UCLouvain

École polytechnique de Louvain

3D change detection to monitor construction sites

Authors: **Dinh Thanh Phong Do, Nicolas Jadoul**
Supervisor: **Pierre Schaus**
Readers: **Achille Morenville, Timothée Lonfils**
Academic year 2022–2023
Master [120] in Mathematical / Data Science Engineering

Abstract

In the construction industry, there is growing interest in adopting new building technologies, including laser scanning technologies such as Light Detection And Ranging (LiDAR).

Additionally, advancements in Simultaneous Localization and Mapping (SLAM) algorithms enable robots, such as Boston Dynamics's SPOT, to autonomously navigate construction sites and generate point clouds. These point clouds serve as valuable data for monitoring purposes.

The aim of this thesis is to develop a method for monitoring significant differences between two point clouds acquired at different stages of the construction process. Furthermore, the proposed method can be extended to compare point clouds with Building Information Modeling (BIM) data, facilitating comprehensive analysis and evaluation of the construction progress.

By addressing the challenges of comparing and analyzing point clouds in the context of construction site monitoring, this research contributes to the advancement of construction industry practices and the adoption of innovative technologies.

Acknowledgements

We would like to express our sincere gratitude to the following individuals and organizations who have contributed to the successful completion of this thesis:

- Pierre Schaus, our promotor, for his valuable guidance, supervision, and continuous support throughout the research process. His insights, enthusiasm, constructive advice, and feedback have been decisive in shaping this work.
- Achille Morenville for his technical guidance and his follow-up throughout the whole year. He has been available at every stage of this master's thesis. We are very thankful for his advice and willingness to share his insights and knowledge.
- Timothée Lonfils from Buildwise, for his trust and willingness to help us and valuable insights into the market and industry practices. His expertise and input have provided valuable context and relevance to the research.
- Buildwise, for providing access to the robot SPOT. Their support and collaboration have allowed us to collect data crucial to this study.

Also, we would like to extend our gratitude to all the professors of UCLouvain for their dedication and their contributions to our education. Their passion for knowledge and commitment to excellence have played an essential role in shaping our understanding of the subject matter.

Additionally, we would like to acknowledge our friends and family for their support, understanding, and encouragement throughout this journey.

Lastly, we would like to express our appreciation to everyone else who has supported us directly or indirectly during this thesis. Your support, whether big or small, has made a significant difference in the completion of this work.

Contents

Chapter 1

Context and Objectives

1.1 Aim and Outline of the master thesis

The Boston Dynamic's Spot robot equipped with a LiDAR sensor provides perspectives of autonomous navigation through the utilization of a pre-implemented service known as GraphNav, or, by implementing a SLAM algorithm as previously explored in Achille's master thesis [1].
Autonomous navigation enables novel applications, including tracking construction sites using the robot, by recording and mapping a point cloud of its surrounding environment.

A significant application of the recorded point cloud is its comparison with other point clouds and detect their differences. Those other point clouds can originate from previous records, or, from a theoretical model known as a building information model (BIM). The objective is to identify changes or faults that may occur during construction.

This master thesis investigates 2 different approaches to detecting those changes, specifically, focusing on translation and rotation changes. The first approach employs different computer vision techniques which allow the detection, classification,

and segmentation of these changes. The second one is based on a deep learning approach, drawing inspiration from recent progress in point cloud classification and semantic segmentation.

More precisely, to enable the evaluation and comparison of both approaches, a dataset representing walls and buildings was generated. The performance of each approach is assessed using different metrics for classification and semantic segmentation tasks. And conclusions about the deep learning approach are drawn, in comparison to the classical method, showing some perspective for future application.

Chapter 1 provides an overview of the context and technologies employed, including the Spot robot provided by Buildwise and the LiDAR sensor utilized for capturing point cloud data.

Afterward, the first approach referred as the classical approach, is described from Chapter 2 to Chapter 4. Each chapter details a step within the deterministic method, including point cloud registration, plane detection, plane matching, and change detection.

Chapter 5 introduces the second approach. Relevant definitions and models are described alongside the presentation of our proposed network architecture.

Finally, Chapter 6 delves into the implementation details and provides a comprehensive analysis of the results obtained from the evaluation of these two approaches across various metrics.

1.2 Point Cloud

In a wide range of applications like robotics, architecture, virtual reality, and many more, it is important to detail information about the surrounding environment.

Specifically, in the context of change detection and monitoring construction sites, there is a need to accurately depict buildings and objects using a 3D model.

To achieve this, point clouds are commonly employed. Point clouds represent a set of unordered discrete data points, where each point lies on a frame described by its cartesian coordinates. Typically, these points mainly represent surfaces providing a description of objects and environment.

Point clouds can be generated through various techniques including Light Detection and Ranging (LiDAR) sensors which use laser beams to capture spatial information, RGB-D cameras which exploit the depth to reconstruct the surrounding, or structure-from-motion (SFM) which leverage multiple images taken from different angles [2].

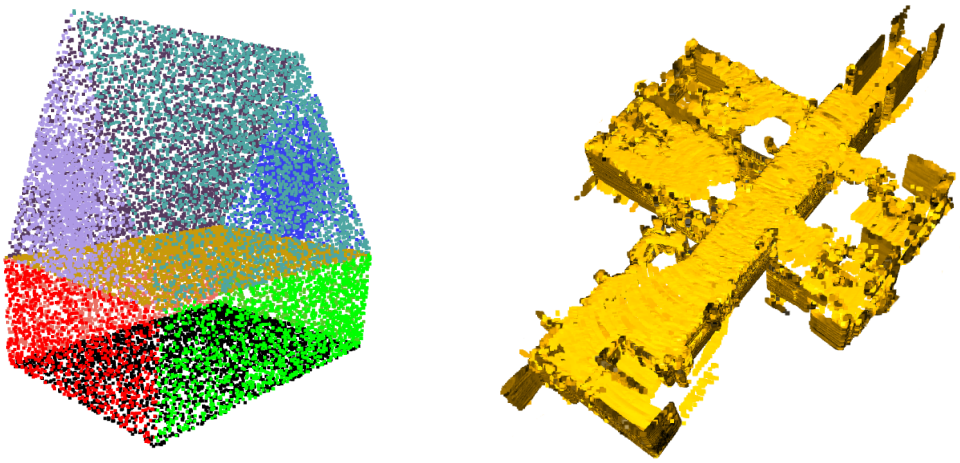


Figure 1.1: Two examples of point clouds: A house with colored surfaces for visualization and a Buildwise's floor containing a hallway and four connected rooms.

1.3 SPOT

Spot is an agile mobile robot capable to navigate in various environments such as construction sites and rough terrains. Moreover, with the help of the Spot SDK, developers and operators can make Spot do repetitive tasks and automate routine inspections [3].

The robot itself possesses 12 degrees of freedom, with 3 degrees per leg. It has the capability to carry a payload weighing up to 14 kg. The estimated autonomy is around 90 minutes for run time and 180 minutes for standby time. Furthermore, it is designed to operate within a temperature range spanning from $-20°C$ to $45°C$ [4].

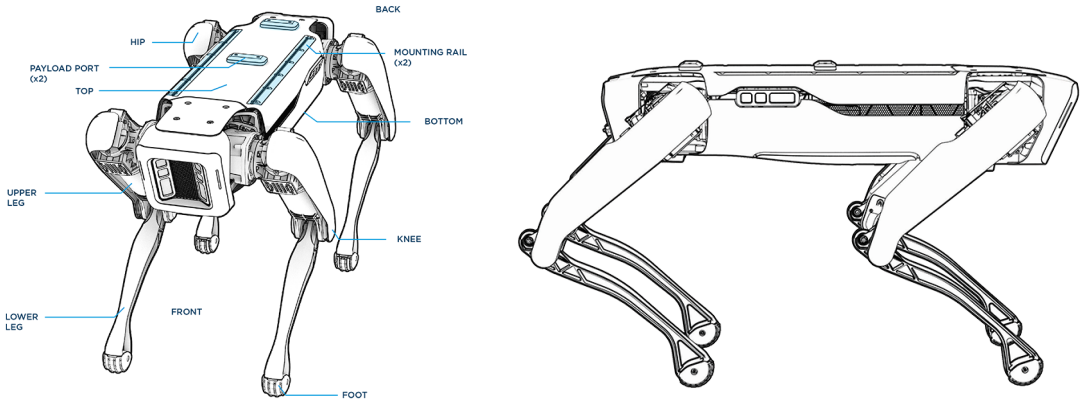


Figure 1.2: Front and side views of Spot with its main physical features [4].

1.3.1 Geometry and Frames

Spot use frames in order to help him navigate in the world. While moving, Spot keeps track of frames such as its body frame, its sensor frames, the inertial frame, objects frames (detected object in its surrounding) and many more [5].

In order to keep a coherent world, Spot applies spatial transformations to monitor relationships between those different frames. The transformation between two

frames is defined by a translation vector $[x, y, z]$ which indicates the difference between both origin and a rotation quaternion $[w, x, y, z]$ which indicates the difference between the axis orientations [6].

Those rigid transformations between frames can also be expressed with a single transformation matrix with homogeneous coordinates. The shape of the transformation matrix is given as follows:

$$T = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \tag{1.1}$$

where $\mathbf{R} \in \mathbb{R}^{3\times3}$ is the rotation matrix, $\mathbf{t} \in \mathbb{R}^3$ is the translation matrix while 1 is a scalar.

More explicitly, if $T_{A\to B}$ is the transformation matrix between frame A and frame B, for a given point in frame A (x_A, y_A, z_A), this point's coordinate in frame B can be found using $T_{A\to B}$.

$$\begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix} = \underbrace{\begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}}_{:= T_{A\to B}} \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix} \tag{1.2}$$

1.3.2 Autonomy Services

Spot robot features a mapping, localization, and autonomous navigation system, known as GraphNav, which consists of on the one hand a service used on the robot and on the other hand an API for the development of autonomous navigation behavior [7].

In order to make Spot do an automated routine, a map recording must be done first. This map, representing the world and generated by GraphNav, consists of edges and waypoints:

- A waypoint represents an area in the world that is accessible for the robot by navigation. During a map recording, it is created approximately every two

meters. It consists of a reference frame, a name, a unique ID, annotations, and sensor data. A waypoint contains a snapshot that bundles those data in one unit.

- An edge in a map represents how the robot moves and the relationship in 3D space between the two waypoints. It involves relative poses between two waypoints and also information about how the robot should move along that edge.

Once the new map is recorded, if an operator wants Spot to make repetitive tasks, the robot must have its localization initialized. As illustrated in Figure 1.3, a way to initialize the robot is to set the localization near a specific fiducial (such as an AprilTag) that is previously recorded.

Once the robot's initial localization has been established, it will consistently monitor its position in relation to waypoints present on the recorded map. As the robot moves in the map, the waypoint used for localization transitions to a neighboring waypoint.

In the Spot tablet app, a feature called Autowalk is available. This feature is built upon GraphNav and allows the robot operator to record a GraphNav map and do repetitive missions.

1.4 Simultaneous localization and mapping

In the case of monitoring construction sites, drastic changes can happen between 2 map records: New walls can be built, and more obstacles appear as the building is constructed.

Because of the dynamic nature of the construction site, Spot autonomy services may not be sufficient to offer real autonomy and the robot need to be able to simultaneously locate itself and map the world as the robot is moving. This common

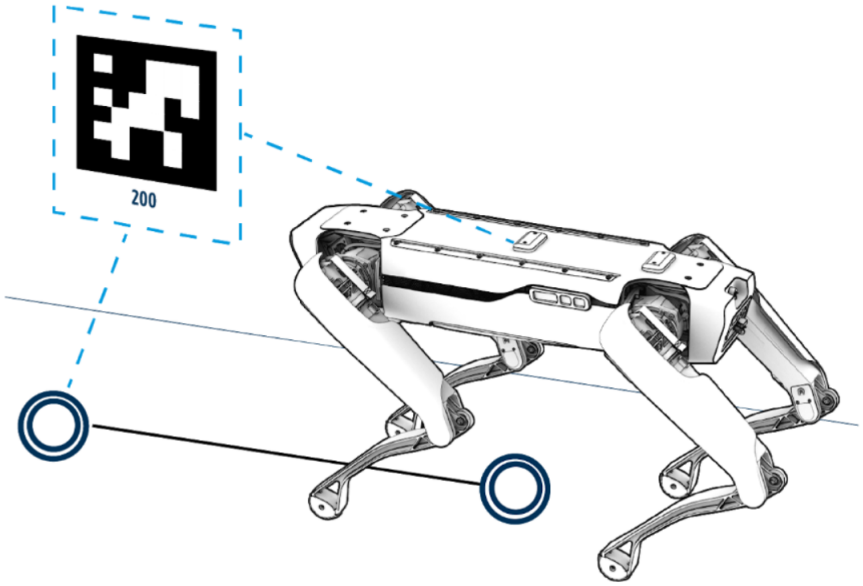


Figure 1.3: Illustration of waypoints, edges and Spot initialization with an AprilTag [7].

problem is also known as Simultaneous Localization and Mapping (SLAM) problem.

Common algorithm to solve the SLAM problem belongs either to the filtering approaches or the smoothing approaches [1].

- Filtering approaches use an on-line description of the SLAM problem. Popular methods such as the Kalman filter, Extended Kalman filter, and Particle filters belong to this category.
  More explicitly, the on-line description means that those methods estimate for every discrete time t, the **robot's pose** x_t and the map m based on its odometry history $u_{1:t}$ and measurement history $z_{1:t}$ derived from LiDAR.
- Smoothing approaches use a full description of the SLAM problem. Graph-based optimization such as GraphSLAM belongs to this category.
  More explicitly, the full description means that those method estimate for every discrete time t, the **full robot's trajectory** $x_{1:t}$ and the map m based

on its odometry history $u_{1:t}$, measurement history $z_{1:t}$ (derived from LiDAR, cameras,...).

1.5 Light Detection And Ranging

As previously mentioned, point clouds can be obtained using Light Detection And Ranging (LiDAR) which use laser beams to measure the distance between the source sensor and a targeted object.

In order to measure the distance, the LiDAR sensor emits a laser beam in the direction of a target object. This laser beam travels through the air, at a speed similar to the speed of light c, and is reflected back to the sensor once it reaches the target. Then, the sensor measures the time t taken for the laser pulse to return and the distance is computed using the equation $d = \frac{ct}{2}$.

In the specific case of robotic and construction sites, LiDAR sensors can be used for several reasons. Firstly, LiDAR offers good accuracy by exploiting wavelength in the range of 514-1064 nm providing greater spatial resolution than radar and achieving centimeter-level accuracy. Secondly, LiDAR sensors can rapidly capture and process data, delivering real-time information about a robot's surroundings. This is beneficial for tasks such as localization, obstacle detection, path planning and SLAM [1][8].

Within the scope of our work, the Velodyne VLP-16 sensor is utilized as the primary sensor. VLP-16 sensor possesses 16 infra-red lasers paired with detectors to measure distances to objects where the vertical field of view obtained is 30° with a vertical resolution of 2°. Each laser operates at a firing frequency of 18 kHz facilitating the acquisition of a set of 3D point data in real-time [11].

In order to capture a point cloud representing a building, it is necessary to gather

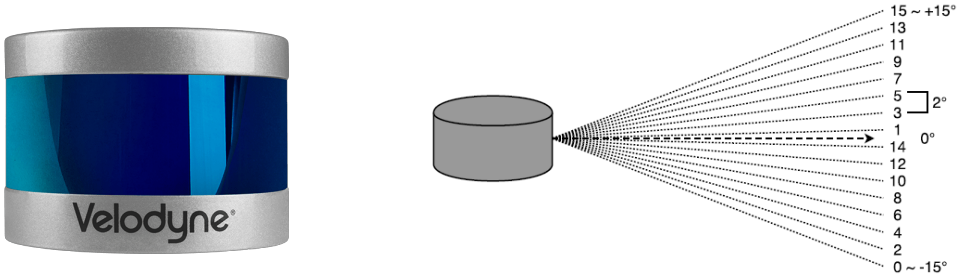


Figure 1.4: Illustration of Velodyne VLP-16 LiDAR and its vertical range [9][10].

distance information in all horizontal directions. With the help of rotating fired lasers with a speed ranging from 5 Hz to 20 Hz, VLP-16 achieves a horizontal field of view of 360° [11].

The obtained point clouds demonstrate a level of accuracy with a measurement noise up to ±3 cm [8].

Chapter 2

Point Clouds Registration

Once our point cloud is acquired through the robot's autonomy services such as GraphNav or using a SLAM algorithm, the one collected is not necessarily aligned with the reference point cloud obtained from the BIM file or past missions.

Indeed, those point clouds can be displayed in different reference frames depending on the exact initial localization and direction of the robot or on the coordinate system used in the BIM file. Thus, alignment of point clouds is important, as otherwise, we wouldn't be comparing the exact same space location.

This chapter explores widespread solutions to register point clouds. It begins by giving a formal definition of the point clouds registration problem followed by an introduction to 2 algorithms to perform the denoted task:

- A random sample Consensus (RANSAC) algorithm which quickly identifies a good estimate.
- An Iterative Closest Point (ICP) algorithm which refines the result obtained by the initial estimate.

2.1 Definition of the registration problem

Given a source point cloud $X := \{x_1, x_2, ...x_n\}$ and a target point cloud $Y = \{y_1, y_2, ..., y_m\}$, the problem of point clouds registration, also known as the transformation estimation problem, is formally equivalent to determine the rigid transformation matrix T^* between 2 point clouds X and Y such that [12]:

$$T^* = \underset{T}{\operatorname{argmin}}\ \operatorname{dist}(T(X), Y) \tag{2.1}$$

More explicitly, the distance between two point clouds denoted as $\operatorname{dist}(T(X), Y)$, is based on the sum of the distance between correspondent points in X and Y. These correspondences are defined by a correspondence set $(i, j) \in K$.

The distance metric used is the Euclidean distance since the robot's frames are described with a 3D orthonormal basis. Therefore, given two corresponding points, $x_i \in X$, $y_j \in Y$ and a transformation matrix T, the distance is computed as follow:

$$\operatorname{dist}(T(x_i), y_j) = \|y_j - T(x_i)\|_2^2 \tag{2.2}$$

Moreover, based on the definition of the transformation between two frames, as defined in Equation 1.2, $T(x_i) = Rx_i + t$. We can give an explicit definition of the problem:

$$R^*, t^* = \underset{R,t}{\operatorname{argmin}} \sum_{(i,j)\in K} \|y_j - Rx_i - t\|_2^2 \tag{2.3}$$

The registration problem can also be viewed as a chicken and egg problem: In order to find the transformation matrix, we need to have the correspondence set. And to have the correspondence set, we need the transformation matrix.

In the following section, we will cover a technique that breaks this circular problem by initially estimating the transformation matrix.

2.2 Random Sample Consensus (RANSAC)

The first technique employed is called RANdom SAmple Consensus (RANSAC) [13].
RANSAC is an iterative method where in each iteration, the method derives a hypothetical transformation $\hat{T}$ using 3 pairs of points: $x_1, x_2, x_3 \in X$ from the source point cloud and $y_1, y_2, y_3 \in Y$ from the target one.

Then, the method evaluates the consistency of the registration with respect to the derived transformation $\hat{T}$. This evaluation is performed using a criterion, such as the distance to the nearest neighbor in the opposite point cloud, allowing the identification of inliers and outliers.

The method finally returns the transformation T^* with the largest number of inliers, indicating the best registration.

Algorithm 1 RANSAC (Global Registration)

Step 1: Let X be the source point cloud and Y be the target point cloud.

Step 2: Select a random subset of 3 points $X' \subset X$ and 3 points $Y' \subset Y$

Step 3: Compute the candidate transformation matrix $\hat{T}$ between X' and Y' and apply it to X.

Step 4: For all $x \in X$, determine if it is an inlier or an outlier based on a criterion.

Step 5: Repeat **Step2 - Step 4** a pre-determined number of iteration. Return the transformation T^* with the largest amount of inliers.

This method offers a significant advantage for point cloud registration since it proves itself to be robust and reliable against outliers such as measurement noise commonly encountered in LiDAR-based method.
Furthermore, this method is characterized as a global method since it allows estimation of transformation without requiring any prior assumption about the proximity of the source and target frames. This attribute is particularly valuable as it allows for flexibility in registering point clouds that may have significant

geometric differences or misalignments.

2.3 RANSAC: Improvements

However, RANSAC suffers from certain limitations which can be addressed. One drawback is that the method treats all correspondences equally and computes the transformation even for false matches.
To tackle these limitations, different approaches can be employed as outlined bellow:

- In **step 2** of algorithm 1, a variation can be made. When choosing random points $X' \in X$, their corresponding points $Y' \in Y$ are selected based on a nearest neighbor search in 33-dimensional feature space [14].
- A pruning step can be implemented to reject quickly false matches. Various criteria, such as the proximity of correspondent points X' and Y' or the alignment of their normals, are utilized to identify and discard inaccurate correspondences[15][16].

2.4 Iterative Closest Point (ICP)

As discussed earlier, the registration problem presents a chicken and egg dilemma since the search for a transformation matrix prerequisite points correspondences and conversely. Fortunately, RANSAC provides a good initial guess of the transformation matrix offering a starting point to address this circular problem.

The Iterative Closest Point (ICP) takes advantage of this starting point in order to further refine the registration outcome. This algorithm operates in a loop of two steps: finding correspondences between the point clouds X and Y, followed by refining the transformation matrix T. By iteratively establishing correspondences and refining the transformation matrix, ICP ultimately achieves a more accurate registration result. The algorithm converges locally to a minimum of the distance

function dist$(T(X), Y)$ and stops when the function value falls below a predetermined threshold [17].

The pseudocode for the ICP algorithm is presented in Algorithm 2. Steps 2 and 3 of the algorithm involve finding correspondences and determining the transformation matrix, respectively.

Assuming an initial matrix $\hat{T}$ is available, finding the correspondent point $y_j \in Y$ based on $x_i \in X$ is simply a 1-nearest neighbor problem. Since a point cloud can contain up to millions of points, the implementation of **Step 2** involves the construction of a kd-tree structure to rapidly find an approximate nearest neighbor [18].

The minimization problem in **Step 3** is often solved in two steps. The first translates the source point cloud so that the center of masses of both point clouds are aligned. Then the second step computes the rotation that aligns them using singular value decomposition (SVD) [17].

Algorithm 2 Iterative Closest Point (ICP)

Step 1 Let $R \in \mathbb{R}^{3\times3}$ be a rotation matrix and $t \in \mathbb{R}^3$ be a translation vector. Let X be the source point cloud and Y be the target point cloud.

Step 2 For all $x_i \in X$, find the corresponding y_j such that:

$$y_j \leftarrow \underset{y\in Y}{\operatorname{argmin}} \|y - Rx_i - t\|_2^2$$

Step 3 Find R, tr such that:

$$R, t \leftarrow \underset{R,t}{\operatorname{argmin}} \sum_{x_i\in X} \|y_j - Rx_i - t\|_2^2$$

Step 4 If $\sum_{x_i\in X} \|y_j - Rx_i - t\|_2^2$ is lower than a threshold *trsh*, then stop. Otherwise, return to step 2.

2.5 ICP: Improvements

Point clouds captured in construction sites often contain walls and buildings consisting of planar surfaces. An idea to improve the ICP algorithm is to align those points with the underlying surface they represent, rather than establishing point correspondences [19] [20].
The point-to-plane ICP addresses this concern by disregarding error of between 2 corresponding points, $x_i \in X, y_j \in Y$, if they are aligned to the same plane. To achieve this, the algorithm computes approximate normals n_i for each point y_j and applies a scalar product between the normal and the vector $y_j - T(x_i)$.

Algorithm 3 ICP Point-to-plane

Step 1 Let $R \in \mathbb{R}^{3\times3}$ be a rotation matrix and $t \in \mathbb{R}^3$ be a translation vector. Let X be the source point cloud and Y be the target point cloud. n is the estimated normal for each point $y \in Y$

Step 2 For all $x_i \in X$, find the corresponding y_j such that:

$$y_j \leftarrow \underset{y\in Y}{\operatorname{argmin}} \|y - Rx_i - t\|_2^2$$

Step 3 Find R, t such that:

$$R, t \leftarrow \underset{R,t}{\operatorname{argmin}} \sum_{x_i\in X} \Big(n_i \cdot (y_j - Rx_i - t)\Big)^2$$

Step 4 If $\sum_{x_i\in X} \|y_j - Rx_i - t\|_2^2$ is lower than a threshold *trsh*, then stop. Otherwise, return to step 2.

Chapter 3

Planes Detection

Not all changes are relevant for analysis in the context of construction sites. The main objects to compare would be the core structure of a building while humans and construction tools can be disregarded as noise.

Since buildings contain many planar structures like walls and ceilings, one can take advantage of this simple geometry to extract flat surfaces from the point cloud data. This approach provides a primitive noise filtering and a planar segmentation which allows the ability to later compare planes together.

This chapter explores multiple popular methods that are relevant for plane detection in point clouds, namely Generalised Hough Transform and RANSAC. Those algorithms will be described and improvements made to these algorithms will be discussed to provide a comprehensive overview of planes detection techniques.

3.1 Generalised Hough Transform

The Generalised Hough Transform is a classical feature detection technique in computer vision and digital image processing that was initially used to identify lines, circles, and ellipses in 2D images [21]. But it has since been extended to a

wide range of applications such as plane detection in point cloud [22].

The method can be decomposed in three steps:

1. Given an input point cloud, the algorithm maps each data point of the original space to a discretized parameter space representation. This parameter space is judiciously selected to fit models such as plane equations.
2. Then, the algorithm systematically explores all possible plane equations within the parameter space. For each of those candidate plane equations, a voting process is conducted and the results are cast in an accumulator.
3. Based on the accumulator, the algorithm selects the most probable planes which are planes with a higher number of votes.

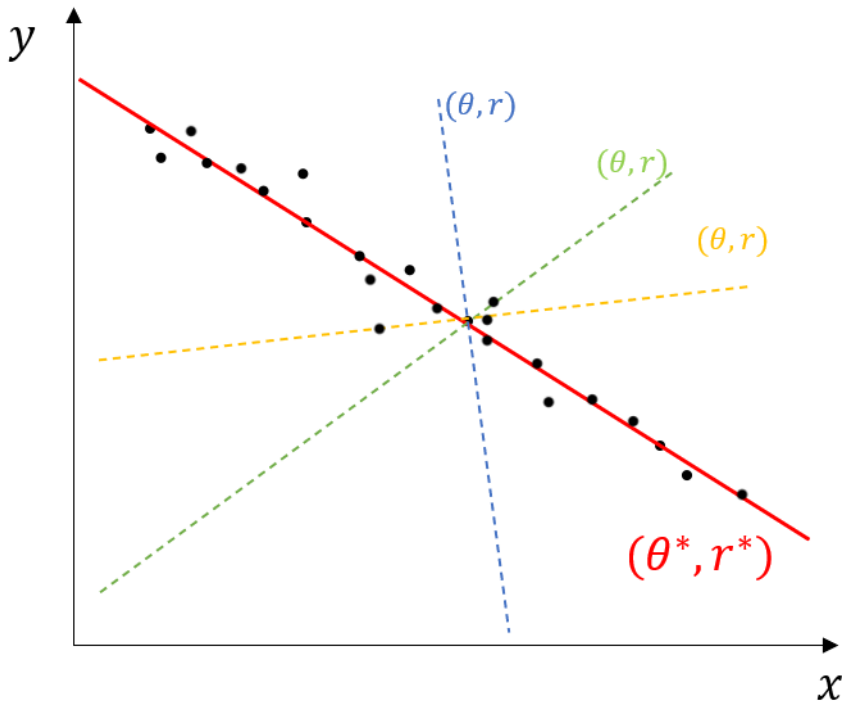


Figure 3.1: Illustration of the Generalised Hough Transform for line detection. Each one of those lines is described by parameters (θ, r). The best line (θ, r), illustrated in red, is the one with the most inliers.

More specifically, a plane in the original space is given by a normal vector $n =$

(n_x, n_y, n_z) and a distance to the origin r. All point $p = (p_x, p_y, p_z)$ which belong to the plane fit the following equation:

$$p_x n_x + p_y n_y + p_z n_z = r \tag{3.1}$$

Moreover, a plane can be described in a spherical coordinate (r, θ, ϕ) which will be the parameter space (Eq 3.2). Given this space, the algorithm will discretize the space and perform a voting process.

$$p_x \cos(\theta)\sin(\phi) + p_y \sin(\theta)\sin(\phi) + z\cos(\phi) = r \tag{3.2}$$

Algorithm 4 Generalised Hough Transform (GHT)

Step 1 Let (r_i, θ_j, ϕ_k) be cells of the discretized parameter space. Let x_m be a point in the given point cloud X, and A be the accumulator.

Step 2 For each cells (r_i, θ_j, ϕ_k), find the number of points x_m that lies on the plane, denoted as n_{ijk}, with the equation 3.2. And update the accumulator:

$$A[i, j, k] = n_{ijk}$$

Step 3 Search the cell with the local maximal score.

One of the major drawbacks of the GHT is its long computation time. Specifically, the second step of the Algorithm 4, which involves incrementing the cells of the accumulator, requires a computational complexity of $\mathcal{O}(|X| \cdot N_\theta \cdot N_\phi)$. Consequently, researchers have conducted studies to enhance its efficiency [22].

The first approach to address this issue is known as the Probabilistic Hough Transform (PHT). This method aims to decrease the number of points involved during this second step by selecting randomly a few of them reducing the computational complexity [23].

Another technique employed is the Adaptive Probabilistic Hough Transform (APHT) which adds a stopping criterion by monitoring the accumulator. Once a stable structure emerges, the algorithm terminates, thereby avoiding unnecessary computations and further reducing the overall processing time [24].

3.2 RANSAC for plane detection

RANSAC is also a popular stochastic method used for plane detection due to its robustness against outliers and its simplicity.

As discussed in the previous chapter on global registration, this method follows a similar approach by selecting random points in a point cloud to fit a model. Then it evaluates the number of inliers that fit a candidate model and returns a consensus based on the agreement among the majority of the data samples.

In the specific case of plane detection in a point cloud, the RANSAC algorithm (Algorithm 5) chooses 3 points x_1, x_2, x_3 and utilizes them to fit a plane. Afterward, it identifies inliers and outliers point from the initial point cloud and returns the plane that possesses the largest amount of inliers [25].

The number of iteration k performed by the algorithm can be expressed in function of the desired probability success. If we define w as the probability of choosing one inlier in the dataset as in Equation 3.3, the probability of choosing 3 inliers becomes w^3.

$$w = \frac{\text{number of inliers}}{\text{number of points}} \tag{3.3}$$

The probability that the algorithm never selects 3 inliers after k iteration, denoted as $(1 - p)$, becomes:

$$(1 - p) = (1 - w^3)^k \tag{3.4}$$

And thus, the number of iteration can be parametrized in the function of the confidence probability p as follow:

$$k = \frac{\log(1 - p)}{\log(1 - w^n)} \tag{3.5}$$

The parameters $trsh_d$ and $trsh_n$ in the criterion in **Step 5** can be chosen wisely depending on the accuracy and noise of the point cloud measurement and normal estimates.

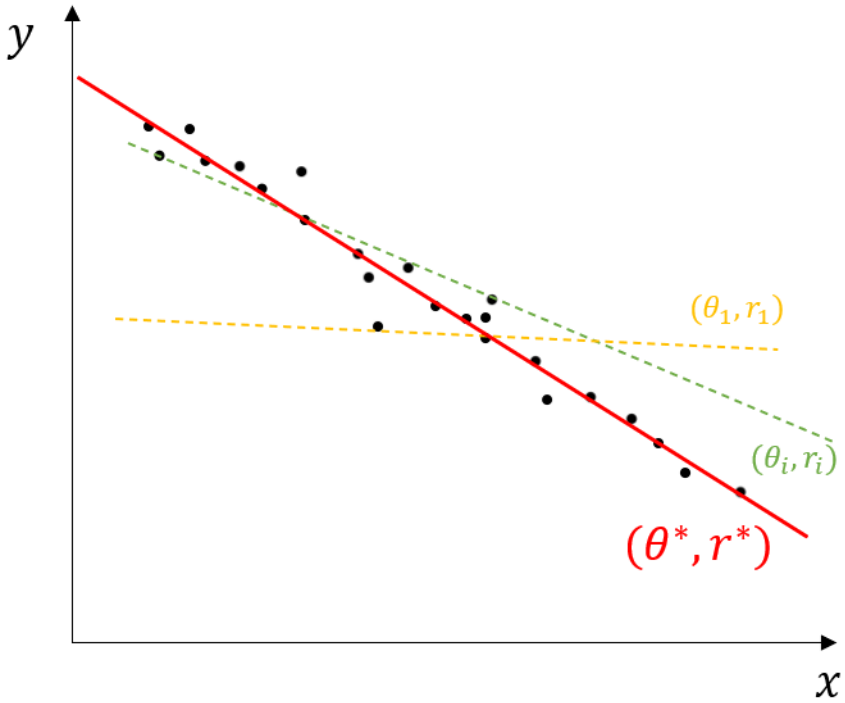


Figure 3.2: Illustration RANSAC algorithm for line detection. The algorithm selects 2 random points to define a line. The one with the best number of inliers, depicted in red, is considered the best fit.

The RANSAC algorithm (Algorithm 5), unlike the Generalised Hough Transform (Algorithm 4), extracts one plane at a time. Hence, the algorithm needs to be recursively used to extract all planes in a point cloud. Each time, we remove the points belonging to the plane at a given iteration.

Algorithm 5 RANSAC (Plane Detection)

Step 1: Let X be the source point cloud. We denote x as a point of X and n_x as the estimated normal for this point.

Step 2: Select a random subset of 3 points $x_1, x_2, x_3 \subset X$ which describe a candidate plane P

Step 3: Compute the vectors

$$\vec{u} = x_2 - x_1 \qquad \vec{v} = x_3 - x_1$$

Step 4: Compute the plane normal

$$\vec{n} = \frac{\vec{u} \times \vec{v}}{\|\vec{u} \times \vec{v}\|}$$

Step 5: For all $x \in X \backslash X'$, determine if it is an inlier or an outlier based on the following criterion:

$$\text{dist}(P, x) \leq trsh_d \qquad n_x \cdot \vec{n} \geq trsh_n$$

Step 6: Repeat **Step2 - Step5** a pre-determined number of time k. Return the candidate plane P with the largest amount of inliers.

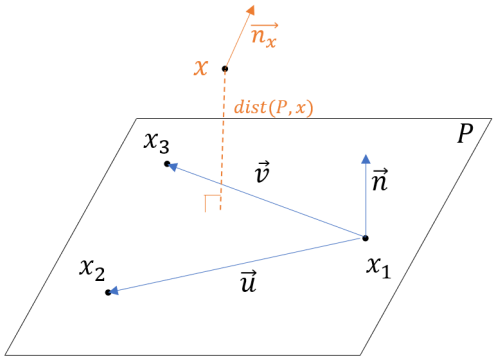


Figure 3.3: Illustration of the symbols described in Algorithm 5

Chapter 4

Planes Matching and Changes Detection

In the previous chapters, two essential tasks were addressed: point cloud registration and the detection of planar structures. Those steps play crucial roles in change detection.

To achieve our main objective of detecting changes in significant structures between two point clouds, two additional steps need to be undertaken. Firstly, after identifying planar structures in both point clouds, our aim is to establish correspondences between the detected planes. Subsequently, once the matching of corresponding planes is performed, the next step involves classifying the specific changes that have occurred between the two point clouds.

The purpose of this chapter is to address these two remaining steps: finding correspondences and finding transformations between 2 planes such as translation, rotation, or no change. For this purpose, a plane-matching algorithm and a transformation classification technique are proposed.

4.1 Plane Matching

In practice, in the scenario where there is a difference between theoretical and built planes such as walls, those differences are typically not too gross and are not always qualitatively visible. This observation allows us to make an assumption that aligns with reality while remaining not too restrictive. The assumption implies the following considerations:

- **Translation Transformation:** If a translation transform exists between two planes, the magnitude of this translation should not exceed a threshold of around a few dozen of centimeters. This involves the distance between their centroid remaining within the predetermined threshold.
- **Rotation Transformation:** If a rotation transformation exists between two planes, the magnitude of this rotation should not exceed a threshold around a dozen of degrees. Thus, their normal vectors should remain aligned within a predetermined limit.

Based on these assumptions, and inspired by a similar approach [26], we propose a point cloud plane matching algorithm (Algorithm 6). This algorithm pre-selects pairs of planes where their centroids are below a threshold and matches the pair where their normals are the most aligned.

4.2 Change Detection

Once the plane matching is completed, we can classify the nature of the transformation between two planar surfaces, specifically determining if there is a translation, rotation, or no transformation present. Since the plane equations along with the normals and centroids are available, that information can be employed to make the classification as described in the proposed Algorithm 7.

To identify a translation, we compare the distance between centroids on the surfaces. If the distance exceeds a minimum threshold value, it indicates a translation be-

Algorithm 6 Plane Matching

Step 1: Let Π_X^i and Π_Y^j be planes of source and target point clouds X, Y. We denote n_X^i, n_Y^j as their respective normal vector and c_X^i, c_Y^j as their respective centroids.

Step 2: For all tuple (i, j), compute: the distance between planes $d(i, j)$ and the alignment $a(i, j)$.

$$d(i,j) := \|c_X^i - c_Y^j\|$$
$$a(i,j) := \langle n_X^i, n_Y^j \rangle$$

Step 3: For each i, find j^* such that:

$$j^*(i) = \underset{j}{\operatorname{argmin}}\, a(i,j) \quad \texttt{Subject to:} \quad d(i,j) \leq trsh_d$$

Step 4: For each i, match the corresponding plane together $(\Pi_X^i, \Pi_Y^{j^*(i)})$

tween the planes. On the other hand, to detect a rotation, we analyze the alignment of the normals. If the scalar product of the normal vectors falls below a certain threshold, it suggests a rotational transformation. If neither a significant translation nor rotation is observed, both planes are classified as having no transformation.

However, it is important to note that this approach has limitations. It is only capable of classifying the transformation into three distinct classes: translation, rotation, or no transformation. In practice, during construction processes, a wider range of changes and deformations beyond these three categories are exhibited.

Algorithm 7 Transformation Classification

Step 1: Let Π_X^i and Π_Y^j be a pair of correspondent planes of source and target point clouds X, Y. We denote n_X^i, n_Y^j as their respective normal vector and c_X^i, c_Y^j as their respective centroids.

Step 2: For all correspondent tuple (i, j), compute the distance between planes $d(i, j)$ and the alignment $a(i, j)$.

$$d(i,j) := \|c_X^i - c_Y^j\|$$
$$a(i,j) := \langle n_X^i, n_Y^j \rangle$$

Step 3: Classify the transform as follow:

- If $d(i,j) \geq trsh_d$ and $a(i,j) \geq trsh_n$: A translation is applied to Π_Y^j compared to Π_X^i.
- If $d(i,j) \geq trsh_d$ and $a(i,j) < trsh_n$: no transform is applied to Π_Y^j.
- Otherwise, a rotation is applied to Π_Y^j compared to Π_X^i.

Chapter 5

Deep Learning method for Changes Detection

The previous approach, called the classical approach, has a big drawback. For each new shape targeted for change detection, such as spheres, and cylinders, it necessitates a specific implementation, resulting in the need for adaptation of the entire pipeline.

Recent progress in deep learning methods has shown promising results in the classification and semantic segmentation of complex objects or scenes [27]. Consequently, this second approach investigates the potential of a neural network to tackle the previous limitation, allowing the detection of changes in complex shapes thereby providing a realistic framework for real-world scenarios.

In this chapter, we first introduce key concepts of neural networks. It is followed by a review of three networks, namely PointNet, PointNet++, and FlowNet3D, which are relevant to this master thesis. Finally, we propose a network tailored for the change detection purpose.

5.1 Fundamentals of Neural Networks

This section provides a comprehensive overview of the fundamental concepts of neural networks and their training process [28].

5.1.1 Overview of Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by the structure and function of biological brains. They consist of interconnected nodes, called neurons or units, organized into layers. Each neuron receives inputs, processes them, and produces an output signal.

Neurons are the basic building blocks of neural networks. Each neuron applies a mathematical operation to its inputs, typically a weighted sum followed by an activation function. The activation function introduces non-linearities and allows neural networks to model complex relationships in the data.

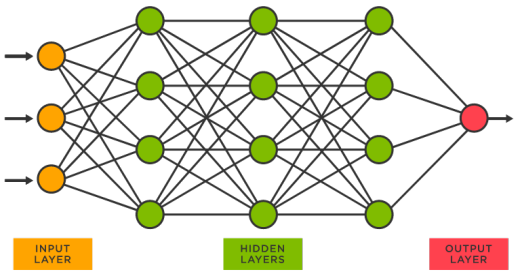


Figure 5.1: ANNs basic architecture [29]

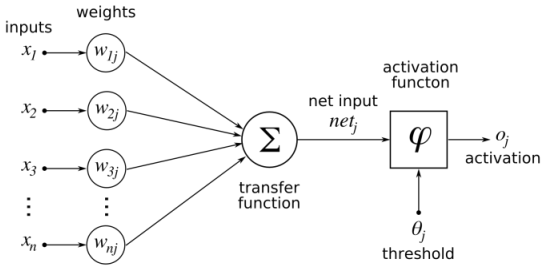


Figure 5.2: Interaction for a single neuron [30]

5.1.2 Feedforward and Backpropagation

Feedforward is the process of passing data through a neural network from the input layer to the output layer. During feedforward, each neuron's output becomes the input for the neurons in the subsequent layer. Backpropagation is the algorithm used to train neural networks by adjusting the weights based on the difference between the predicted output and the ground truth.

5.1.3 Training and Optimization

Training a neural network involves optimizing its weights to minimize a defined loss function. This is done through an optimization algorithm, such as gradient descent, which iteratively adjusts the weights based on the computed gradients. The learning rate and the choice of optimization algorithm play crucial roles in training efficiency and convergence.

5.1.4 Loss Functions

Loss functions quantify the discrepancy between the predicted outputs and the ground truth labels. Different tasks and applications require specific loss functions, such as mean squared error for regression problems or cross-entropy loss for classification tasks.

5.2 PointNet

PointNet is a deep learning architecture designed to process point clouds. It can be supplemented with attributes such as colors, intensity, or normals. It is one of the pioneering architectures that achieves good results on many classification tasks and semantic segmentation tasks [31].

The authors of PointNet identified three key challenges in processing point clouds:

- First, point clouds lack a specific structure and are unordered, unlike grids or meshes.
- Secondly, each data point interacts directly with its local neighbors forming structures. This interaction becomes particularly significant in segmentation tasks, where local structures may correspond to different classes.
- Finally, the point cloud should be invariant under permutation and transformation. This signifies that two point clouds can represent the same object, although the points are described in a different order. Additionally, invariance

under transformation means rotation or translation of a whole point cloud can represent the same properties.

5.2.1 PointNet architecture

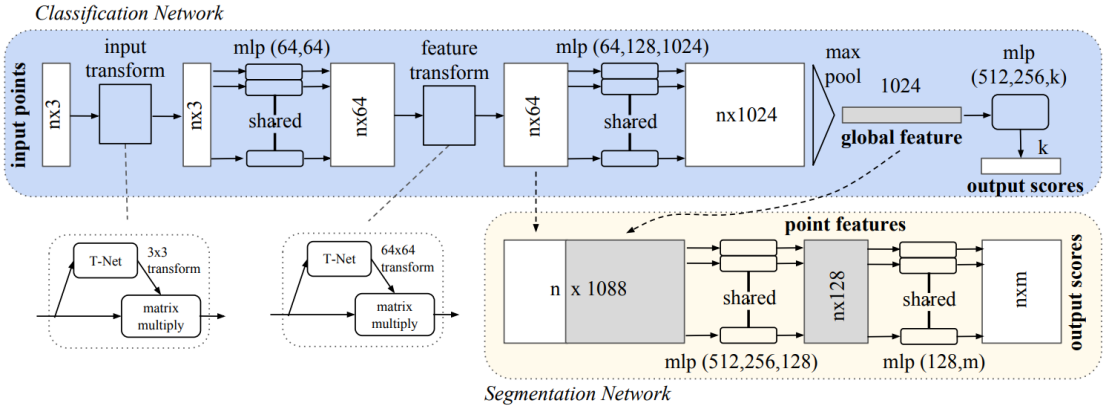


Figure 5.3: PointNet Architecture [31]

PointNet offers a solution that tackles the previously mentioned challenges. The architecture of PointNet, illustrated in Figure 5.3, can be separated into two networks: the classification and segmentation one. As their name suggests, each network has different objectives and outputs.

Classification

As shown in Fig 5.3, the classification network undergoes an 'input transform', followed by a shared multi-layer perceptron (MLP) and a 'feature transform'. Once those steps are done, the network obtains $n \times 64$ local features .

The 'input transform' and 'feature transform' are essential to the network as they predict a transform that projects input points (in the input transform) or input features (in the feature transform) in a canonical space ensuring points and features invariance under transformation which is one of the challenges of points data mentioned earlier. Those essential transformations have recourse to the **Joint**

Alignment Network, which utilizes a network called T-Net that predicts the transformation matrix to achieve invariance.

For classification, the main interest is the global features of the point cloud, enabling discrimination between different classes. To achieve this, local features enter a shared MLP followed by a **max pooling** layer. This max pooling layer is also critical for the network. Selecting the maximum value for each feature ensures invariant global features of the point cloud.

Once the global feature vectors are obtained, it goes through a fully connected MLP layer to obtain an output classification score. The predicted class is simply the one where the score is maximum.

Segmentation

The segmentation network can be viewed as an extension built upon the preceding classification network. It concatenates global and local features to generate points features, which will be processed through a MLP, to produce per-point output scores. By utilizing both local and global features, it achieves scene understanding, capturing local and global interaction between points.

5.2.2 Why PointNet

Our main motivation for utilizing PointNet, and its improvement PointNet++, stems from their pioneering ability to directly process point clouds. PointNet has shown good results on different benchmarks. For example, in classification task, the method reached an accuracy of 89.2% on the ModelNet40 dataset, which encompass synthetic object split into 40 categories [32]. For semantic classification, it performed with a mean accuracy of 78.9% on S3DIS, an indoor scene dataset consisting mostly of rooms and hallways relevant for our purpose [33].

Moreover, its value lies in its ability to directly learn from raw point cloud data,

eliminating the requirement for manual feature engineering or complex preprocessing steps. This aspect becomes especially advantageous when compared to other networks, such as RandLA-Net, that typically rely on additional structured inputs or handcrafted features derived from complex preprocessing pipelines [34].

Considering the satisfactory results, the availability of implementations, and the user-friendly nature of the network, we decided to investigate PointNet for our research.

5.3 PointNet++

PointNet++, sometimes denoted as PointNet 2, is built upon the original PointNet architecture by addressing some of its limitations and improving its performance. The main enhancement introduced by PointNet++ is the ability to capture local features from the point cloud data in a structured and hierarchical manner [35].

Instead of processing each point independently, PointNet++ introduces a hierarchical neural network architecture that takes into account the local structures and relationships between points. It achieves this by dividing the input point cloud into a series of nested partitions called "local regions."

5.3.1 PointNet++ architecture

The PointNet++ architecture is characterized by a hierarchical structure comprising several stages referred to as set abstraction levels. Each set abstraction level encompasses three key layers: a 'sampling layer', a 'grouping layer', and a 'PointNet layer'.

In the sampling layer l, the purpose consists in selecting a predetermined number of points N_l from the input data. The selection of these points is not random. Instead, a technique called **farthest point sampling** selects points. This technique ensures the N_l chosen points evenly cover the input data and thus, captures its essential

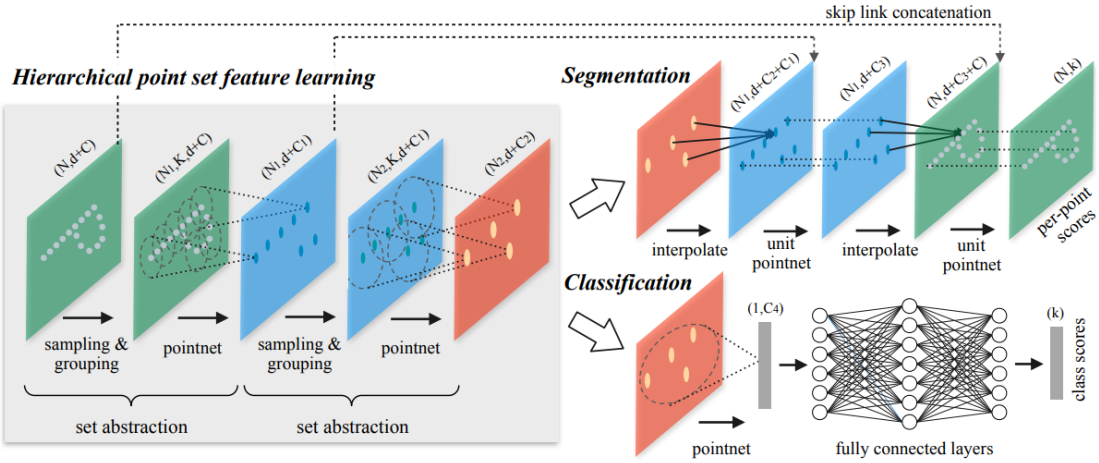


Figure 5.4: PointNet++ architecture [35].

structure while reducing the computational complexity.

Following the sampling layer, the grouping layer search for local points surrounding the selected N_l centroids. It is achieved through a method called **ball query** which associates each centroid with its neighboring points within a specified radius.

Within each local region, PointNet++ applies the same set of operations as PointNet to extract C global features for each N_l point. The output of the set abstraction layer consists of N_l points, each represented by a vector of size $d + F$ where d represents the input dimension, typically of size 3 or 6, and F is the number of global features.

By applying iteratively the set abstraction layer, PointNet++ captures a hierarchy of features in a more structured manner. At each subsequent level, the considered regions become more global and capture increasingly global details. The final output of the multiple levels is a collection of hierarchical features that encompass both local and global information for a subsampled input point cloud.

Classification

Regarding classification tasks, the output after applying different set abstraction layers is a set of shape $(N_l, d + C)$ capturing both local and global features. On this set, a PoinNet layer for classification can be applied just as in Figure 5.3.

Semantic segmentation and Point Feature Propagation

When it comes to semantic segmentation tasks, directly applying PointNet to the output of the set abstraction layers is not suitable. This is because the N_l points obtained from the layers represent a subsampling of the input point set and it requires an additional upsampling step.

To tackle this challenge, PointNet++ adopts a hierarchical propagation strategy utilizing distance-based interpolation and skip links across different levels. At each feature propagation level, the point features of size $(N_l, d + F)$ are propagated to N_{l-1} points, where N_{l-1} and N_l denote the point set sizes of the input and output of the set abstraction level l ($N_{l-1} \geq N_l$) respectively.

Feature propagation is achieved by interpolating the feature values of N_l points at the coordinates of the N_{l-1} points. Inverse distance weighted average interpolation, based on the k nearest neighbors, is utilized as shown in Equation 5.1. The interpolated features on the N_{l-1} points are then concatenated with the previous point features obtained from the set abstraction level $l - 1$.

$$f^{(j)}(x) = \frac{\sum_{i=1}^{k} w_i(x) f_i^{(j)}}{\sum_{i=1}^{k} w_i(x)} \quad \text{where} \quad w_i(x) = \frac{1}{d\left(x, x_i\right)^2}, j = 1, \ldots, F \tag{5.1}$$

The concatenated features are passed through a 'unit PointNet', which is equivalent to the PointNet segmentation network illustrated in Fig 5.3.

This process is repeated until the features are propagated to the original set of points, ensuring that the per-point scores and labels are obtained for the entire point cloud.

5.4 FlowNet3D

Flownet3D is a deep learning architecture designed specifically for point cloud data, enabling the estimation of 3D motion between two point clouds. It builds upon the concept of optical flow, which is a technique used in computer vision to estimate the motion of pixels between consecutive frames in a video sequence [36].

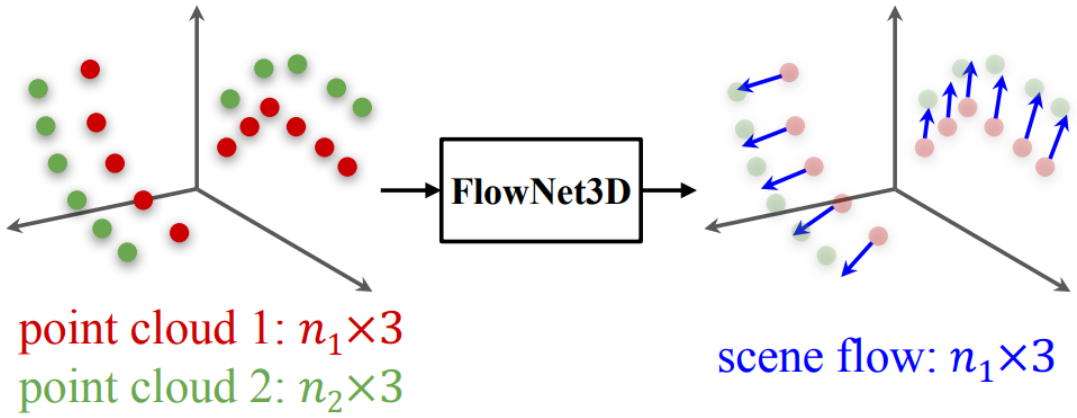


Figure 5.5: Scene flow estimation from point clouds [36]

The goal of FlowNet3D is to estimate the correspondences between points in two input point clouds and infer the 3D motion vector for each point. This information can be useful for various applications, such as motion tracking, scene reconstruction, and understanding the dynamics of 3D objects or scenes.

5.4.1 FlowNet3D architecture

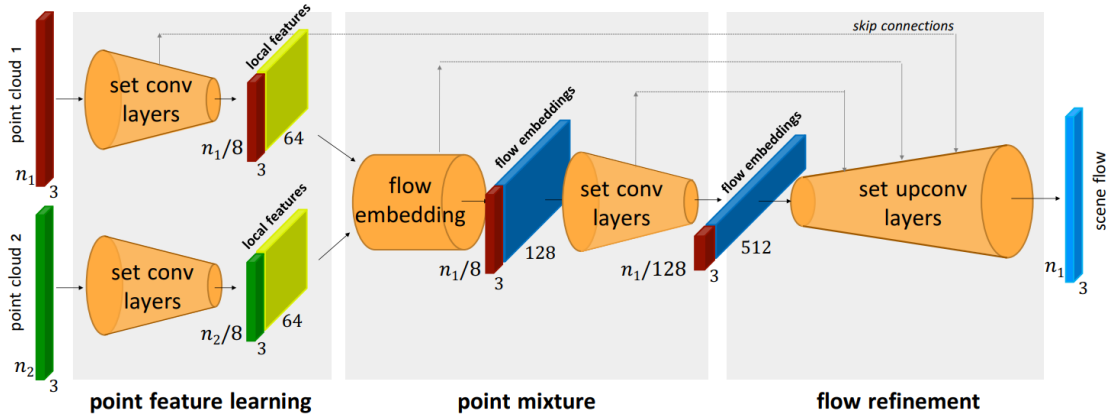


Figure 5.6: FlowNet3D Architecture [36].

The network consists of three key modules: 'point feature learning', 'point mixture', and 'flow refinement'. Each module is supported by specific layers, including 'set conv layers', 'flow embedding layers', and 'set upconv layers'.

The point feature learning is done by a set conv layer. This layer is based on the PointNet++ set abstraction layer architecture. It takes as input a point cloud and outputs a sub-sampled point cloud with hierarchical features. As previously explained, this layer uses the farthest point sampling and ball query method to select representatives to extract local features which capture spatial locality and translation invariance.

The point mixture module uses a flow embedding layer to mix two point clouds and a set conv layer. Flow embedding aggregates flow votes from neighboring points in the second frame to estimate point motions. It considers both geometric feature similarities and spatial relationships to encode point motions. It employs a non-linear function and element-wise max pooling to compute flow embeddings.

The flow refinement module involves the set upconv layer, which up-samples the

final flow embedding obtained. The set upconv layer aggregates neighboring source points' features using a similar strategy as the set conv layer but with a different local region sampling approach. Similarly to PointNet++ in Fig 5.4, the network includes skip connections that concatenate set conv output features to facilitate information flow.

5.4.2 Why FlowNet3D

Estimation of 3D motion between two point clouds is relevant in the context of change detection between point clouds. These motion flows represent desirable features that can be leveraged for classifying or segmenting various changes. FlowNet3D is a network specifically designed for this purpose.

Furthermore, it is noteworthy that the authors of FlowNet3D are also involved in the development of PointNet and PointNet++, revealing coherence and consistency among these networks.

Additionally, different implementations with pre-trained models are available which allows us to rapidly evaluate the proof of concept in this master thesis. These factors, collectively combined, contribute significantly to the decision of selecting this network.

However, it is important to acknowledge that there exists a set of less established methods that demonstrated superior performance over FlowNet3D [37]

5.5 Proposed Network

The goal of our network is to either classify or segment construction faults within an input point cloud based on a reference point cloud representing the same scene. For that purpose, we developed a network designed to detect faults in a manner that is transferable and easily adaptable when new types of faults are introduced.

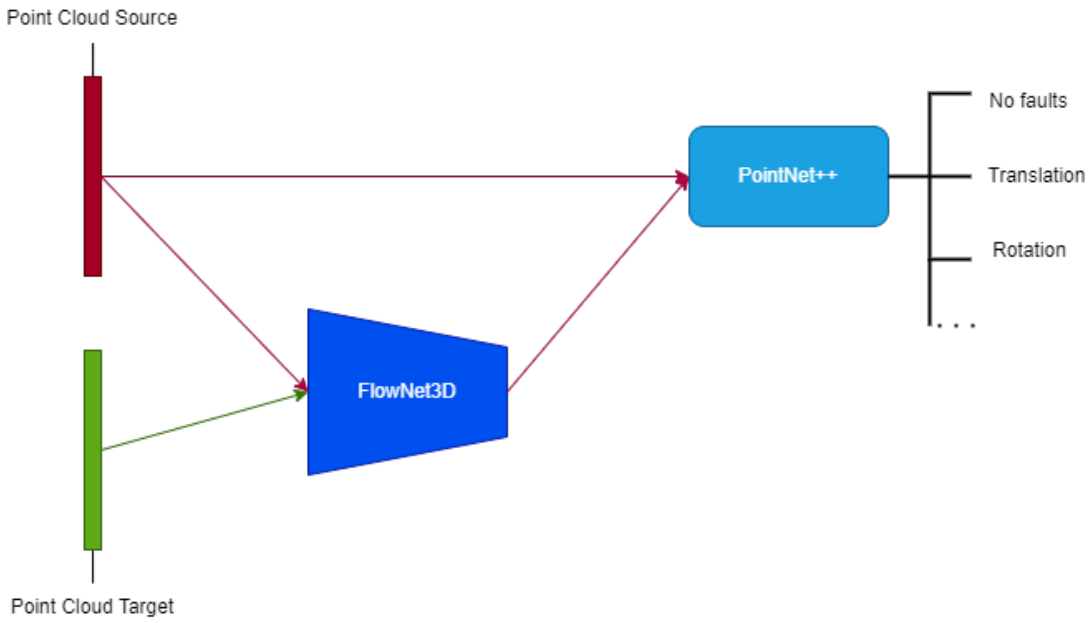


Figure 5.7: Our construction fault classifier/segmentation network

The proposed network architecture comprise two main components: FlowNet3D and PointNet++. It takes a pair of input point clouds, representing different frames of the same scene, and produces a semantic segmentation or classification of potential changes that have occurred.

First, FlowNet3D is applied to the pair of point clouds. FlowNet3D estimates the scene flow, which captures the points' motion between those two frames by leveraging the spatial information in both point clouds. The displacement vectors provide temporal information about the motion between the frames.

Next, the estimated scene flow is then combined with the source point cloud and fed into PointNet++. This integration allows the network to take into account spatial and temporal information, crucial for change detection.

Finally, PointNet++ produces the desired output based on the specific task at hand, whether it is a classification or a semantic segmentation.

Chapter 6

Implementation and Results

This chapter aims to provide a comprehensive overview of the implementation and results of previous models. It begins by describing the dataset generated specifically for evaluation or training purposes and used evaluation metrics. Next, we present the implementation of the deterministic model elaborating the process, design choices, and the obtained results. A similar analysis is undertaken regarding the deep learning model. Finally, we conclude this chapter by summarizing the results obtained from both models.

6.1 Dataset

Detecting changes in construction sites between theoretical and experimental point clouds is highly specific and there is no public dataset available. In order to train our proposed network and compared it with the classical method, we need to have a dataset. Therefore, we attempted to create a point cloud dataset aiming to reflect real surroundings such as rooms and hallways in a simplified way.

Since buildings' main geometry is planar, we generated a point cloud that represents planar structures. These planes are defined by three of their vertices (p_1, p_2, p_3). For each point in the cloud, $x \in X$, they are uniformly distributed on the plane the relation Eq 6.1. A room is modeled and simplified as a set of planes assembled

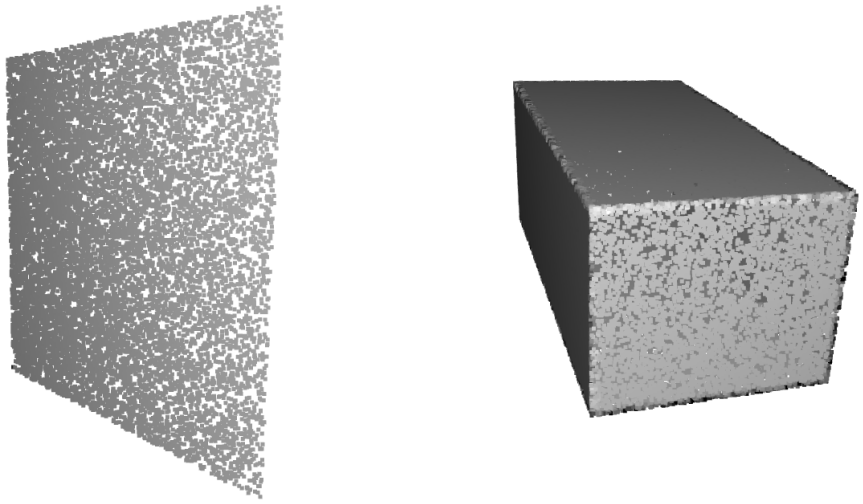


Figure 6.1: Two examples of point clouds: On the left, a plane that represents walls. On the right, is a rectangular cuboid models a room.

together to form a rectangular cuboid as shown in Figure 6.1.

$$x = \lambda_1(p_3 - p_1) + \lambda_2(p_2 - p_1) \qquad \forall x \in X \text{ and } \lambda_1, \lambda_2 \in U([0,1]) \tag{6.1}$$

Different transform has been implemented for our change detection purpose. Such transformations include translation and rotation around coordinate axis. It can be applied to a source point cloud X to obtain a target point cloud Y using a transformation matrix $T_{X \to Y}$ explicitly described in Eq 1.2. For all point $x \in X$, their corresponding transformed points $y \in Y$ follow the Eq 6.2.

$$y = T_{X \to Y}\ x \qquad \forall x \in X \tag{6.2}$$

Based on this method to generate point clouds, we create two datasets: the classification dataset and the segmentation dataset. Both datasets contain three classes: one for each of the aforementioned transforms, along with a class where no transformation is applied, often referred as the 'base' transform in this work. Moreover, variations of those generated datasets where a noise following a Gaussian $\mathcal{N}(0, 0.015)$ are also available in order to simulate measurement noise using LiDAR, which is up to $\pm$3cm.

6.1.1 Classification dataset

The classification dataset is composed solely of pairs of planes where each element is composed of a source plane and a modified plane. As shown in Figure 6.2, the modified one, shown in dark gray, is a either translated or rotated version of the source one.

In order to have a rich dataset, where rotation angle can range anywhere up to a dozen degrees along different axis, where translation is up to a dozen of centimeters, and where plane dimension can vary strongly, we automated this process creating random transform with random values for translation, rotation and plane dimension.

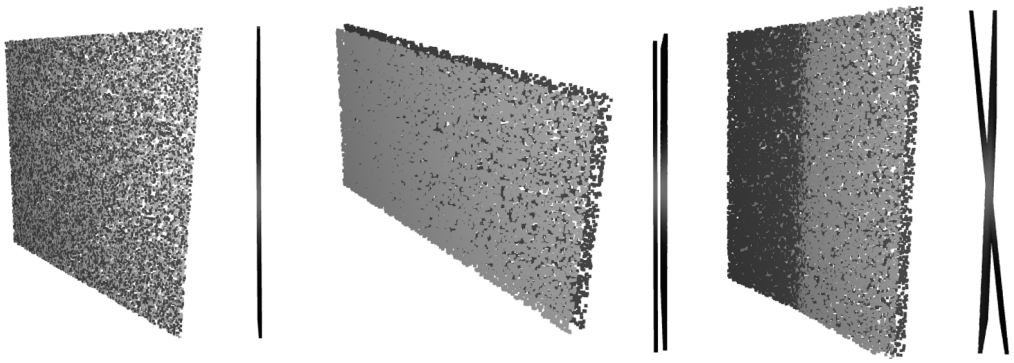


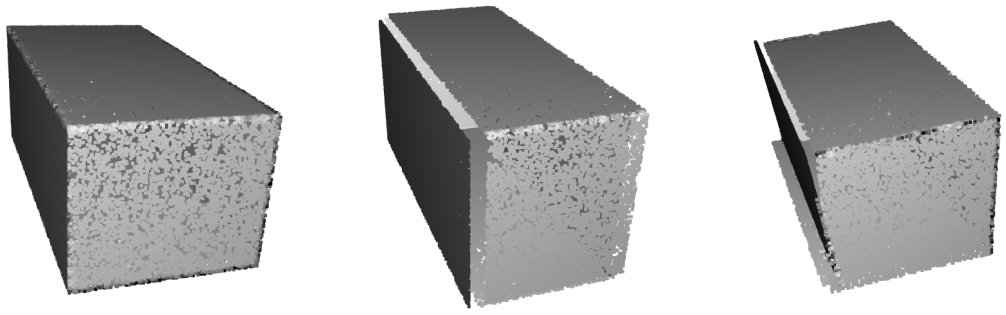
Figure 6.2: Examples of elements in the classification dataset with side and top view. On the left, no transform is applied. In the middle, a translation is performed. On the right, a rotation occurred.

6.1.2 Segmentation dataset

The segmentation dataset is composed of pairs of cuboids representing respectively a source and a modified cuboid. Figure 6.3 illustrates one element of the pair present in the dataset. The source point cloud can be considered as Figure 6.3a while modified ones are Figure 6.3b or Figure 6.3c. We observe that on the modified point cloud, transformation can appear for each plane composing it. In other words, a modified cuboid can have multiple transformed planes. Figure 6.3b shows a

translation on one of its planes, while 6.3c shows a rotation for the same plane.

In similar regard to the classification set, a rich dataset with different translation directions, rotation angles/ axis, and cuboid sizes is generated using uniformly random values. Moreover, a variant of this dataset containing noise is also available.



(a) (b) (c)

Figure 6.3: Examples of elements inside the segmentation dataset.

6.2 Evaluation metrics

Performance evaluations are essential to assess the correctness of a network. Whether dealing with a classification or segmentation problem, various tools, and metrics can be employed.

6.2.1 Confusion Matrix

A confusion matrix is a matrix containing four different elements as shown in Table 6.1: True Positive (TP) and False Positive (FP) refer respectively to correctly and wrongly predicted positive elements while True Negative (TN) and False Negative (FN) are accurate and inaccurate predicted negative elements [38].

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Table 6.1: Confusion Matrix

Accuracy

Based on the elements in the confusion matrix, the accuracy metric measures the overall correctness of a model's predictions. It calculates the ratio between correctly classified elements and the total number of available elements [38]:

$$\text{Acc} = \frac{TP + TN}{TP + FP + TN + FN} \tag{6.3}$$

However, this metric has its limitation. When dealing with an unbalanced dataset and one class dominate in the dataset, if the model classifies all instances as belonging to the dominant one, it can achieve a high accuracy score while lacking relevance.

6.2.2 Precision and Recall

Precision is the ratio between correctly predicted instances and all the instances predicted as positive. It indicates the relevance of the model's prediction over the predicted instance.
Recall, on the other hand, is the ratio between correctly predicted instances overall actual instances and it is a useful indicator of the model prediction over the actual instance [38].

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.4}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6.5}$$

6.2.3 F1-score

Precision and Recall are complementary metrics. Precision cannot measure the amount of False Negatives while Recall cannot measure the amount of False

Positives. To capture both aspects, the f1-score combines both precision and recall by computing their harmonic mean [38].

$$F1 = \frac{2}{\text{Precision}^{-1}\text{Recall}^{-1}} = \frac{2\,TP}{2\,TP + FP + FN} \tag{6.6}$$

6.2.4 Intersection over Union

Intersection over Union (IoU), also known as Jaccard index, is defined as the ratio between the overlapping region between the ground truth and the prediction, and the union of those 2 sets [39]. IoU measures both similarity and diversity of the predictions and thus is relevant for semantic segmentation tasks when multiple classes are present.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of union}} = \frac{TP}{TP + FP + FN} \tag{6.7}$$

6.3 Classical Model

6.3.1 Implementation

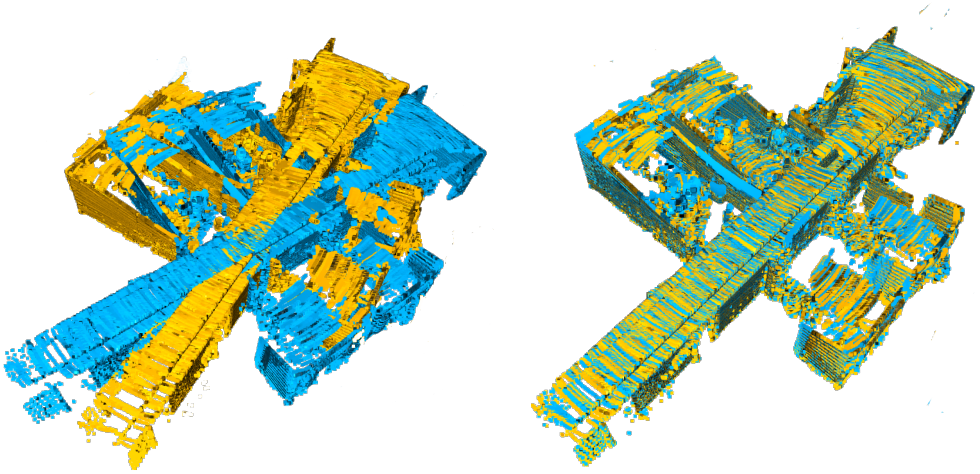
Global Registration

The implementation global registration is performed using Open3d library. Pre-built methods perform the improved version of RANSAC (section 2.3) using pruning and a nearest neighbor search. It is followed by an ICP point-to-plane algorithm (section 2.5) to refine the obtained transformation matrix [16].

Figure 6.4 illustrates the registration performed on a building's floor point cloud. On the left, in Figure 6.4a, there are 2 point clouds, one colored in blue and the other colored in yellow, which are not referenced in the same frame because the initial position of Spot robot is different. After the registration, both point clouds are expressed in the same frame as depicted in Figure 6.4b.

Plane Detection

During the Plane matching phase, 2 algorithms were implemented: RANSAC for plane detection and also Generalized Hough Transform.



(a) (b)

Figure 6.4: Global registration.

The GHT described in Algorithm 4 was implemented and tested on an artificial cuboid and provided correct results under restrictive conditions: the point cloud must be free of noise, the cuboid's planes must be aligned with the frame axis and there must be sufficient discretization of the parameter space.

Once noise is introduced, the Algorithm 4 fails at **Step 3** which consists of finding local maxima in the accumulator. It tends to detect numerous redundant walls. To tackle this issue, a potential solution is to apply clustering techniques like DBSCAN to group the accumulator cells. Then, the global maxima within each cluster can be selected. Although this attempt provides improved performance for the artificial point cloud, it fails to segment complex scenes such as those representing buildings.

In light of the inconclusive results obtained thus far, the RANSAC approach is explored. The Algorithm 5 yields satisfactory results both on our artificial cuboid and also on the building as shown in Figure 6.5. However, when noise appears in the point cloud, the proposed plane does not always align accurately. To enhance

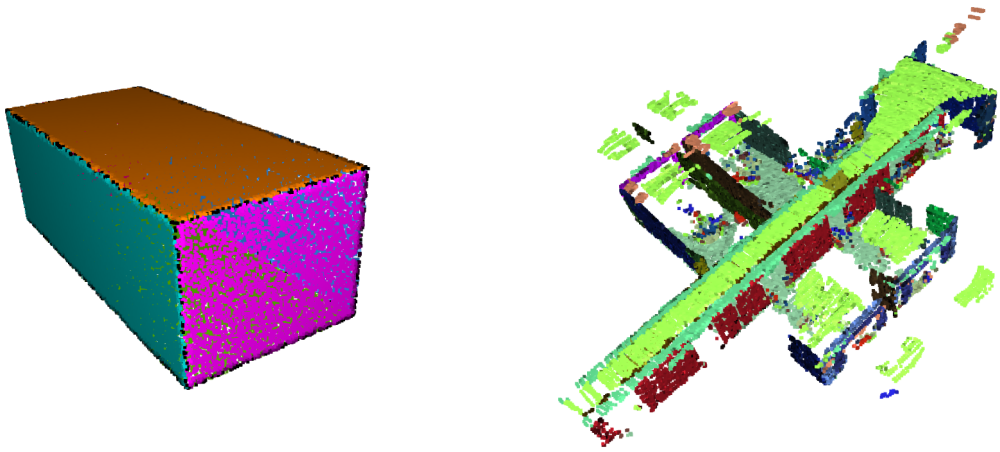


Figure 6.5: Plane Detection

the accuracy of results, a condition in **Step 5** was introduced. This condition involves selecting the plane where the mean squared error (MSE) between the candidate plane and its identified inliers is minimized.

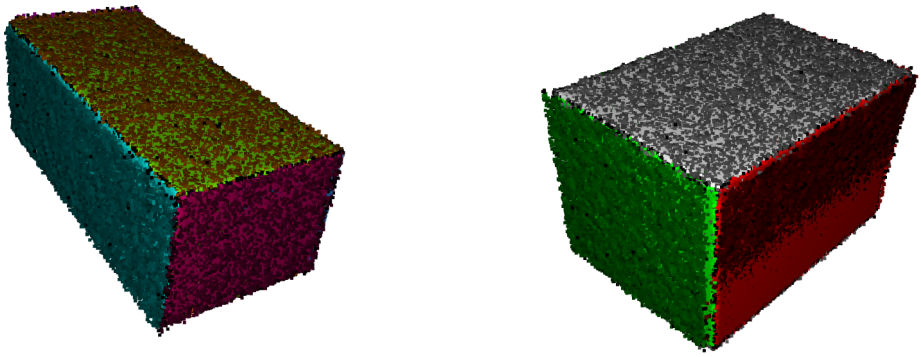
Plane Matching

Plane matching process follow Algorithm 6. The only parameter of the algorithm is the threshold distance between a candidate pair of planes denoted as $trsh_d$. For our implementation, a value of approximately 0.5m is chosen as it demonstrates a balance between being too restrictive and too relaxed. Figure 6.5 describes the matching obtained for a cuboid. Different colors represent different planes while the light accent of a color represents the source plane and the darker accent of the same color represents the matched plane in the target cloud.

Change detection

Change detection is introduced by Algorithm 7 and it involves the utilization of two parameters that enable the fine-tuning of classification: a threshold distance denoted as $trsh_d$, and a threshold for normal alignment denoted as $trsh_n$.

Tuning those parameters reveals to be a complicated task as it relies on the point



(a) Plane matching

(b) Change Detection

Figure 6.6: Plane matching and change detection

cloud's characteristics, the quality of the estimated point's normal, and proposed planes by RANSAC. For our specific case, these parameters are chosen to fit as much as possible in both classification and segmentation tasks when the dataset point clouds are either noisy or not. After our experiment, the following values provide robust results for these different evoked scenarios:

$$trsh_d = 0.05$$
$$trsh_n = 0.999$$

6.3.2 Classification Results

The classification implementation is tested on the dataset containing solely noisy planes with different faults. A noise of ±3 cm has been added to be a more representative real-world point cloud. Results obtained from the dataset without noise can be found in Appendix A.1.

Table 6.2 summarizes the model's performance on the accuracy, precision, recall, and f1-score metrics. It shows all classes are globally well-identified. However, there is confusion between the rotation and base class as the rotation class is often misclassified as a base class when looking at recall and precision. This is particularly visible in Table 6.3 which represents the actual confusion matrix that

has been normalized over all the population. We also observe that a small fraction of base is actually predicted as belonging to the translation class.

Overall, the method performs well and achieves around $0.85 - 0.87$ in all metrics. This is a piece of evidence proving that implementation is a possible way to tackle the change detection problem we defined. The obtained performance is highly sensitive to the threshold parameters and also to the dataset.

Evaluation metrics	Base	Translation	Rotation	Average
Accuracy	0.911	1	0.634	0.853
Precision	0.806	0.844	1	0.872
Recall	0.911	1	0.634	0.853
f1-Score	0.855	0.915	0.776	0.848

Table 6.2: Classification result with the classical method.

	Predicted base	Predicted translation	Predicted rotation
Actual base	0.911	0.089	0
Actual translation	0	1	0
Actual rotation	0.344	0.022	0.634

Table 6.3: Classification confusion matrix with the classical method.

6.3.3 Segmentation Results

For the segmentation task, the dataset is composed of rectangular cuboids of random length, width, and height representing rooms. For each point of the point cloud, a noise of ± 3 cm is added. Results for the dataset without noise can be found in Appendix A.1.

As one could expect, the accuracy of the segmentation task given in Table 6.4 is closely related to the classification one. Indeed, the method decomposes the

rectangular cuboid into planes and classifies the different changes.

The Intersection over Union metric (IoU), is less relevant since in reality, the method classifies plane per plane. But in this case, it offers a comparison measure to the deep learning approach described earlier. Overall, the same conclusion can be made as the classification task. Transforms are globally well determined although some confusion is visible in Table 6.4.

Evaluation metrics	Base	Translation	Rotation	Average
Class accuracy	0.938	0.905	0.600	0.814
Class IoU	0.723	0.840	0.566	0.710

Table 6.4: Segmentation Results

6.4 Deep Learning Model

6.4.1 Implementation of the proposed network

Implementation of FlowNet3D[40] and PointNet++[41] leveraged existing resources and frameworks such as their respective Pytorch implementation. These well-established models have open-source code available, which was adapted and customized to suit our specific data requirements. The adaptation process involved implementing PyTorch's abstract class `Dataset`. For FlowNet3D, we implemented one `Dataset` class, while for PointNet++, we implemented two separate `Dataset` classes for classification and segmentation.

The open-source code of FlowNet3D also provided a pre-trained model trained on the FlyingThings3D dataset. FlyingThings3D is a popular benchmark dataset used for optical flow estimation in 3D scenes. It comprises synthetic stereo sequences that include ground truth optical flow, depth maps, and color images. The dataset focuses primarily on outdoor scenes and encompasses a variety of objects typically encountered in real-world environments [42]. Although the pre-trained model may

not be directly suited to our construction context, we were unable to dedicate sufficient time or resources to train the FlowNet3D network on our specific data. Training the FlowNet3D model on our own dataset remains a potential avenue for future improvement.

PointNet++ training was undertaken from scratch for both classification and segmentation tasks, following a standard procedure. Initially, the dataset was divided into training and validation subsets to ensure effective model evaluation. The models were trained using an appropriate optimizer and loss function, with regular adjustments to model parameters based on the obtained results and convergence rate. Multiple epochs were employed during the training process to capture essential patterns and features within the data. Our objective was to harness the capabilities of PointNet++ to accurately classify and segment objects in our dataset. However, we encountered challenges with segmentation, as the model failed to learn from our data effectively. To address this issue, we opted to simplify the network architecture by reducing the number of abstraction and feature propagation layers, deviating from the original open-source implementation. Our intuition proved to be significant, yielding satisfactory results in the segmentation task.

6.4.2 Classification Results

The implemented PointNet++ classification network was evaluated using 4 different datasets ranging from the least realistic to the most realistic. These datasets are used in the following cases:

- **Case 1:** In this scenario, the classification dataset does not contain any noise, and the flows between corresponding points are theoretically computed. This case allows us to prove the network is well-built and can work in ideal condition.
- **Case 2:** Noise is added to the previous dataset making it more realistic of real-world surroundings while still using the theoretical flow. This scenario provides an upper bound on the expected results in the real world using our designed network. If the flow predicted by FlowNet behaves exactly as

desired, this can be a result reachable by the network.

- **Case 3 and 4:** These two cases are variations of the first two, but instead of using theoretical flows, the predicted scene flow of the pre-trained FlowNet3D is used. Case 4 allows us to observe a lower bound on the expected behavior in real-life situations.

In the following section, we will analyze the results for Case 2 and Case 4 since they provide lower and upper bounds, respectively, on the real behavior of the model.

Case 2: Noisy dataset with theoretical flows

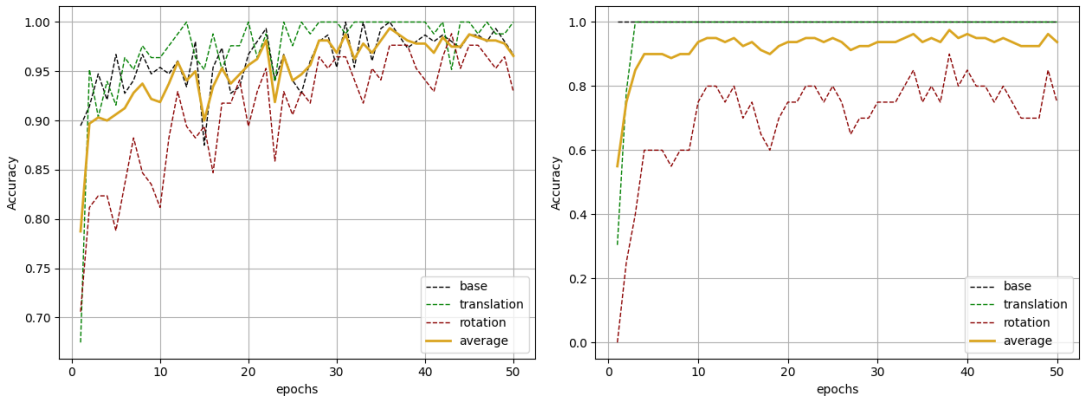
The results for accuracy, precision, recall, and f1-score are shown in Figures 6.7, 6.8, 6.9, and 6.10 respectively. Each figure represents the convergence of these metrics during each training epoch. The network demonstrates improvements in all metrics as the number of epochs increases, indicating a learning phenomenon.

A direct comparison between the f1-score and accuracy of Case 1 (see Appendix **??**) and Case 2, shows slightly better performance when no noise was applied. The mean f1-score stays around 0.97 and 1.00 for both cases, in their respective best model. This suggests that even if the point cloud contains noise, the classification network still demonstrates good results as long as the theoretical flows are accurately computed.

Analyzing the precision (Figure 6.8) and recall (Figure 6.9), the following conclusions can be drawn for each class. The network accurately identifies translation, as they exhibit high recall and precision. However, there appears to be confusion between the rotation class and the base class. The base class demonstrates high recall and low precision, while the rotation class shows low recall and high precision in both the training and testing sets. This suggests that some rotation transformations are misclassified as base. Overall, rotation transformations seem to be more challenging to detect and perform poorly in many metrics.

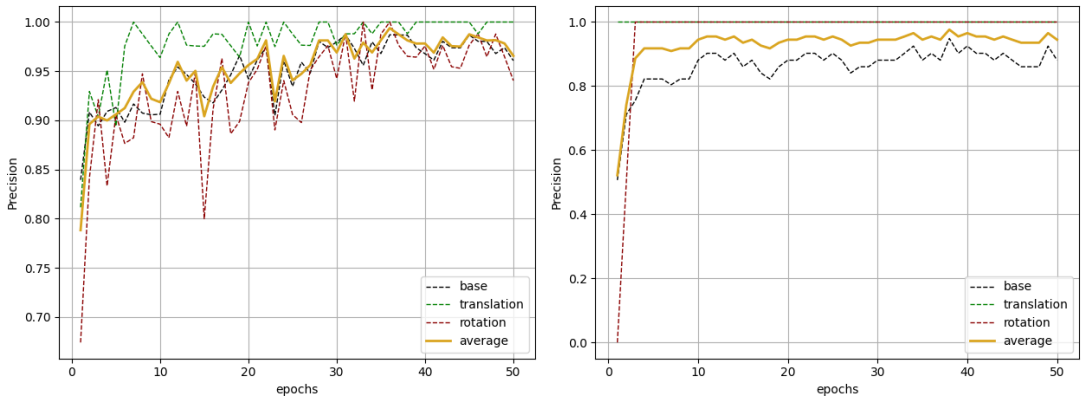
Evaluation metrics	Base	Translation	Rotation	Average
Accuracy	1.000	1.000	0.900	0.975
Precision	0.949	1.000	1.000	0.976
Recall	1.000	1.000	0.900	0.975
f1-Score	0.974	1.000	0.947	0.975

Table 6.5: Case 2 - Classification: Best model after 50 epochs. Obtained at epoch 38/50



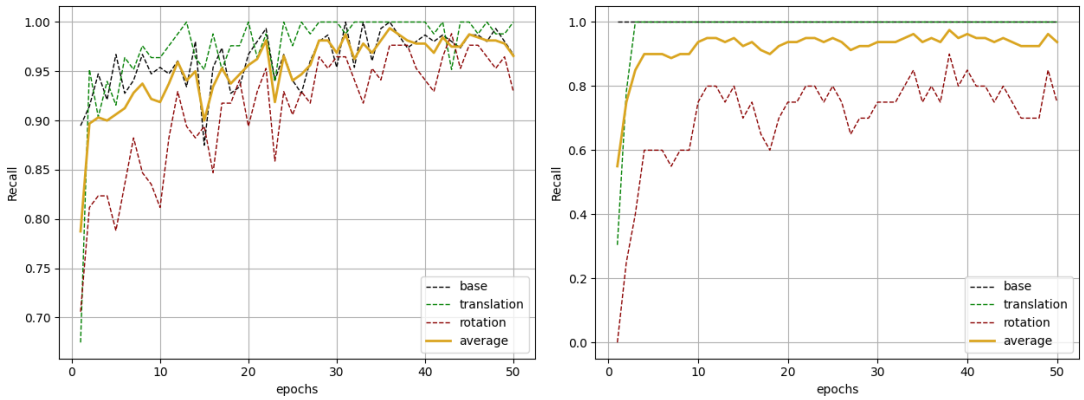
(a) Train (b) Test

Figure 6.7: Case 2 - Classification: The accuracy metric steadily increases in both the training and testing sets, with the translation class being the most accurate, followed by the base and rotation classes.



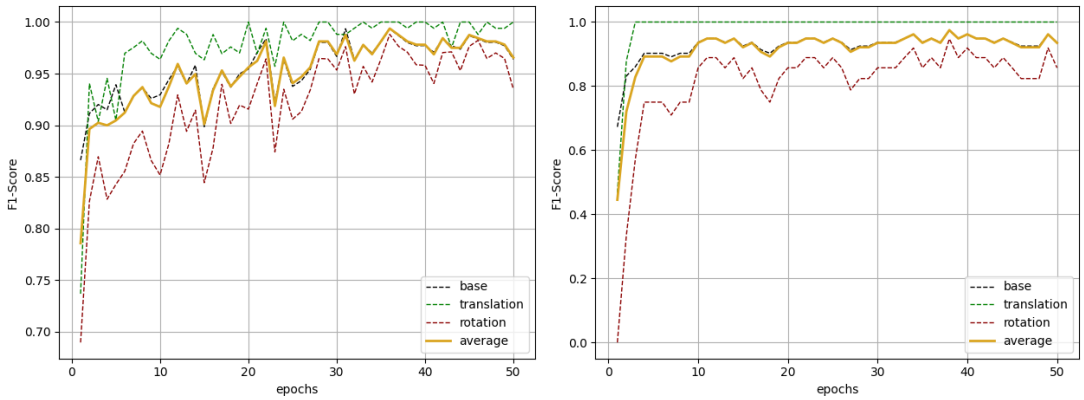
(a) Train (b) Test

Figure 6.8: Case 2 - Classification: The precision metric shows a general increase for all classes in the training set as the iterations progress. In the test set, both the rotation and translation classes achieve a precision of 1, while the precision of the base class ranges between 0.8 and 1.



(a) Train (b) Test

Figure 6.9: Case 2 - Classification: The average recall reached its peak at 0.98 in the training set. In the testing set, the recall for rotation is approximately 0.8, while both the base and translation classes achieve a recall of around 1.



(a) Train (b) Test

Figure 6.10: Case 2 - Classification: The average f1-score reach its peak at 0.98 and 0.95 in training and testing dataset.

Case 4: Noisy dataset with pre-trained flows

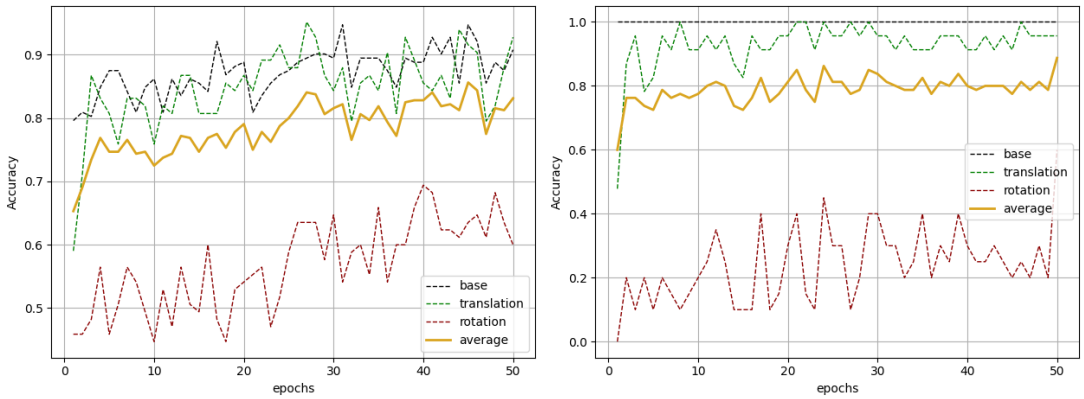
Figures 6.11, 6.12, 6.13, and 6.14 represent graphs for accuracy, precision, recall, and f1-score, respectively. The network consistently demonstrates improvements in all metrics throughout the training process, even after 50 iterations.

When compared to Case 2, where theoretical flows were utilized, the network performs slightly worse in important metrics such as accuracy and f1-score. In the best case, it achieves approximately 0.89 and 0.88, respectively, instead of 0.975 and 0.975.

Similar to Case 2, the model displays a good understanding of the translation class while frequently misclassifying rotation as no transformation (base) based on precision and recall metrics in Figures 6.12 and 6.13. This misclassification is reflected in the f1-score.

Evaluation metrics	Base	Translation	Rotation	Average
Accuracy	1.000	0.957	0.600	0.887
Precision	0.804	1.000	1.000	0.910
Recall	1.000	0.957	0.600	0.887
f1-Score	0.892	0.978	0.750	0.881

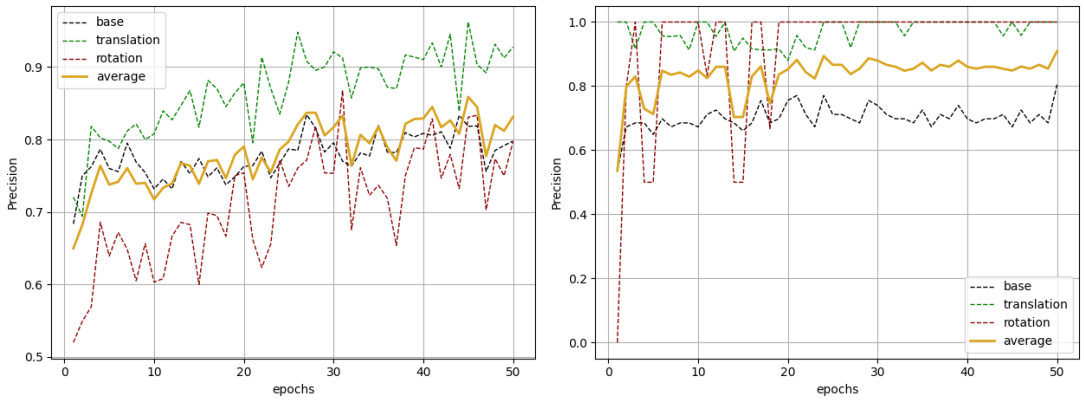
Table 6.6: Case 4 - Classification: Best model after 50 epochs. Obtained at epoch 50/50



(a) Train

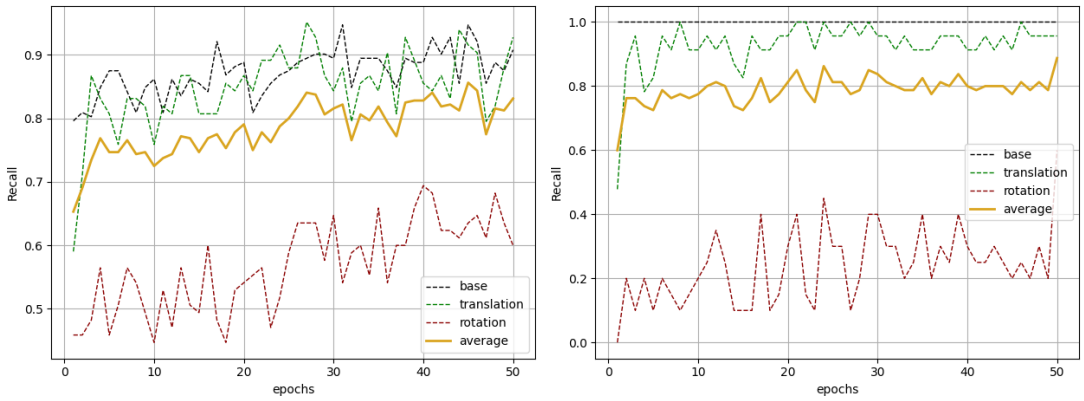
(b) Test

Figure 6.11: Case 4 - Classification: The accuracy metric reaches a value around 0.8-0.89 for both the training and testing sets after 50 epochs. This value is significantly lower than the accuracy observed in Figure 6.7, highlighting the importance of the flow channels in our network.



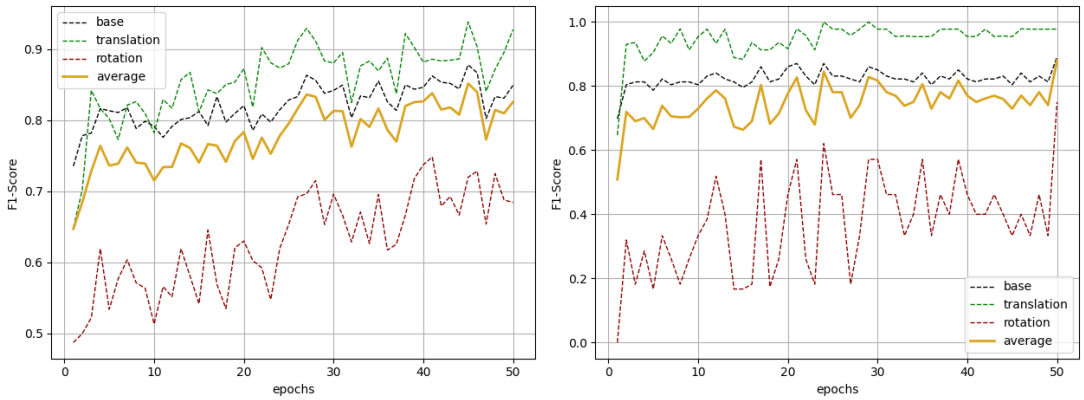
(a) Train (b) Test

Figure 6.12: Case 4 - Classification: The precision metric for rotation is low in the training set but high in the testing set compared to the other two classes, similar to obtained results in Figure 6.8.



(a) Train (b) Test

Figure 6.13: Case 4 - Classification: The recall metric for the base transformation shows a high value, indicating that all actual point clouds where no change was applied were correctly predicted. This observation is consistent with what is shown in Figure 6.9.



(a) Train (b) Test

Figure 6.14: Case 4 - Classification: There is a noticeable drop in performance compared to Figure 6.10 in terms of F1-Score. This drop can be attributed to the utilization of the pre-trained flow in the network.

6.4.3 Segmentation Results

Similarly to the classification part, the PointNet++ segmentation network has been implemented for the four previous cases (section 6.4.2). Specifically, we will focus on Case 2, where theoretical flow is used, and Case 4, where the flow is obtained from the pre-trained FlowNet3D model. The results for Case 1 and Case 3 will be provided in Appendix A.2 for reference.

Case 2: Noisy dataset with theoretical flows

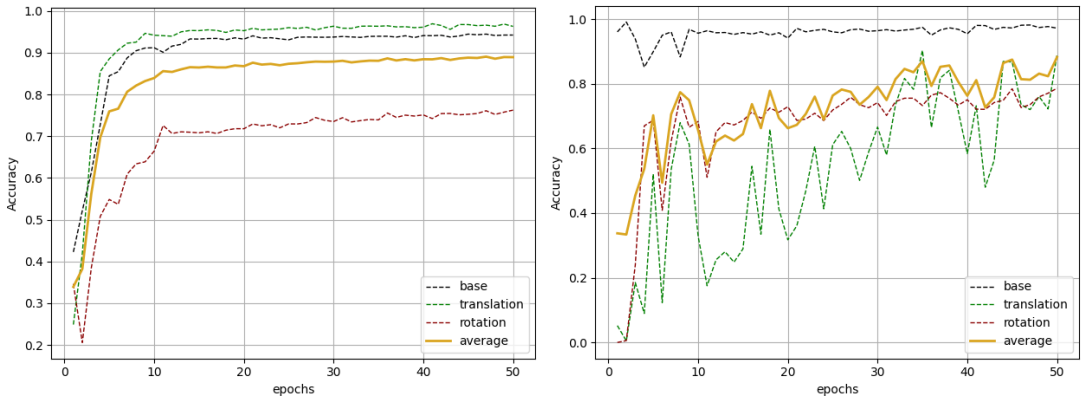
The obtained results for various metrics are presented in Figure 6.15 for accuracy and Figure 6.16 for Intersection over Union. The training process consisted of 50 epochs, during which the method learned from the training set while maintaining better performance than in the test set.

This method peaked at the last iteration displaying an average accuracy of 0.88 and a mean IoU of 0.80, demonstrating the proposed network can reach good results when scene flows are perfectly computed.

Similar conclusions can be drawn, which align closely with those of the classification task. Overall, the rotation transformation proves to be the most challenging for the network, often resulting in failures to accurately identify it. Table 6.7 shows the best IoU value for rotation after 50 epochs remain at 0.67 while those for the base and translation class reach an IoU of 0.85 and 0.89 respectively.

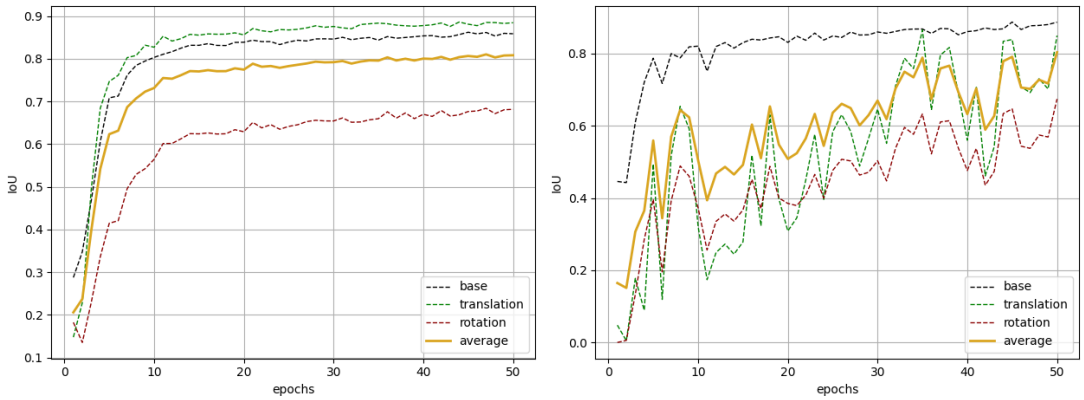
Evaluation metrics	Base	Translation	Rotation	Average
train accuracy	0.942	0.963	0.763	0.889
train IoU	0.858	0.884	0.681	0.808
test accuracy	0.972	0.889	0.786	0.882
test IoU	0.887	0.849	0.674	0.803

Table 6.7: Case 2 - Segmentation: best model after 50 epochs. Obtained at epoch 50/50.



(a) Train (b) Test

Figure 6.15: Case 2 - Segmentation: Accuracy metric



(a) Train (b) Test

Figure 6.16: Case 2 - Segmentation: IoU metric

Case 4: Noisy dataset with pre-trained flows

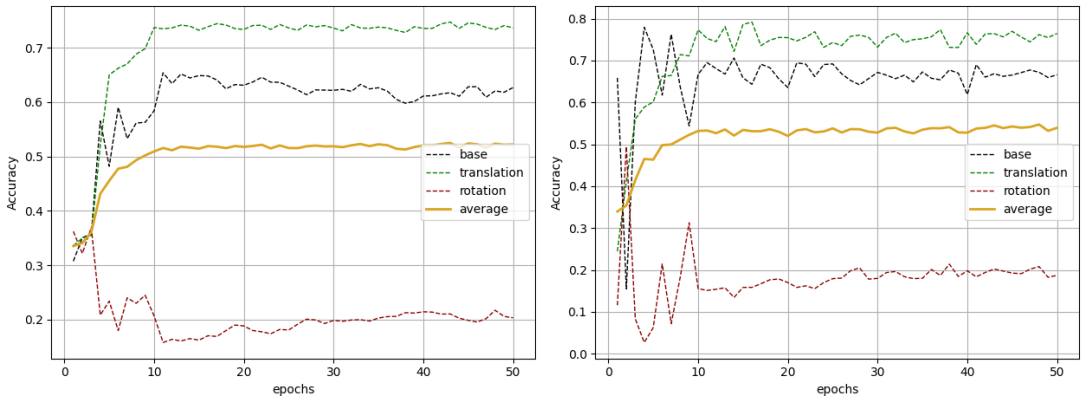
Figures 6.17 and 6.18 present the accuracy and IoU results for the segmentation task using the PointNet++ network. The network displays a slow convergence after 50 iterations and both the training and testing sets show similar performance outcomes.

In Figure 6.17, the accuracy for the translation class is the highest, followed by the base and rotation classes. The accuracy for translation is approximately 0.67, while for rotation, it drops to around 0.16, indicating poor results. The average accuracy converges at around 0.55, which represents a slightly better performance than a random segmentation.

The IoU values in Figure 6.18 converge to approximately 0.35 which is lower than the results obtained in Case 2. These findings demonstrate the limited performance when utilizing the pre-trained scene flow, underlining the importance of flow features as a means to capture distinctions between point clouds.

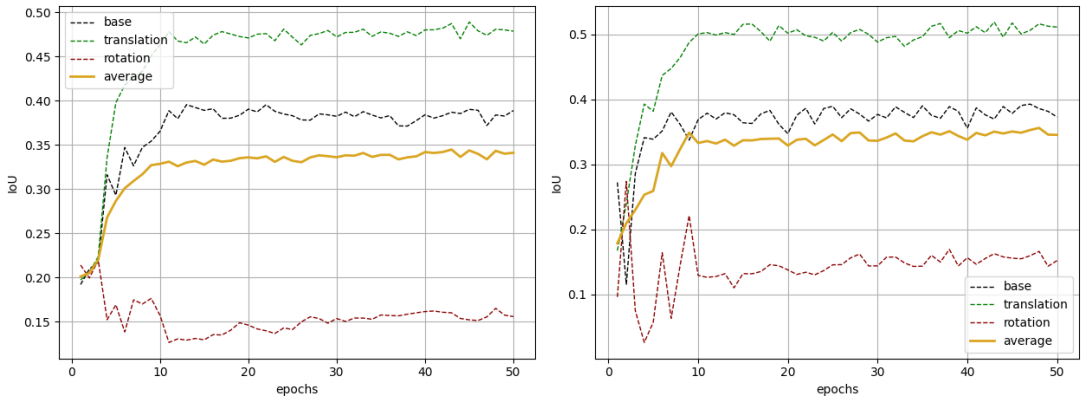
Evaluation metrics	Base	Translation	Rotation	Average
train accuracy	0.620	0.733	0.217	0.524
train IoU	0.384	0.481	0.165	0.343
test accuracy	0.672	0.762	0.208	0.547
test IoU	0.386	0.516	0.166	0.356

Table 6.8: Case 4 - Segmentation: best model after 50 epochs. Obtained at epoch 48/50.



(a) Train (b) Test

Figure 6.17: Case 4 - Segmentation: Accuracy metric



(a) Train (b) Test

Figure 6.18: Case 4 - Segmentation: IoU metric

6.5 Discussion

In this chapter, we described how the dataset for both classification and semantic segmentation is generated. Then, we detailed the implementation of the classical model and the deep learning one. Finally, those models are evaluated on various generated datasets. The obtained results reveal distinct characteristics and limitations of our work worth to be highlighted .

First, the deep learning approach utilizing exact scene flows and PointNet++ demonstrates promising results, surpassing the classical approach in terms of both classification and segmentation tasks. Moreover, this approach offers a more flexible framework when new faults and features will be added. Neural networks for point cloud application is still an active field of research and newer network architectures are outperforming the one employed in this work. Given the favorable outcomes achieved with these primary networks, there is a potential for further improvements in the change detection task by adopting more recent architectures.

Moreover, in both classification and segmentation, the network shows better performance in the second case with the theoretical flows, than in the fourth case,

where pre-trained scene flow is used. This underlines two observations. First, it proves that the network is coherent and can achieve good results when properly trained. Secondly, it emphasizes the significance of scene flows to detect changes and the success of the proposed deep learning is heavily reliant on the performance of FlowNet3D in providing accurate estimates.

The rotation class poses the greatest challenge for the network. This transformation relies heavily on the scene flows of their neighboring points as well as the overall points' features within a plane. This complexity is the main reason explaining why rotation is hard to detect accurately. Particularly, the accuracy drops when noise and imprecise pre-trained flows are used in a complex semantic scene. In case 4 of semantic segmentation, illustrated in Figure 6.17 and 6.18, the algorithm decides to abandon the detection of rotation for translation and base class. A way to solve this issue is to increase the proportion of rotational transform available in the point cloud.

The dataset generated also has its limitation. Despite efforts to model real scenes and simulate diversity through data augmentation, the cuboids used do not fully capture the complexity of real scenes. Indeed, translation and rotation may occur on some parts within a wall rather than affecting the entire structure. Additionally, unpredictable events can occur on construction sites. Those elements are not represented in the artificial dataset and it implies that the performance of the models on the artificial dataset may not be directly translated to real-world applications. A public high-quality and diverse dataset of construction sites containing labeled faults would be beneficial to transfer those results for real application.

The training aspect is also an important factor contributing to the success of a network. Apart from hyperparameters, training the entire network together containing a FlowNet3D followed by a PointNet++ can be challenging due to the large size of these networks and the potential issue of vanishing gradients. To tackle this issue, those 2 networks can be trained independently to make FlowNet3D

predict desired scene flow and PointNet++ classifying or segmenting. Leveraging transfer learning, those two pre-trained models can be combined and trained in a single network, which can potentially improve the results compared to training the networks independently. However, due to time constraints, we were unable to finish this implementation, but it is a possibility for future work and improvement.

6.6 Future Improvements

Several potential avenues for future improvements can be explored to enhance the capabilities of the developed methods:

- Direct or Indirect FlowNet3D Training: To achieve a more comprehensive and integrated approach, a possible future improvement is to train the FlowNet3D network on a personalized dataset. Another possibility is to combine the training of both FlowNet3D and PointNet++ into a single network, the system can leverage the complementary strengths of both models and potentially improve the overall performance and efficiency of the monitoring process.
- Generating a More Diverse and Realistic Dataset: While the developed method has utilized a generated dataset, further improvements can be made by generating an even more diverse and realistic dataset specific to construction sites. By capturing a wider range of construction scenarios, environmental conditions, and potential challenges, the training data can better represent real-world situations. Unfortunately, obtaining labeled real-world data for the motion flow in construction sites would be cumbersome and impractical. Therefore, a self-supervised approach would be more suitable for training the flow network using this expanded dataset, as it leverages the inherent structure and temporal coherence within the data itself, eliminating the need for manual labeling. As FlowNet3D is not designed as a self-supervised network, an alternative self-supervised flow network would be required for this purpose.
- BIM-Oriented Methods: Currently, the method focuses on monitoring signifi-

cant differences between point clouds. Building Information Modeling (BIM) offers a wealth of information that can be utilized to enhance the monitoring process. Exploring BIM-oriented methods, such as integrating BIM data with another type of data like mesh, can provide valuable insights for noise filtering, error detection, and precise fault localization. This integration could enable more efficient and accurate analysis of construction progress and facilitate better decision-making.

- Expanding Class Variations and Providing Detailed Information: To enhance the detection capabilities of the developed method, an extension could involve incorporating additional classes or faults beyond the existing set. By including a broader range of classes, such as various types of structural deformations or different construction elements, the method can better capture and differentiate specific changes in the point clouds. Furthermore, once a change or fault is detected, providing detailed information about the detected class could be beneficial. For instance, this could include additional attributes such as the angle of rotation, the magnitude of translation, or the distance measurement associated with the identified change. By enriching the analysis with these supplementary details, a more comprehensive understanding of the detected changes can be achieved, enabling more precise and informative construction site monitoring.

Chapter 7

Conclusion

Recent improvements in robot autonomy have opened up possibilities for future applications in the construction industry. One potential application is monitoring changes or faults that may occur in construction sites. The purpose of this master thesis is to investigate potential approaches for future application in the construction industry demonstrating a proof of concept.

To begin, a review of the existing literature was conducted to explore relevant methods and frameworks that can serve as our proof of concept. The main frameworks for performing the designated task are presented and explained in this thesis. Two main approaches were identified. The first method is the classical approach which encompasses RANSAC-based algorithms, and geometric properties-based matching. The second approach is a deep learning one, based on PointNet, PointNet++, FlowNet3D.

Both methods were implemented and evaluated using an artificial dataset. Although the generated datasets were simplistic, they provided sufficient richness and diversity for the initial proof of concept and training purposes. Evaluation metrics were introduced for both classification and semantic segmentation tasks, and our two models were assessed accordingly.

In terms of classification, we achieved an average f1-score of 84.8%. The deep learning approach exhibited performance ranging from 88.1% to 97.5%. For semantic segmentation, the average Intersection over Union (IoU) was 71.0%, with the classical and deep learning methods achieving IoU values ranging from 35.6% to 80.3%.

In conclusion, the obtained results indicate the deep learning approach, being a more adaptable framework, can outperform the classical approach in change detection tasks. These findings highlight the potential for further improvements in the promising deep-learning model.

Bibliography

[1] A. Morenville and L. Vermeulen, "Simultaneous localization and mapping and autonomous navigation on SPOT robot from Boston Dynamics," Master's thesis, UCLouvain, Louvain-la-Neuve, 2022.

[2] W. Liu, J. Sun, W. Li, T. Hu, and P. Wang, "Deep learning on point clouds and its application: A survey.," in *Sensor (Basel)*, 2019.

[3] Boston Dynamics, "Spot." `https://www.bostondynamics.com/products/spot`. Accessed: 2023-06-02.

[4] Boston Dynamics, "About spot - spot sdk." `https://dev.bostondynamics.com/docs/concepts/about_spot`. Accessed: 2023-06-02.

[5] Boston Dynamics, "Geometry and frames - spot sdk." `https://dev.bostondynamics.com/docs/concepts/geometry_and_frames`. Accessed: 2023-06-02.

[6] Y.-B. Jia, "Lecture notes in foundations of robotics and computer vision (cs 477/577)," 2022.

[7] Boston Dynamics, "Autonomy technical summary - spot sdk." `https://dev.bostondynamics.com/docs/concepts/autonomy/graphnav_tech_summary`. Accessed: 2023-06-02.

[8] N. Mehendale and S. Neoge, "Review on lidar technology," in *SSRN*, 2020.

[9] Velodyne Lidar, "Puck products - velodyne lidar." `https://velodynelidar.com/products/puck`. Accessed: 2023-06-02.

[10] M. V. Okunsky and N. V. Nesterova, "Velodyne lidar method for sensor data decoding," in *IOP Publishing*, 2019.

[11] Velodyne Lidar, "Vlp-16 user manual." `https://velodynelidar.com/wp-content/uploads/2019/12/63-9243-Rev-E-VLP-16-User-Manual.pdf`. Accessed: 2023-06-02.

[12] X. Huang, G. Mei, J. Zhang, and R. Abbas, "A comprehensive survey on point cloud registration," *CoRR*, vol. abs/2103.02690, 2021.

[13] C.-S. Chen, Y.-P. Hung, and J.-B. Cheng, "Ransac-based darces: a new approach to fast automatic registration of partially overlapping range images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 11, pp. 1229–1234, 1999.

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

[15] S. Choi, Q.-Y. Zhou, and V. Koltun, "Robust reconstruction of indoor scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[16] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[17] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[18] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration.," *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[19] C. Stachniss, "Point-to-plane and generalized icp - 5 minutes with cyrill." `https://www.youtube.com/watch?v=2hC9IG6MFD0`. Accessed: 2023-06-01.

[20] Z. Zhang, *Iterative Closest Point (ICP)*, pp. 433–434. Boston, MA: Springer US, 2014.

[21] P. V. Hough, "Method and means for recognizing complex patterns," Dec. 18 1962. US Patent 3,069,654.

[22] D. Borrmann, J. Elseberg, K. Lingemann, and A. Nüchter, "The 3d hough transform for plane detection in point clouds: A review and a new accumulator design," *3D Research*, vol. 2, no. 2, pp. 1–13, 2011.

[23] N. Kiryati, Y. Eldar, and A. M. Bruckstein, "A probabilistic hough transform," *Pattern recognition*, vol. 24, no. 4, pp. 303–316, 1991.

[24] A. Yla-Jaaski and N. Kiryati, "Adaptive termination of voting in the probabilistic circular hough transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 9, pp. 911–915, 1994.

[25] M. Y. Yang and W. Förstner, "Plane detection in point cloud data," in *Proceedings of the 2nd int conf on machine control guidance, Bonn*, vol. 1, pp. 95–104, 2010.

[26] J. Bao, X. Yuan, G. Huang, and C.-T. Lam, "Point cloud plane segmentation-based robust image matching for camera pose estimation," *Remote Sensing*, vol. 15, no. 2, p. 497, 2023.

[27] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, pp. 1–1, 06 2020.

[28] D. Pierre, "Linfo2262 machine learning :classification and evaluation," January 2022.

[29] A. Kumar, "Sklearn neural network example – mlpregressor." `https://vitalflux.com/sklearn-regression-example-mlpregressor/`. Accessed: 2023-05-23.

[30] "Backpropagation." `https://en.wikipedia.org/wiki/Backpropagation`. Accessed: 2023-05-23.

[31] C. R. Qi, H. Su, K. Mo, and L. 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*, pp. 652–660, 2017.

[32] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

[33] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, "3d semantic parsing of large-scale indoor spaces," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1534–1543, 2016.

[34] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "Randla-net: Efficient semantic segmentation of large-scale point clouds," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11108–11117, 2020.

[35] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[36] X. Liu, C. R. Qi, and L. J. Guibas, "Flownet3d: Learning scene flow in 3d point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 529–537, 2019.

[37] MaxChanger, "Awesome-point-cloud-scene-flow." `https://github.com/MaxChanger/awesome-point-cloud-scene-flow`. Accessed: 2023-05-23.

[38] M. Hossin and M. N. Sulaiman, "A review on evaluation metrics for data classification evaluations," *International journal of data mining & knowledge management process*, vol. 5, no. 2, p. 1, 2015.

[39] A. A. Taha and A. Hanbury, "Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool," *BMC medical imaging*, vol. 15, no. 1, pp. 1–28, 2015.

[40] hyangwinter, "flownet3d_pytorch." https://github.com/hyangwinter/flownet3d_pytorch/tree/master. Accessed: 2023-05-23.

[41] yanx27, "Pointnet_pointnet2_pytorch." https://github.com/yanx27/Pointnet_Pointnet2_pytorch. Accessed: 2023-05-23.

[42] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4040–4048, 2016.

Appendix A

Additional Results

A.1 Classical Model

A.1.1 Classification Result

Evaluation metrics	Base	Translation	Rotation	Average
Accuracy	0.812	1	0.768	0.853
Precision	0.910	0.718	1	0.879
Recall	0.812	1	0.768	0.853
f1-Score	0.858	0.836	0.869	0.855

Table A.1: Classification results with the Classical model

A.1.2 Segmentation Result

	Predicted base	Predicted translation	Predicted rotation
Actual base	0.812	0.188	0
Actual translation	0	1	0
Actual rotation	0.146	0.085	0.768

Table A.2: Segmentation results with the Classical model

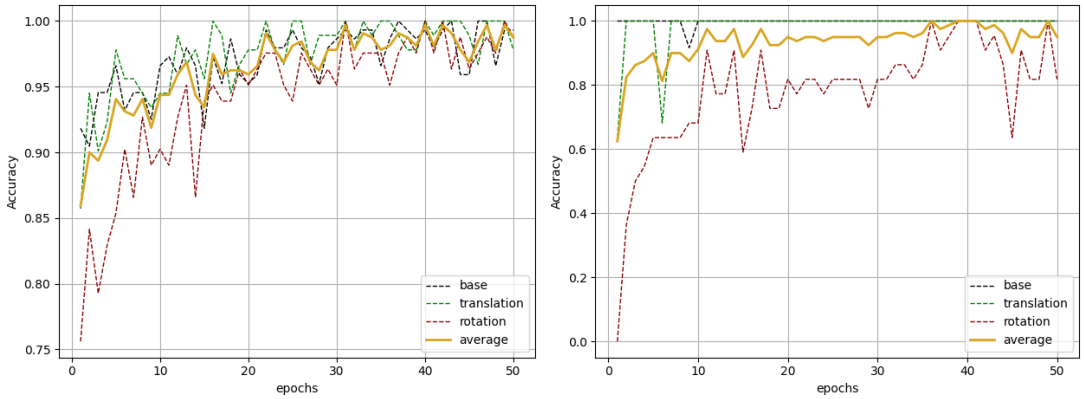
A.2 Deep Learning Model

A.2.1 Classification Results

Case 1: Point cloud data without noise and theoretical flows

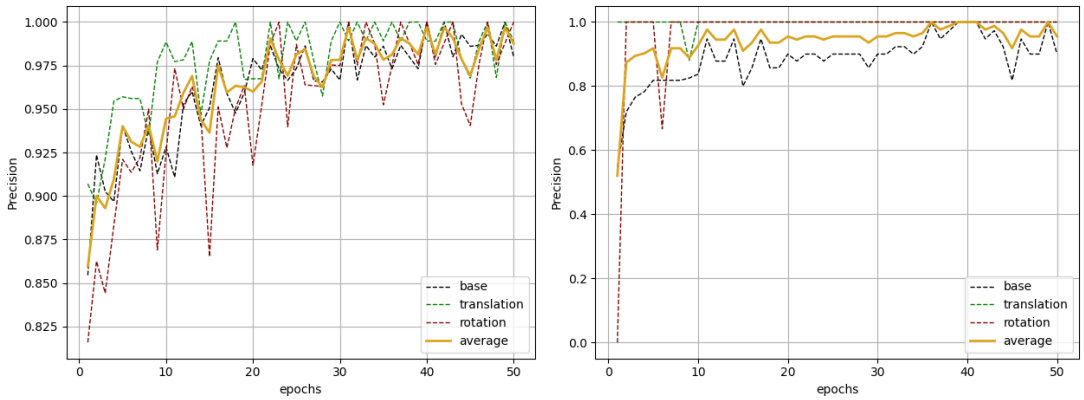
Evaluation metrics	Base	Translation	Rotation	Average
Accuracy	1.000	1.000	1.000	1.000
Precision	1.000	1.000	1.000	1.000
Recall	1.000	1.000	1.000	1.000
f1-Score	1.000	1.000	1.000	1.000

Table A.3: Case 1 - Classification: Best model after 50 epochs. Obtained at epoch 36/50.



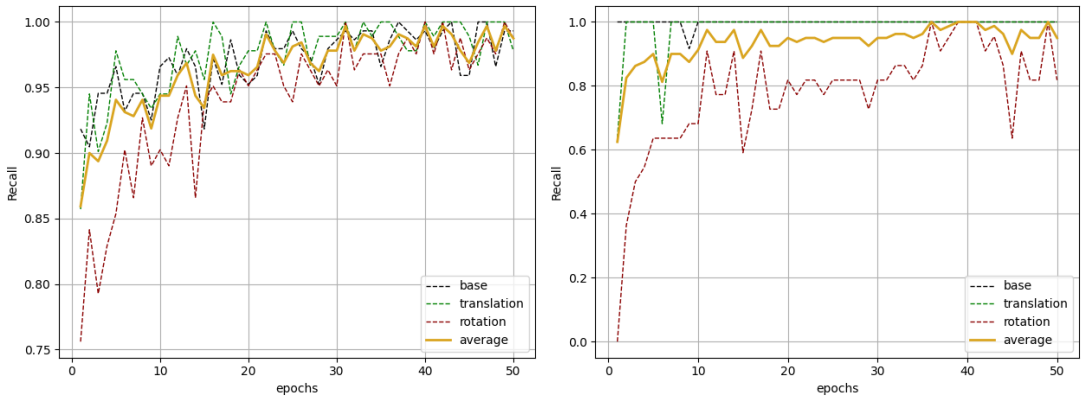
(a) Train (b) Test

Figure A.1: Case 1 - Classification: Accuracy metric.



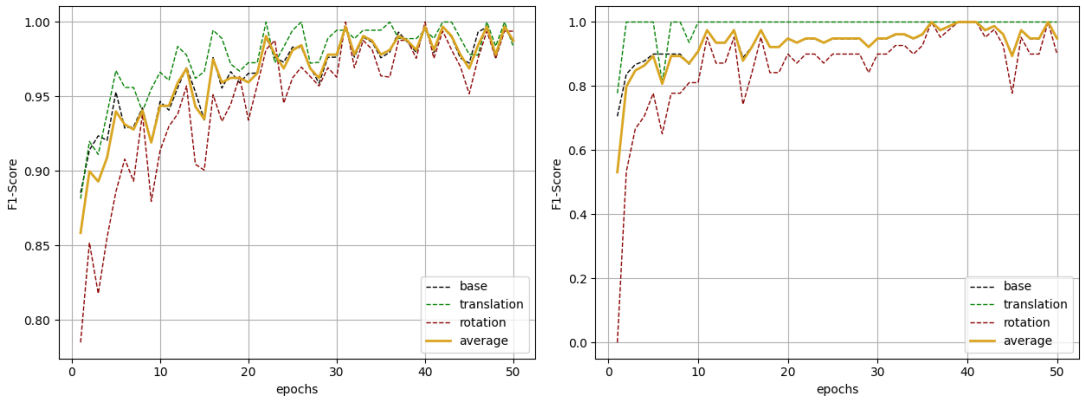
(a) Train (b) Test

Figure A.2: Case 1 - Classification: Precision metric.



(a) Train (b) Test

Figure A.3: Case 1 - Classification: recall metric.



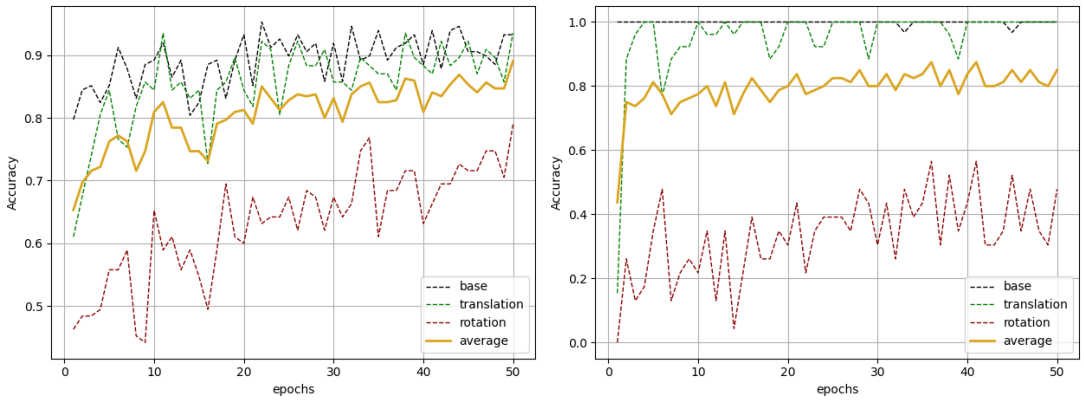
(a) Train (b) Test

Figure A.4: Case 1 - Classification: F1-score metric.

Case 3: Point cloud data without noise and pre-trained flows

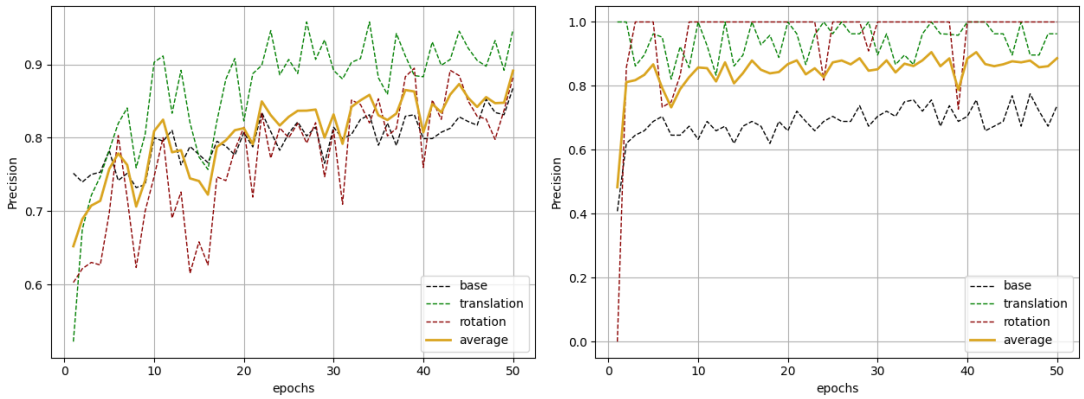
Evaluation metrics	Base	Translation	Rotation	Average
Accuracy	1.000	1.000	0.565	0.875
Precision	0.756	1.000	1.000	0.905
Recall	1.000	1.000	0.565	0.875
f1-Score	0.861	1.000	0.722	0.866

Table A.4: Case 3 - Classification: Best model after 50 epochs. Obtained at epoch 36/50.



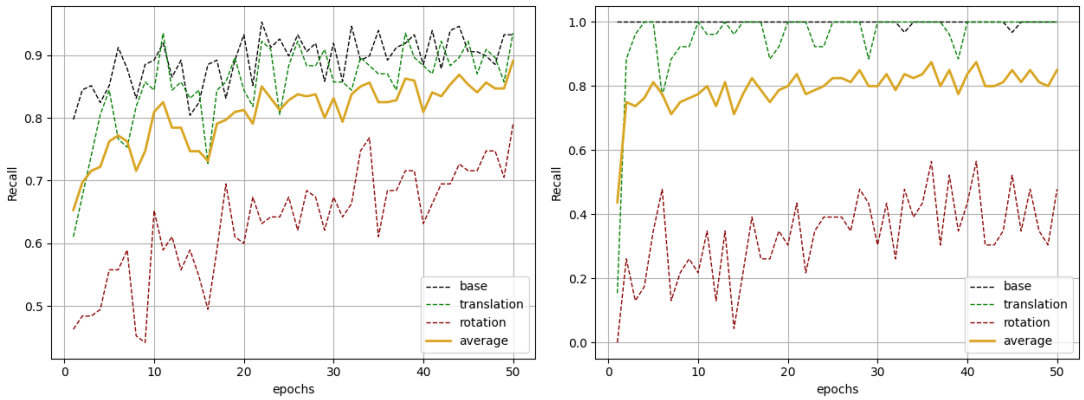
(a) Train (b) Test

Figure A.5: Case 3 - Classification: Accuracy metric.



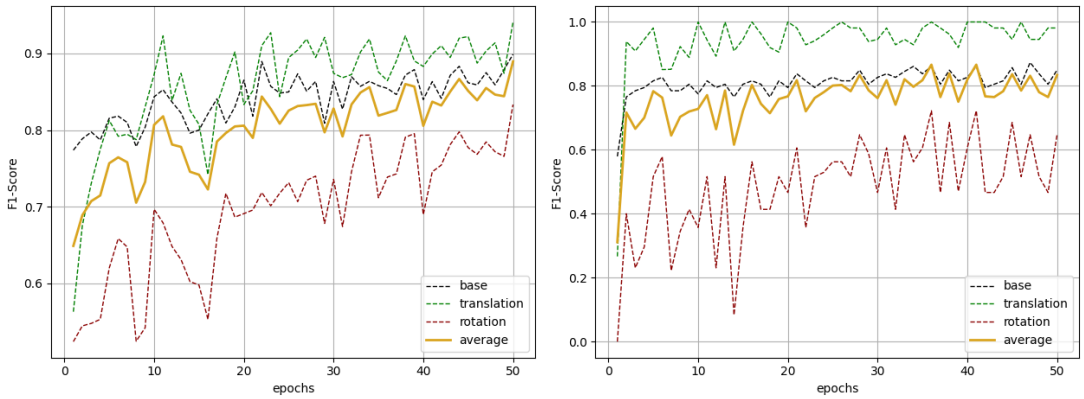
(a) Train (b) Test

Figure A.6: Case 3 - Classification: Precision metric.



(a) Train (b) Test

Figure A.7: Case 3 - Classification: Recall metric



(a) Train (b) Test

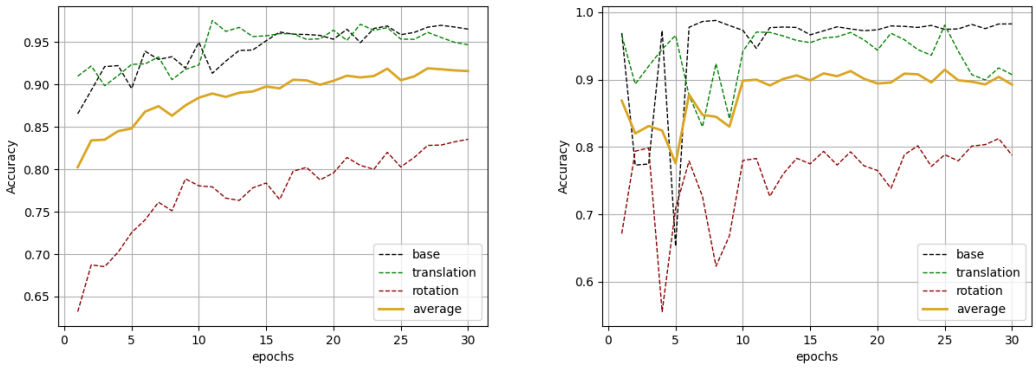
Figure A.8: Case 3 - Classification: F1-score metric

A.2.2 Segmentation Results

Case 1: Point cloud data without noise and theoretical flows

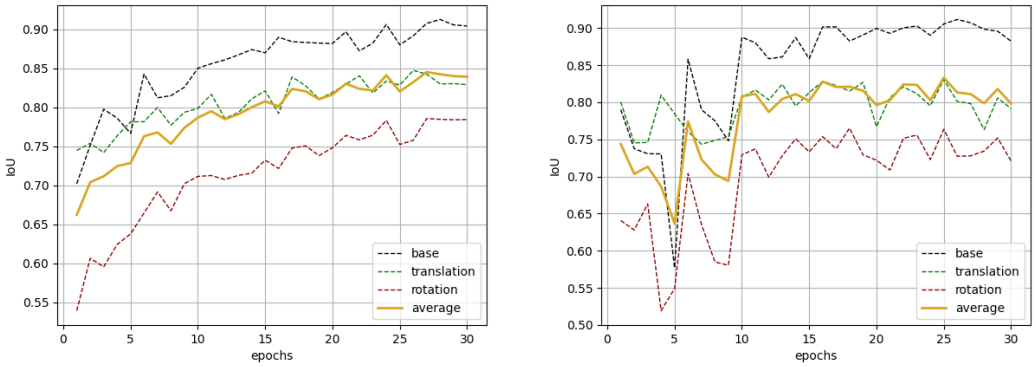
Evaluation metrics	Base	Translation	Rotation	Average
train accuracy	0.967	0.965	0.796	0.910
train IoU	0.927	0.865	0.728	0.840
test accuracy	0.976	0.952	0.884	0.937
test IoU	0.951	0.894	0.809	0.884

Table A.5: Case 1 - Segmentation: Best model after 50 epochs. Obtained at epoch 47/50.



(a) Train (b) Test

Figure A.9: Case 1 - Segmentation: Accuracy metric.



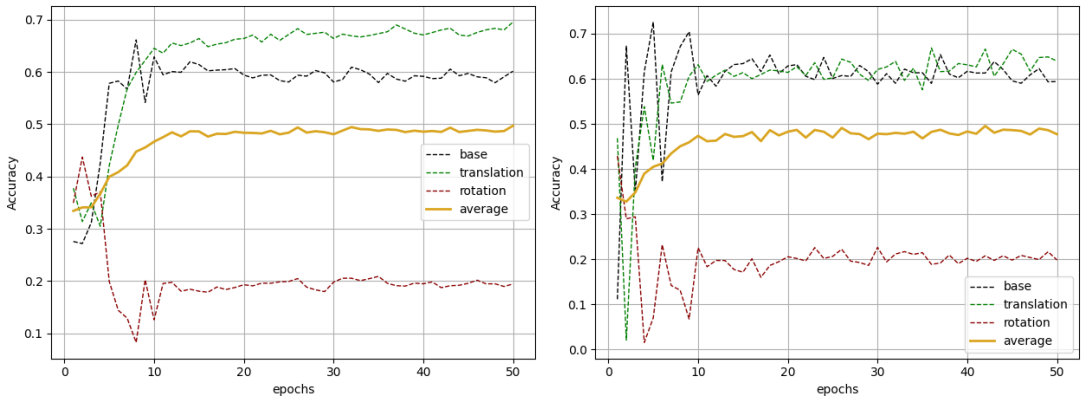
(a) Train (b) Test

Figure A.10: Case 1 - Segmentation: IoU metric.

Case 3: Point cloud data without noise and pretrained flows

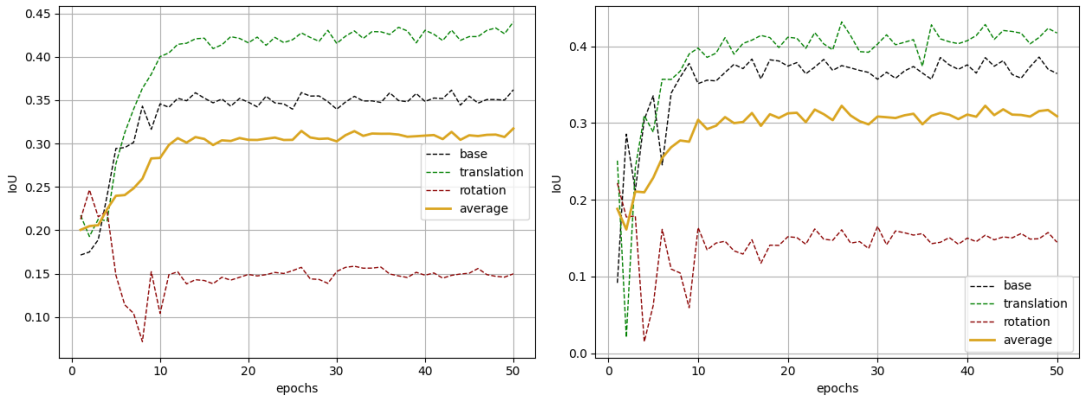
Evaluation metrics	Base	Translation	Rotation	Average
train accuracy	0.588	0.680	0.187	0.485
train IoU	0.352	0.419	0.145	0.305
test accuracy	0.613	0.666	0.208	0.496
test IoU	0.385	0.429	0.154	0.323

Table A.6: Case 3 - Segmentation: Best model after 50 epochs. Obtained at epoch 42/50.



(a) Train (b) Test

Figure A.11: Case 3 - Segmentation: Accuracy metric.



(a) Train (b) Test

Figure A.12: Case 3 - Segmentation: IoU metric.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**