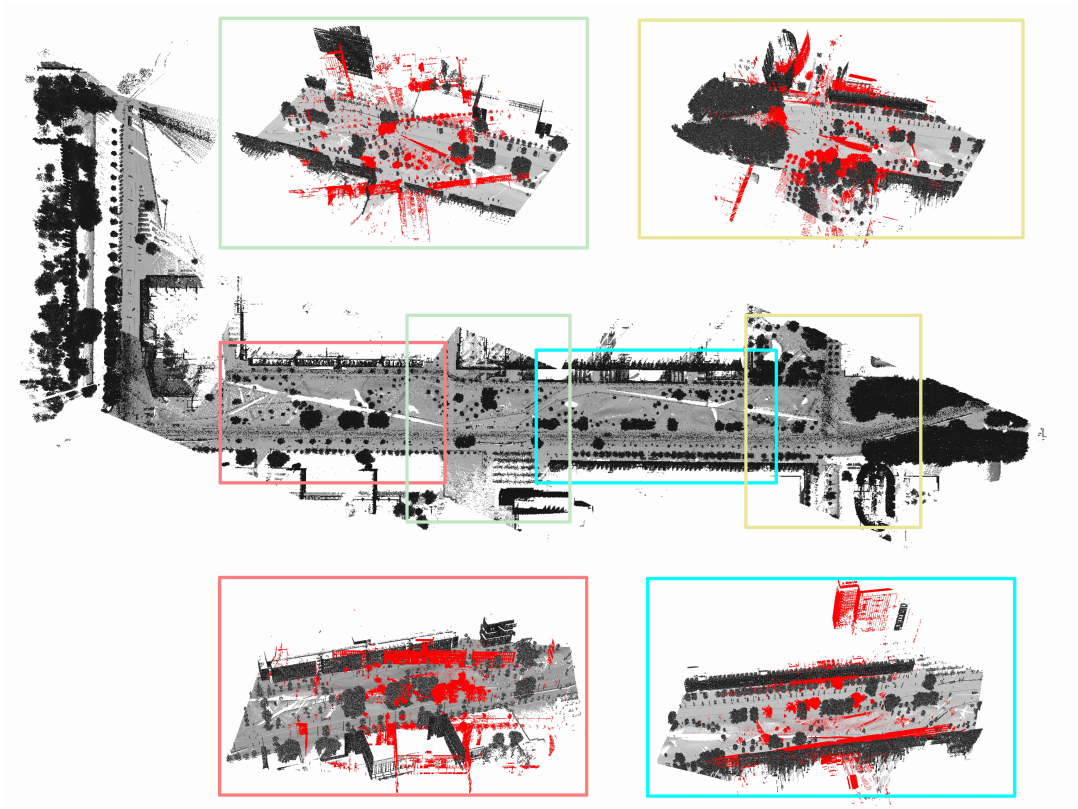


Figure A.2: Place recognition results of (TLS_2, MLS_2) (red box), (TLS_3, MLS_3) (green box), (TLS_4, MLS_4) (cyan box), (TLS_5, MLS_5) (yellow box). Transformed **TLS** point clouds are shown in red in the **MLS** reference map (grey).

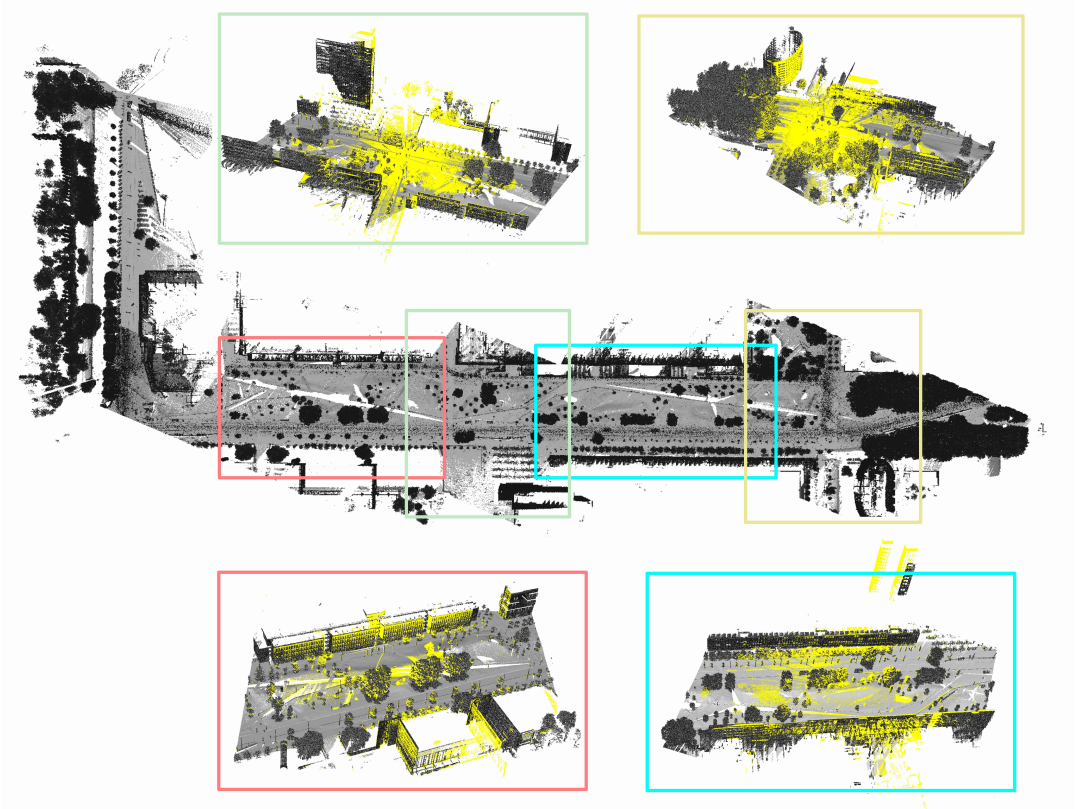


Figure A.3: Pose refinement results of (TLS_2, MLS_2) (red box), (TLS_3, MLS_3) (green box), (TLS_4, MLS_4) (cyan box), (TLS_5, MLS_5) (yellow box). Transformed **TLS** point clouds are shown in yellow in the **MLS** reference map (grey).

Bibliography

[1] Leica scan station p40/p30 system field manual. URL https://surveyequipment.com/assets/index/download/id/735/.

[2] Pixar points. URL https://web.archive.org/web/20131202234149/http://www.cgsociety.org/index.php/CGSFeatures/CGSFeatureSpecial/pixar_points.

[3] Mikaela Angelina Uy and Gim Hee Lee. Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4470–4479, 2018.

[4] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. Pointnetlk: Robust & efficient point cloud registration using pointnet. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7163–7172, 2019.

[5] K Somani Arun, Thomas S Huang, and Steven D Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on pattern analysis and machine intelligence*, (5):698–700, 1987.

[6] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117, 2006.

[7] David Barber, Jon Mills, and Sarah Smith-Voysey. Geometric validation of a ground-based mobile laser scanning system. *ISPRS journal of photogrammetry and remote sensing*, 63(1):128–141, 2008.

[8] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE transactions on pattern analysis and machine intelligence*, 24(4):509–522, 2002.

[9] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. 3dmfv: Three-dimensional point cloud classification in real-time using convolutional neural networks. *IEEE Robotics and Automation Letters*, 3 (4):3145–3152, 2018.

[10] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. International Society for Optics and Photonics, 1992.

[11] Marcus A Brubaker, Andreas Geiger, and Raquel Urtasun. Map-based probabilistic visual self-localization. *IEEE transactions on pattern analysis and machine intelligence*, 38(4):652–665, 2015.

[12] Wolfram Burgard, Armin B Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun. The interactive museum tour-guide robot. In *Aaai/iaai*, pages 11–18, 1998.

[13] Krishna Chintalapudi, Anand Padmanabha Iyer, and Venkata N Padmanabhan. Indoor localization without the pain. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pages 173–184, 2010.

[14] Michael Connor and Piyush Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE transactions on visualization and computer graphics*, 16(4):599–608, 2010.

[15] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[16] Haowen Deng, Tolga Birdal, and Slobodan Ilic. Ppfnet: Global context aware local features for robust 3d point matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 195–205, 2018.

[17] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 998–1005. Ieee, 2010.

[18] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[19] Gil Elbaz, Tamar Avraham, and Anath Fischer. 3d point cloud registration for localization using a deep neural network auto-encoder. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4631–4640, 2017.

[20] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[21] Klaus Finkenzeller. *RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*. John wiley & sons, 2010.

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

[23] Christofer Fröhlich, Markus Mettenleiter, et al. Terrestrial laser scanning—new perspectives in 3d surveying. *International archives of photogrammetry, remote sensing and spatial information sciences*, 36 (Part 8):W2, 2004.

[24] Andrea Frome, Daniel Huber, Ravi Kolluri, Thomas Bülow, and Jitendra Malik. Recognizing objects in range data using regional point descriptors. In *European conference on computer vision*, pages 224–237. Springer, 2004.

[25] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868, 2008.

[26] Haiyan Guan, Jonathan Li, Yongtao Yu, Cheng Wang, Michael Chapman, and Bisheng Yang. Using mobile laser scanning data for automated extraction of road markings. *ISPRS Journal of Photogrammetry and Remote Sensing*, 87:93–107, 2014.

[27] Timo Hackel, Nikolay Savinov, Lubor Ladicky, Jan D Wegner, Konrad Schindler, and Marc Pollefeys. Semantic3d. net: A new large-scale point cloud classification benchmark. *arXiv preprint arXiv:1704.03847*, 2017.

[28] Fei Han, Xue Yang, Yiming Deng, Mark Rentschler, Dejun Yang, and Hao Zhang. Sral: Shared representative appearance learning for long-term visual place recognition. *IEEE Robotics and Automation Letters*, 2(2):1172–1179, 2017.

[29] Xian-Feng Hana, Jesse S Jin, Juan Xie, Ming-Jie Wang, and Wei Jiang. A comprehensive review of 3d point cloud descriptors. *arXiv preprint arXiv:1802.02297*, 2018.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[31] J Hyyppä, H Hyyppä, D Leckie, F Gougeon, X Yu, and M Maltamo. Review of methods of small-footprint airborne laser scanning for extracting forest inventory data in boreal forests. *International Journal of Remote Sensing*, 29(5):1339–1366, 2008.

[32] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[33] Giseop Kim and Ayoung Kim. Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4802–4809. IEEE, 2018.

[34] Pileun Kim, Jingdao Chen, and Yong K Cho. Slam-driven robotic mapping and registration of 3d point clouds. *Automation in Construction*, 89:38–48, 2018.

[35] Antero Kukko, Harri Kaartinen, Juha Hyyppä, and Yuwei Chen. Multiplatform mobile laser scanning: Usability and performance. *Sensors*, 12(9):11712–11733, 2012.

[36] Akiyoshi Kurobe, Yusuke Sekikawa, Kohta Ishikawa, and Hideo Saito. Corsnet: 3d point cloud registration by deep neural network. *IEEE Robotics and Automation Letters*, 5(3):3960–3966, 2020.

[37] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. A sparse texture representation using local affine regions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1265–1278, 2005.

[38] Jesse Levinson, Michael Montemerlo, and Sebastian Thrun. Map-based precision vehicle localization in urban environments. In *Robotics: science and systems*, volume 4, page 1. Citeseer, 2007.

[39] Yong Li, Jinling Wang, Chris Rizos, Peter Mumford, and Weidong Ding. Low-cost tightly coupled gps/ins integration based on a nonlinear kalman filtering design. In *Proceedings of ION National Technical Meeting*, pages 18–20, 2006.

[40] Fuxun Liang, Bisheng Yang, Zhen Dong, Ronggang Huang, Yufu Zang, and Yue Pan. A novel skyline context descriptor for rapid localization of terrestrial laser scans to airborne laser scanning point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 165:120–132, 2020.

[41] Jiarong Lin and Fu Zhang. Loam livox: A fast, robust, high-precision lidar odometry and mapping package for lidars of small fov. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3126–3131. IEEE, 2020.

[42] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 82–92, 2019.

[43] JW Lovse, WF Teskey, G Lachapelle, and ME Cannon. Dynamic deformation monitoring of tall structure using gps technology. *Journal of surveying engineering*, 121(1):35–40, 1995.

[44] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[45] Debiao Lu and Eckehard Schnieder. Performance evaluation of gnss for train localization. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):1054–1059, 2014.

[46] Weixin Lu, Guowei Wan, Yao Zhou, Xiangyu Fu, Pengfei Yuan, and Shiyu Song. Deepvcp: An end-to-end deep neural network for point cloud registration. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 12–21, 2019.

[47] Weixin Lu, Yao Zhou, Guowei Wan, Shenhua Hou, and Shiyu Song. L3-net: Towards learning based lidar localization for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6389–6398, 2019.

[48] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[49] Gerald F Marshall and Glenn E Stutz. *Handbook of optical and laser scanning*. CRC Press, 2011.

[50] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[51] Michael J Milford and Gordon F Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *2012 IEEE International Conference on Robotics and Automation*, pages 1643–1649. IEEE, 2012.

[52] Baback Moghaddam and Alex Pentland. Probabilistic visual learning for object representation. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):696–710, 1997.

[53] Idrees Mohammed Oludare and Biswajeet Pradhan. A decade of modern cave surveying with terrestrial laser scanning: A review of sensors, method and application development. *International Journal of Speleology*, 45(1):8, 2016.

[54] Andriy Myronenko and Xubo Song. Point set registration: Coherent point drift. *IEEE transactions on pattern analysis and machine intelligence*, 32(12):2262–2275, 2010.

[55] Uzair Nadeem, Mohammad AAK Jalwana, Mohammed Bennamoun, Roberto Togneri, and Ferdous Sohel. Direct image to point cloud descriptors matching for 6-dof camera localization in dense 3d point clouds. In *International Conference on Neural Information Processing*, pages 222–234. Springer, 2019.

[56] Balázs Nagy and Csaba Benedek. Real-time point cloud alignment for vehicle localization in a high resolution 3d map. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[57] G Dias Pais, Srikumar Ramalingam, Venu Madhav Govindu, Jacinto C Nascimento, Rama Chellappa, and Pedro Miraldo. 3dregnet: A deep neural network for 3d point registration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7193–7203, 2020.

[58] Guan Pang and Ulrich Neumann. 3d point cloud object detection with multi-view convolutional neural network. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 585–590. IEEE, 2016.

[59] Shi Pu, Martin Rutzinger, George Vosselman, and Sander Oude Elberink. Recognizing basic structures from mobile laser scanning data for road inventory studies. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(6):S28–S39, 2011.

[60] I Puente, H González-Jorge, J Martínez-Sánchez, and P Arias. Review of mobile mapping and surveying technologies. *Measurement*, 46(7):2127–2145, 2013.

[61] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[62] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[63] Anshul Rai, Krishna Kant Chintalapudi, Venkata N Padmanabhan, and Rijurekha Sen. Zee: Zero-effort crowdsourcing for indoor localization. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 293–304, 2012.

[64] Gerhard Reitmayr and Dieter Schmalstieg. Location based applications for mobile augmented reality. In *Proceedings of the Fourth Australasian user interface conference on User interfaces 2003-Volume 18*, pages 65–73, 2003.

[65] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation*, pages 1–4. IEEE, 2011.

[66] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE, 2009.

[67] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2155–2162. IEEE, 2010.

[68] Samuele Salti, Federico Tombari, and Luigi Di Stefano. Shot: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125:251–264, 2014.

[69] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. From coarse to fine: Robust hierarchical localization at large scale. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12716–12725, 2019.

[70] Vinit Sarode, Xueqian Li, Hunter Goforth, Yasuhiro Aoki, Rangaprasad Arun Srivatsan, Simon Lucey, and Howie Choset. Pcrnet: point cloud registration network using pointnet encoding. *arXiv preprint arXiv:1908.07906*, 2019.

[71] Paul Scovanner, Saad Ali, and Mubarak Shah. A 3-dimensional sift descriptor and its application to action recognition. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 357–360, 2007.

[72] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics: science and systems*, volume 2, page 435. Seattle, WA, 2009.

[73] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765. IEEE, 2018.

[74] Freysteinn Sigmundsson, Andrew Hooper, Sigrún Hreinsdóttir, Kristín S Vogfjörd, Benedikt G Ófeigsson, Elías Rafn Heimisson, Stéphanie Dumont, Michelle Parks, Karsten Spaans, Gunnar B Gudmundsson, et al. Segmented lateral dyke growth in a rifting event at bárðarbunga volcanic system, iceland. *Nature*, 517(7533):191–195, 2015.

[75] Bastian Steder, Michael Ruhnke, Slawomir Grzonka, and Wolfram Burgard. Place recognition in 3d scans using a combination of bag of words and point feature based relative pose estimation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1249–1255. IEEE, 2011.

[76] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. Point feature extraction on 3d range scans taking into account object boundaries. In *2011 IEEE International Conference on Robotics and Automation*, pages 2601–2608. IEEE, 2011.

[77] Stephen V Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1):77–89, 1997.

[78] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[79] Shamik Sural, Gang Qian, and Sakti Pramanik. Segmentation and histogram generation using the hsv color space for image retrieval. In *Proceedings. International Conference on Image Processing*, volume 2, pages II–II. IEEE, 2002.

[80] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[81] M Tsakiri, D Lichti, and N Pfeifer. Terrestrial laser scanning for deformation monitoring. 2006.

[82] Jinhu Wang. Scalable information extraction from point cloud data obtained by mobile laser scanner. 2017.

[83] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[84] Zhen Wang, Liqiang Zhang, Tian Fang, P Takis Mathiopoulos, Xiaohua Tong, Huamin Qu, Zhiqiang Xiao, Fang Li, and Dong Chen. A multiscale and hierarchical feature extraction method for terrestrial laser scanning point cloud classification. *IEEE Transactions on Geoscience and Remote Sensing*, 53(5):2409–2425, 2014.

[85] Aloysius Wehr and Uwe Lohr. Airborne laser scanning—an introduction and overview. *ISPRS Journal of photogrammetry and remote sensing*, 54(2-3):68–82, 1999.

[86] Chenglu Wen, Xiaotian Sun, Jonathan Li, Cheng Wang, Yan Guo, and Ayman Habib. A deep learning framework for road marking extraction, classification and completion from mobile laser scanning point clouds. *ISPRS journal of photogrammetry and remote sensing*, 147:178–192, 2019.

[87] Yuquan Xu, Vijay John, Seiichi Mita, Hossein Tehrani, Kazuhisa Ishimaru, and Sakiko Nishino. 3d point cloud map based vehicle localization using stereo camera. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 487–492. IEEE, 2017.

[88] Bisheng Yang, Zhen Dong, Fuxun Liang, and Yuan Liu. Automatic registration of large-scale urban scene point clouds based on semantic feature points. *ISPRS Journal of Photogrammetry and Remote Sensing*, 113:43–58, 2016.

[89] Haoyang Ye, Yuying Chen, and Ming Liu. Tightly coupled 3d lidar inertial odometry and mapping. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3144–3150. IEEE, 2019.

[90] Huan Yin, Li Tang, Xiaqing Ding, Yue Wang, and Rong Xiong. Locnet: Global localization in 3d point clouds for mobile vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 728–733. IEEE, 2018.

[91] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1802–1811, 2017.

[92] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, 2014.

[93] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41(2):401–416, 2017.

[94] Zhenxin Zhang, Liqiang Zhang, Xiaohua Tong, P Takis Mathiopoulos, Bo Guo, Xianfeng Huang, Zhen Wang, and Yuebin Wang. A multilevel point-cluster-based discriminative feature for als point cloud classification. *IEEE Transactions on Geoscience and Remote Sensing*, 54(6):3309–3321, 2016.

[95] Lei Zhou, Siyu Zhu, Zixin Luo, Tianwei Shen, Runze Zhang, Mingmin Zhen, Tian Fang, and Long Quan. Learning and matching multi-view descriptors for registration of point clouds. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 505–522, 2018.

[96] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[97] Song Chun Zhu and Alan Yuille. Region competition: Unifying snakes, region growing, and bayes/mdl for multiband image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 18 (9):884–900, 1996.