Tartu University

Faculty of Science and Technology

Institute of Technology

Lucas Johannes Adriaan Marais

Improving Railway 3D Point Cloud Segmentation

Master's thesis (30 EAP)
Robotics and Computer Engineering

Supervisors:

Ludovic Chuet
Veiko Vunder

Tartu 2026

Resümee/Abstract

Raudtee 3D-punktipilve segmentatsiooni parandamine

Käesoleva lõputöö eesmärk on parandada praegust punktipilve segmentatsiooni kontaktvõrgu (catenary) renoveerimis- ja hooldusprotsessides. Praegune segmentatsioon on piiratud tuvastatavate klasside arvu poolest, mis omakorda piirab töötajate jaoks automatiseerimise võimalusi.

Segmentatsiooni parandamiseks viiakse läbi andmestiku täiustamine ja mudeli ümberhindamine. Andmestik klassifitseeritakse ümber, et lisada rohkem klasse, ning seda täiendatakse Blenderi abil genereeritud sünteetiliste andmetega.

Praegust PTv3 mudelit võrreldakse 3D-UMamba mudeliga, mis on hiljutine mudel ja on näidanud suurepäraseid tulemusi LiDAR-andmestike puhul. Teostatakse hüperparameetrite häälestamine ning treenimissessioonid logitakse parema visualiseerimise ja võrdluse eesmärgil.

Sünteetiliste andmete genereerimine viiakse läbi Blenderi abil, kasutades olemasolevaid kontaktvõrgu elementide BIM-mudeleid täpsuse tagamiseks. Luuakse erinevad stsenaariumid, et katta võimalikult palju juhtumeid. Lisaks kasutatakse C++/pybind11 Embree-põhist kiirtejälituse skannerit, mis genereerib märgendatud punktipilvi koos primitiivipõhise semantikaga, et simuleerida LiDAR-andmete kogumist.

Sünteetilisi näidiseid kasutatakse reaalse maailma andmestiku täiendamiseks ja klasside ebatasakaalu vähendamiseks.

Tulemused näitavad, et sünteetiline andmete täiendamine parandab alaesindatud klasside tuvastamist. 3D-UMamba mudel ületab PTv3 mudelit täpsuse poolest, kuid toob kaasa kerge täitmisaja pikenemise. Lõplik mudel vajab enne tootmiskeskkonda kasutuselevõttu siiski edasist häälestamist ja hindamist.

Edasine töö hõlmab rohkemate reaalse maailma andmete lisamist treeningandmestikku, täiendavat hüperparameetrite häälestamist ning järeltöötlustoru rakendamist, et eraldada töötajatele kasulikku teavet ja leevendada nende tüütuid ja korduvaid ülesandeid. Täiendavaks eesmärgiks on maksimaalse võimaliku andmemahu kogumine, nende eraldamine ja tõhus kasutamine kontrollitud töövoos kõigi projekteerimisuuringute jaoks.

CERCS: T120 Süsteemitehnika, arvutitehnika; T125 Automatiseerimine, robootika, juhtimistehnika; T220 Ehitustehnika; T290 Raudteetranspordi tehnoloogia

Märksõnad: raudtee, punktipilv, LiDAR, 3D punktipilve segmentatsioon, semantiline segmentatsioon, kontaktvõrgu infrastruktuur, andmestiku täiendamine, sünteetiliste andmete gene-

reerimine, Blender, LiDAR-simulaator, 3D-UMamba, Point Transformer, hüperparameetrite häälestamine, Ray Tune, MLflow, BIM

Improving Railway 3D Roint Cloud Segmentation

This thesis aims to improve the current point cloud segmentation in the catenaries overhaul and maintenance processes. The current segmentation is limited in the number of classes it can detect, which limits the automation possibilities for the workers.

To improve segmentation, enhancements on the dataset and a reevaluation of the model will be done. The dataset will be reclassified to add more classes and augmented with synthetic data generated with Blender.
The current model PTv3 model will be compared to 3D-UMamba, a recent model showing great results on LiDAR datasets. Hyperparameter tuning will be done and the training sessions will be logged for better visualization and comparison.

Synthetic data generation will be made using Blender, with existing BIM models for the catenary elements to ensure accuracy. Different scenarii will be created to cover as much cases as possible. And a C++/pybind11 Embree-backed raycasting scanner that produces labeled point clouds with per-primitive semantics will be used to simulate the LIDAR acquisition.

Synthetic samples are used to augment a real-world dataset and reduce class imbalance.

Results show that the synthetic augmentation improves detection of under-represented classes. The 3D-UMamba model outperforms PTv3 in accuracy, with a slight increase in execution time. The final model still needs to be further tuned and evaluated before deployment to production.

Future work includes adding more real world data to the training set, further hyperparameter tuning, and implementing the after processing pipeline in order to extract useful information for the workers and ease out their tidious and repetitive tasks. Further objectives are to retrieve the maximum data available, extract and use them efficiently on a controled pipeline for all the design studies.

CERCS: T120 Systems engineering, computer technology; T125 Automation, robotics, control engineering; T220 Civil engineering; T290 Railway transport technology

Keywords: railway, point cloud, LiDAR, 3D point cloud segmentation, semantic segmentation, catenary infrastructure, dataset augmentation, synthetic data generation, Blender, LIDAR, simulator, 3D-UMamba, Point Transformer, hyperparameter tuning, Ray Tune, MLflow, BIM

Contents

List of Figures

List of Tables

Abbreviations, Constants, Generic Terms

PTv3 - Point Transformer version 3

TGA - Token Grouping and Aggregating

VTS - Voxel-based Token Serialization

SDS - Semantic feature-based Dynamic Sampling

SDC - Semantic feature-based Dynamic Clustering

LFA - Local Feature Aggregation

GLF - Global Feature Learning

IoU - Intersection over Union

mIoU - mean IoU

LIDAR - LIght Detection And Ranging, here is the scanning device to retrieve the 3D point clouds.

lr - Learning rate

FPS - Farthest Point Sampling

PP - Messenger wire

PA - Auxiliary messenger wire

FC - Contact wire

CdPA - Aerial protection wire

DB - Data Base

1 Introduction

In the overhaul and maintenance of catenaries, the use of point clouds is becoming predominant to refer to the existing installations. They are currently used as a reference point to determine the positions of components from existing material, measure various distances and clearances but above all of this, to have existing data available without going on cite.

In order to redraw existing installations, the worker had to manually extract cuts from the point cloud and flatten the resulting point cloud to have a base to work on. Currently, with machine learning, this extraction is almost completely automated, which significantly reduces the processing time for the worker. However, as for now, the method can be further pushed and improved.

Indeed, workers still have to manually measure everything to reconstitute the elements. And by improving the point cloud segmentation, it would be possible not only to find poles or basic elements but also smaller elements that can lead to direct and precise placement of dimension lines and elements.

To further improve calculation and study time, specific terrain parts like cliffs, bridges, funnels, embankments can help the worker to better understand the terrain and adapt the installation accordingly.

1.1 Problem Statement

In order to further improve the current process, the model has to be trained on more labels (classes). But the current dataset is small, not representative of many cases, and does not contain everything that needs to be detected in sufficient quantities for training.

There is then a need to reclassify the current dataset and add more data for it to learn better without overfitting. All that with enough accuracy for it to be reliable. It needs to be added the fact that the model should work on a regular workstation and not cloud based for both performance and material limitations.

From that, it will be possible to further assist the workers and having their performances improved, reduce the number of errors and concentrate on more important topics and therefor are relevant for the point cloud segmentation.

1.2 Objectives and Roadmap

This thesis aims at improving the current machine learning implementation for it to be segmented into more classes, without loosing too much in accuracy.

The objectives are then to choose the best existing model, increase the dataset and tune the model for it to be efficient and accurate.

For this, multiple actions can be determined. The first is to acknowledge and research what has been done as the state of the art in this field.
The second one is that due to the lack of complete dataset, there is a need to augment it, for the model to better understand the cases. For this we have multiple choices, the first would be to improve the current one, train a model on it and then reclassify other point clouds and manually clean them. Or, another option would be to create a simulated dataset in order to have a much broader set of examples to give to the model.
A third method can be used in addition of the two previous ones, to tune the model to obtain the best performances in accuracy but also in execution time and processing efficiency.

2 State of the Art

Various models were already developed and put in competition for point cloud classification and segmentation. But also various data augmentation methods were studied to improve the model learning and results.

2.1 Theoretical solutions.

Multiple models already exist for point cloud segmentation and classification. This allows us to extract the best principles and architectures to use or build a model that would fit our needs.
The most common architectures are PointNet [1], Point-BERT [2], Point-MAE [3], DGCNN [4], PointCNN [5], PointMamba [6] and some others. Various models are based on these architectures with some modifications to improve either the accuracy or the processing time. These models are then compared on various datasets such as S3DIS [7], ScanNet [8], SemanticKITTI [9], DALES [10] and others.

The models that stand out the most are 3D-UMamba [11, 12], PTv3 [13] and DCTNet [14]. These three models have shown very good results in terms of accuracy and processing time. They are based on encoder-decoder U-net architectures.

3D-UMamba [11, 12] is based on PointMamba [15]. The architecture is based on three main components, TGA, VTS, and Mamba.
PTv3 [13] is based on point cloud serialization and serialized attention.
DCTNet [14] is based on LFA with SDS and SDC for downsampling and on Transformer-based GFL blocks. It aims at reducing computational costs while ensuring homogeneity in local neighborhoods.
The two first model are focused on enhancing global understanding while DCTNet aims a faster execution time while keeping enough understanding of the point's neighborhood.

Table 2.1: Performance comparison on the airborne MS-LiDAR dataset.

Model	Input Points	F1 (%)	mIoU (%)	OA (%)	Latency (ms)
PointNet++ [16]	4096	72.1	58.6	90.1	322.6
DCTNet [14]	4096	86.0	80.2	95.0	**23.3**
3D-UMamba [11]	4096	**90.8**	**84.5**	**95.9**	66.1

Table **2.1** from [11, 17, 18] shows the performances of DCTNet and 3D-UMamba on the airborne MS-LiDAR dataset. It can be seen that 3D-UMamba outperforms DCTNet in all metrics

excepted the latency, with a significant margin in mIoU and F1 score.

Table 2.2: Performance comparison on the airborne DALES dataset.

Method	Input points	OA (%)	mIoU (%)	Latency (ms)
PointNet++ [16]	8192	95.7	68.3	487.1
PointTransformerV3 [13]	8192	96.9	77.4	**41.9**
3D-UMamba [11]	8192	**97.2**	**78.1**	79.2

Table **2.2** from [10, 11] shows the performances of PTv3 and 3D-UMamba on the airborne DALES dataset. It can be seen that 3D-UMamba outperforms PTv3 in OA and mIoU, while PTv3 has a lower latency.

Due to no open sourced code and weak precision compared to 3D-UMamba, the DCTNet model will not be tested and implemented on our pipeline. Even though it has great results overall and really low execution time.

2.2 Theoretical enhancements.

In addition to the impact of the model on accuracy, there is also the training that impact a lot the results. This can be separated in two main categories, which are the quality of the dataset used for training as well as the fine tuning of the training parameters.

For the first part, there is currently two main dataset about railway, WHU-Railway3D [19] and Rail3D [20]. These dataset do not correspond exactly to our needs but can be reclassified or some parts can be extracted. This allows us to train on a more balanced dataset.
Reducing class imbalance will help improve the results of the model but also allow a more stable training with less risks of overfitting.
Another way of improving the dataset would be to generate data that we don't have or that would be too long to hand classify close enough to the real data to train the model on. There are some LIDAR simulations made like Blensor [21] and Blainder [22], two Blender [23] LIDAR add-on that alows the simulation of a LIDAR within the Blender scene.

In order to get the most out of the model, various existing tools can be used in order to visualize and improve both training speed and accuracy as well as automatize the fine tunning.
Tools like MLFlow [24] and Tensorflow [25] can help logging and visualizing the training data and help detect problems within the training. It can compare training and validation data for multiple runs and different models. Other tools like Ray Tune [26] and Hydra [27] may help on the tuning of the hyperparameters.

3 Methodology

3.1 Research Methods

3.1.1 Dataset augmentation

To augment the dataset, two main methods will be used. The first one is to reclassify existing dataset. The previous dataset used already contained the main classes. The reclassification will then focus on adding the missing classes. This will be done manually due to the small size of the dataset.

The second method will be to create simulated data using Blender. This will allow us to balance the dataset to have more representative data for the model to learn from. Various scenarii will be created to cover as much cases as possible.

The different options of Blender add-on will be tested in order to get the best result possible while being in line with the rest of the database modeling.
The goal will then be to create the scene, with the necessary elements. The necessary elements will be first defined by the available elements available on site. There are existing Inventor 3D models of the different elements that can be imported into Blender.
These elements are from BIM projects and are as close as possible to the real elements. A railway with a ballast will first constitute the scene where the supports will then be imported and the different catenaries modeled on them.
This method using Blender will allow us to create multiple scenarii with supports for one track or more and gantries. Another benefit of using Blender is that we can use its rendering engine to simulate the LIDAR acquisition and retrieve the name of the different elements for the points to be classified automatically and precisely
Blender will also allow us to simulate different types of LIDAR with different characteristics (number of points, noise, range, etc..). This will help diversify the dataset and make the model more robust.

If the model is trained enough from the current and simulated dataset with new classes and gives satisfying results, the model could be used in order to segment new point clouds. It will then be faster to correct manually the pre-segmented point cloud to enlarge even more the dataset, making the training even self-sufficient.

3.1.2 Model enhancements and choosing

The current model used in production is PTv3 [13]. This model allows a great understanding of local and global features. But as seen previously, 3D-UMamba [11] is giving better results with little losses in execution time.

The two models will be compared to evaluate if 3D-UMamba performs indeed better for our use case and if the execution time is not too much impacted. Indeed, the workers have to process quite big point clouds representing multiple kilometers of railways. And the impact of time processing for these huge files can expand the worker unproductive time. A line must then be drawn between speed and accuracy depending on the needs of the workers and the output performance of the models.

In order to evaluate the model performances, running multiple and independent training sessions would be quite tidious. To make the process faster, Ray Tune [26] will be used to find the best hyperparameters for the models. This script will allow to run and evaluate automatically different parameters from a search space.
To better visualize the results and compare the different models, MLFlow [24] will be used to log the different training sessions and visualize them. This will allow us to compare the different models and their hyperparameters in order to choose the best one for our use case. It will also help us detect problems like underfitting or overfitting during the training and fix them. The aim of using MLFlow to log everything is to be able to compare models within or between their trainings.
Finally, once the model will be chosen, Hydra [27] will be used to manage the configuration of the training and testing scripts. This will allow us to easily change the parameters and configurations of the model without having to modify the code itself. This will make the process more efficient.

3.2 Research Tools

3.2.1 Software

The main software that will be used for this thesis is Python with conda and pip to manage the installation and use of the necessary libraries within virtual environments. The main libraries that will be used are PyTorch [28], Tensorflow [25], MLFlow [24], Ray Tune [26] and Hydra [27].
Blender [23] will be used for the simulation of the LIDAR data. The add-ons Blensor [21] and Blainder [22] will be tested to see which one gives the best results for our use case. Inventor and REVIT from Autodesk will be used as source for the 3D models from the BIM

3.2.2 Hardware

The main hardware that will be used for this thesis is a portable workstation with a NVIDIA RTX 3000. The GPU will be used for the training and testing of the models as well as for the data augmentation, visualisation and synthetic data generation. Another portable workstation

with a NVIDIA RTX 4090 will be used for the training of the production model, allowing larger batch.

3.3 Research Object

The research object of this thesis is the improvement of the current point cloud segmentation model used in the overhaul and maintenance of catenaries. The goal is to improve the model to be able to segment more classes with a good accuracy and execution time.

The current model is PTv3 [13], which will be compared to 3D-UMamba [11]. The dataset used for the training will be augmented with reclassified existing data as well as simulated data created with Blender [23]. If the model performs well enough, it could be used to pre-segment new point clouds to further enlarge the dataset with less manual work.

The final goal is to have a model that can be used in production for finding the catenary supports. This would help in the future to automate the process of creating the plans for the overhaul and maintenance of catenaries, by extracting the positions of the important elements directly from the point clouds. This should enable a significant reduction of the time spent on these tasks and allow workers to focus on more important tasks. But should also open to automatic checking of the installations, automatic finding of the important factors in the correct placement of the catenary parts.
Having the model performing well enough to detect more classes would allow to directly place dimension lines and elements in the drawings, further reducing the time spent on these tasks. For evaluation line, mean and per class IoU will be used as main metrics.

3.4 Research Materials

The main material that will be used for this thesis is the existing point cloud dataset used for the current model training, made by the company. This dataset will be reclassified to add more classes and make it more representative of the real cases.
Another dataset will be used to augment the training data and reduce class imbalance, generated with Blender [23].
All assets used for the simulation will be sourced from existing BIM models to ensure accuracy and realism. They will be extracted from Autodesk Inventor and REVIT files of real catenary projects.

To train PTv3 [13] and 3D-UMamba [11], the open sourced code from their respective GitHub repositories will be used and modified to fit our use case and dataset and the codes from the current model will be used as baseline and starting approach.

4 The Results

4.1 Artificial dataset generation

4.1.1 Scene generation

For supports and gantries, the first attempt was to generate them randomly but with controlled parameters. But due to lack of coherent result and lack of time, the use of real assemblies from a project was finally set. Even though it has some limitations for placing correctly the support structures above the tracks.

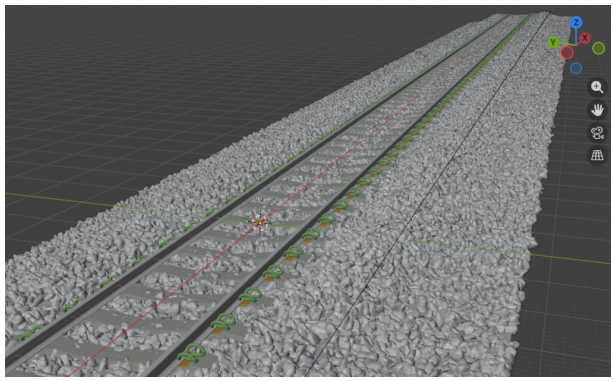

Figure 4.1: Blender view of the generated railway, without catenary equipments

The scene generation on Blender [23] can be separated in two main components. The tracks and ballast assembly is made out of the geometry nodes system. This will allow to create a procedural track that can be modified easily. The ballast is made out of a particle system that will scatter small rocks on the ground. The rails and sleepers are then placed along a curve that can be modified to create straight or curved tracks.
The complete ballast and track assembly is modified using the parameters in the table **4.1**.

Table 4.1: Track geometry and structural parameters

Parameter	Value (mm)	Description
Start point	(x, y, z)	Coordinates defining the starting location where the track passes
Mid point	(x, y, z)	Intermediate control point defining curvature and elevation changes
End point	(x, y, z)	Coordinates defining the final location where the track passes
Gauge	float	Distance between the inner faces of the two rails
Sleeper spacing	float	Longitudinal distance between consecutive sleepers
Ballast height	float	Vertical thickness of the ballast layer beneath the sleepers
Number of tracks	float	Total number of parallel tracks along the alignment
Embankment	float	Raised earth structure supporting the track and ballast

This allows to create different track configurations to avoid overfitting the model. The resulting track is then rendered in the scene as shown in Fig. **4.1**.

An internal script in python, imported, allows multiple tasks. It first allows to modify automatically the track configuration by influencing directly on the parameters instead of doing it manually. And then it allows to import different external scripts in order to manage the automated dataset creation.

Regarding the supports positioning, they were first sorted by the number of tracks they cover (from 1 to 5 including gantries) and separated to have supports on one side and gantries on the other. This allows the creation of multiple fake scenarii, e.g. two tracks scenario with one support for two track or two supports covering one track. The result can be seen on Fig. **4.2**.

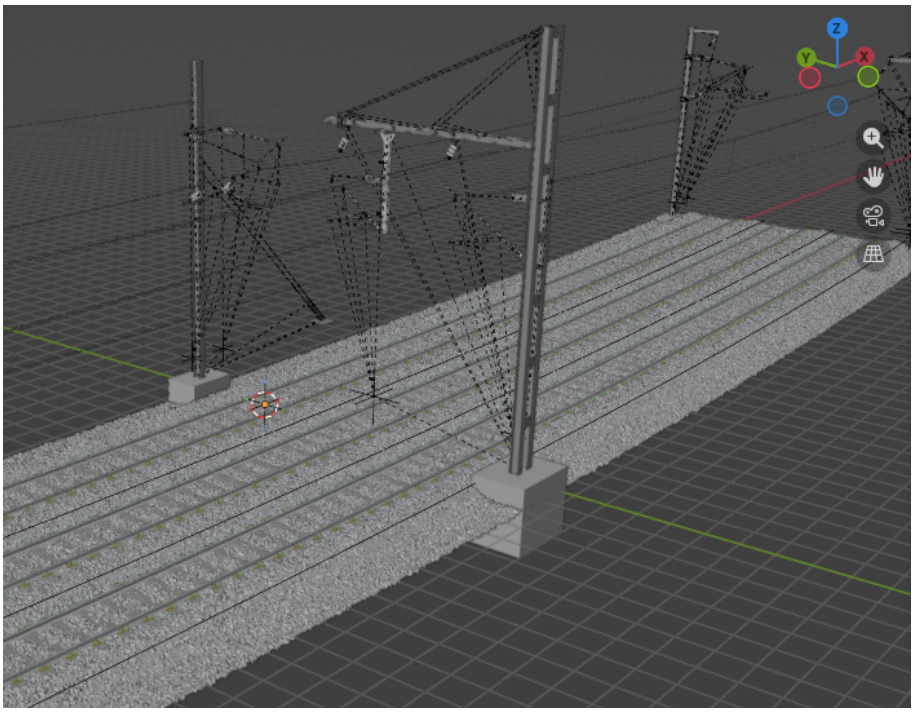

Figure 4.2: Blender view of the generated railway, with catenary equipments

The processing of the support position is made as follow in the code: Let $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^3$ be the three given points "start, mid, end". We define **4.1**

$$\mathbf{AB} = \mathbf{B} - \mathbf{A}, \qquad \mathbf{AC} = \mathbf{C} - \mathbf{A}. \tag{4.1}$$

The normal to the plane containing the points is

$$\mathbf{n} = \mathbf{AB} \times \mathbf{AC}, \qquad \hat{\mathbf{n}} = \frac{\mathbf{n}}{\|\mathbf{n}\|}, \tag{4.2}$$

with an exception raised if $\|\mathbf{n}\|$ is (numerically) zero (collinear points).

The midpoints of segments AB and AC are

$$\mathbf{M}_{AB} = \tfrac{\mathbf{A}+\mathbf{B}}{2}, \qquad \mathbf{M}_{AC} = \tfrac{\mathbf{A}+\mathbf{C}}{2}. \tag{4.3}$$

Directions perpendicular to AB and AC within the plane are obtained by

$$\mathbf{n}_1 = \hat{\mathbf{n}} \times \mathbf{AB}, \qquad \mathbf{n}_2 = \hat{\mathbf{n}} \times \mathbf{AC}. \tag{4.4}$$

Finally, the center of the circle $\mathbf{O}$ is the intersection of the perpendicular bisectors:

$$\mathbf{O} = \mathbf{M}_{AB} + \alpha\, \mathbf{n}_1 = \mathbf{M}_{AC} + \beta\, \mathbf{n}_2. \tag{4.5}$$

Once the center of the arc is found, we can process the tangent and perpendicular at the points in order to place the support. Let $\mathbf{O}$ denote the circle center and let $\mathbf{P}$ be the chosen point on the arc (start, mid or end). Define the radius vector from center to point:

$$\mathbf{r} = \mathbf{P} - \mathbf{O}. \tag{4.6}$$

Compute two radius vectors to endpoints used to get the plane normal **4.7** and the unit normal to the arc plane **4.8**:

$$\mathbf{r}_s = \mathbf{S} - \mathbf{O}, \qquad \mathbf{r}_e = \mathbf{E} - \mathbf{O}, \tag{4.7}$$

$$\hat{\mathbf{n}} = \frac{\mathbf{r}_s \times \mathbf{r}_e}{\|\mathbf{r}_s \times \mathbf{r}_e\|}. \tag{4.8}$$

The tangent at $\mathbf{P}$ is perpendicular to the radius and lies in the plane; it is obtained by the cross product of the normal and the radius vector, then normalized:

$$\mathbf{t} = \frac{\hat{\mathbf{n}} \times \mathbf{r}}{\|\hat{\mathbf{n}} \times \mathbf{r}\|}. \tag{4.9}$$

To get the horizontal (XY) perpendicular used for lateral offsets, project the tangent onto the XY plane:

$$\mathbf{t}_{xy} = (t_x, t_y, 0). \tag{4.10}$$

If $\|\mathbf{t}_{xy}\|$ is numerically near zero (tangent nearly vertical), a fallback horizontal unit vector is used. Otherwise, it is normalized and rotated by 90° about Z to obtain the outward perpendicular in XY:

$$\mathbf{p}_{xy} = \frac{(-t_y, t_x, 0)}{\|(-t_y, t_x, 0)\|}. \tag{4.11}$$

The normalized perpendicular vector obtained in Eq.**4.11** can then be used to place the supports and the LIDARs precisely on the scene.

4.1.2 Catenary implementation

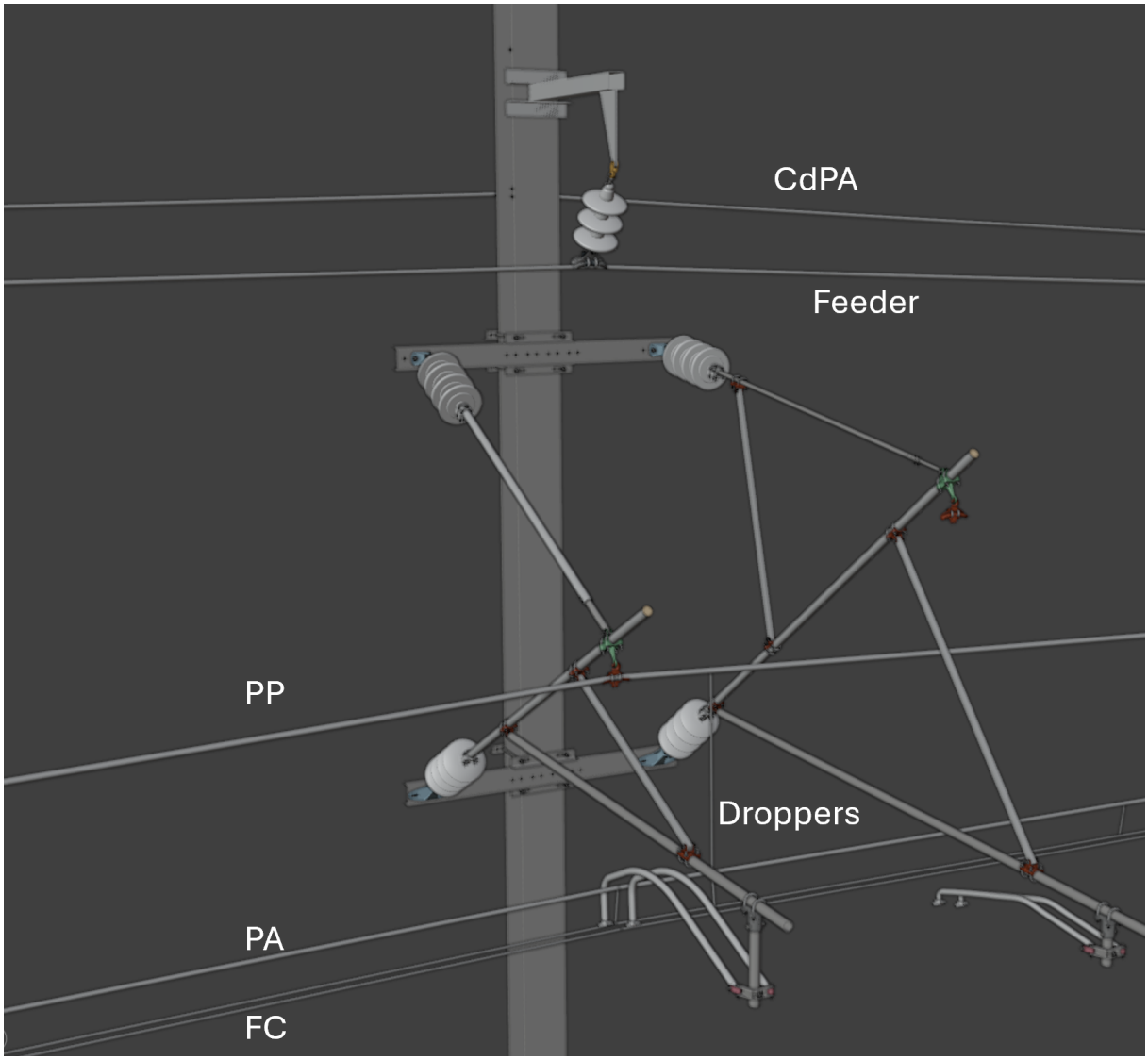

Figure 4.3: Feeder, CdPA and catenary with PA on support

In order to better predict catenary types, they were also implemented within the simulation. Although they were simplified, they were made so that the output of the LIDAR is looking as closely as the real situation. The catenaries parameters can be adjusted as follows in Tab. **4.2**:

Table 4.2: Catenary Manager Parameters — Values and explanation

Variable	Value	Explaination
REGEX_CASE_INSENSITIVE	True/False	Whether regex matching for anchor names is case-insensitive.
ANCHOR_REGEX_XXX	string	Regex to detect XXX cable anchor object names.
ENABLE_CATENARY	True/False	Toggle generation of PP / PA / FC and pendulums.
ENABLE_FEEDERS	True/False	Toggle generation of feeder cables.
ENABLE_CdPA	True/False	Toggle generation of CdPA cables.
FC_DOUBLE_PROBABILITY	0.0–1.0	Probability to use double FC/PA on a side (controls random/side decision).
FC_DOUBLE_SPACING_MM	mm	Lateral spacing between the two FC lines when double FC is used.
SAG_REFERENCE_LENGTH_MM	mm	Reference length used to scale sag with span length.
SAG_EXPONENT_ALPHA	float	Exponent controlling how sag scales with span length.
SAG_XXX_MM	mm	Reference sag for XXX (cable).
DIAM_XXX_MM	mm	Diameter of XXX cable.
CURVE_RESOLUTION_POINTS_PER_10M	int	Sampling density for parabolic curves (points per 10 m).
CURVE_CONVERT_TO_MESH	True/False	Whether created curves are converted (duplicated) to meshes.
MESH_SMOOTH_SHADING	True/False	Enable auto-smooth on created mesh/curve objects.
PENDULUM_SPACING_MM	mm	Target spacing along PP between pendulums.
PENDULUM_JITTER_MM	mm	Random jitter applied to pendulum base positions.
FEEDER_REQUIREMENT_GROUP_SIZE	int	Number of supports considered per feeder/CdPA grouping window.
FEEDER_MIN_ANCHORS_PER_GROUP	int	Minimum anchors required in a group to create a feeder/CdPA polyline.
CLASS_ID_XXX	int	Optional class IDs assigned to objects for export/labeling.
ASSIGN_CLASS_IDS	True/False	Whether to write ‘class_id‘ property on created objects.

For each catenary wire, the code computes a local span frame, scales a reference sag using a power law, and samples a parabola along the baseline between $\mathbf{A}$ and $\mathbf{B}$. The sampled points are returned as world-space 3D vectors. Then, the calculation of the span frame is made as follows:

We define the chord vector and its length:

$$\mathbf{x} = \mathbf{B} - \mathbf{A}, \qquad L = \|\mathbf{x}\|.$$

A numerical guard ε (the code uses 10^{-6}) prevents division by zero. If $L < \varepsilon$ the implementation uses a default direction and $L \leftarrow 1$.

Otherwise the longitudinal unit vector is

$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{L}.$$

A global upward axis is fixed as

$$\mathbf{z} = (0, 0, 1),$$

and a lateral axis is computed by the cross product

$$\mathbf{y} = \mathbf{z} \times \hat{\mathbf{x}}.$$

If $\|\mathbf{y}\| < \varepsilon$ (span nearly vertical) an arbitrary lateral axis $\mathbf{y} = (0, 1, 0)$ is chosen; otherwise $\mathbf{y}$ is normalized. The function returning $(\hat{\mathbf{x}}, \mathbf{y}, \mathbf{z}, L)$ is used for orientation, lateral offsets and the length L that drives sag scaling and sampling density.

A reference sag value f_{ref} (specified per cable type in configuration) is scaled with a power law:

$$f(L) = \begin{cases} f_{\text{ref}}, & L_{\text{ref}} \le \varepsilon, \\ f_{\text{ref}} \left(\dfrac{L}{L_{\text{ref}}} \right)^{\alpha}, & \text{otherwise}, \end{cases}$$

where L_{ref} is `SAG_REFERENCE_LENGTH_MM` and α is `SAG_EXPONENT_ALPHA`. This makes sag vary smoothly with span length and lets the user tune the exponent α to increase or reduce sag sensitivity to length.

The vertical deflection is modelled by a symmetric parabola about the midpoint added (with negative sign) to a linear baseline interpolation of endpoint heights. Parameterise the span by $t \in [0, 1]$ and $x = Lt$. The baseline point (without sag) at coordinate x is:

$$\mathbf{base}(x) = \mathbf{A} + \hat{\mathbf{x}}\, x.$$

The linear interpolation of endpoint heights is

$$z_{\text{line}}(t) = A_z + (B_z - A_z)\, t.$$

Let $s = f(L)$ be the scaled sag. The parabolic sag term is

$$z_{\text{par}}(x) = \frac{4\, s\, x\, (L - x)}{L^2},$$

which satisfies $z_{\text{par}}(0) = z_{\text{par}}(L) = 0$ and $z_{\text{par}}(L/2) = s$. The final z value at parameter t is

$$z(t) = z_{\text{line}}(t) - z_{\text{par}}(x),$$

so the sample point is $\mathbf{p}(t) = \big(\mathbf{base}_x(x),\ \mathbf{base}_y(x),\ z(t)\big)$.

Sampling density uses a parameter `points_per_10m`. Because units are mm the code computes the span length in "10 m units" as $L/10000$. The number of samples n is:

$$n = \max\Big(3,\ \Big\lfloor \max\big(2.0, \tfrac{L}{10000}\big) \cdot \texttt{points\_per\_10m} \Big\rfloor\Big),$$

which ensures at least three points and a minimum density even for short spans. Samples are taken at $t = i/(n-1)$ for $i = 0, \ldots, n-1$.

Finally, pendulums are placed along the wire with a configurable spacing $s = \max(1.0,$ `PENDULUM_SPACING_MM`$)$. The active span length is estimated as $L_{\text{est}} = \|\text{PP}_{\text{end}} - \text{PP}_{\text{start}}\|$ and the number of pendulums is chosen as $n = \max(1, \lfloor L_{\text{est}}/s \rfloor)$. Given the sampled point list $\{\mathbf{p}_k\}_{k=0}^{m-1}$ produced by the parabola's sampling, a sampling step is computed

$$\text{step} = \max\big(1, \lfloor (m-1)/(n+1) \rfloor\big),$$

and indices $i \in \{\text{step}, 2 \cdot \text{step}, \ldots\}$ are used. For each chosen index the base position $\mathbf{b} = \mathbf{p}_i$ is perturbed by a random jitter of magnitude controlled by 'PENDULUM_JITTER_MM'. If a PA object exists and 'PENDULUM_DOUBLE_TO_PA' is enabled, the code finds the nearest point on the PA (sampling the PA mesh or spline) and creates a pendulum from $\mathbf{b}$ to that PA location; otherwise it projects $\mathbf{b}$ to the nearest point on the available FC curves and creates a pendulum to that FC point. When two FCs are present, a second pass connects PA points to FC points and pendulums are staggered (quinconced) by an offset offset $= \max(1, \lfloor \text{step}/2 \rfloor)$ to better represent reality. Unlike in reality each dropper is created as a short curve via '_create_curve_from_points' instead of stirrup droppers as can be seen in Fig.**4.4**. This simplification showed great results none the less to detect PA to FC droppers in real applications. The final assembly can be seen on Fig.**4.3**.

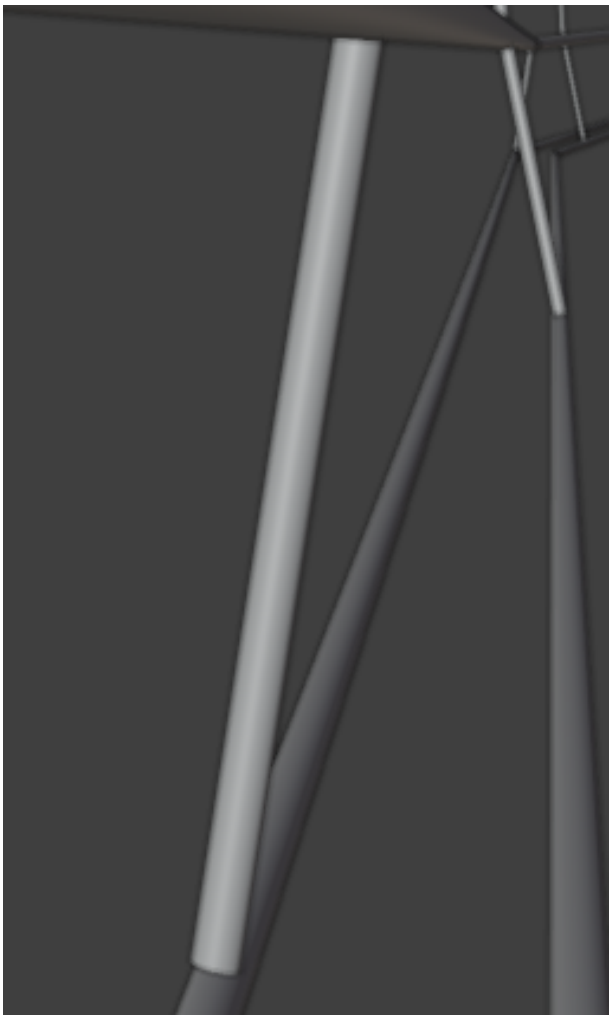

Figure 4.4: Droppers between PA and FC

4.1.3 LIDAR

Blensor and Blainder were left due to lack of modularity for our case, mostly the retrieving of the objects information. A second implementation was made using the blender API's ray casting capabilities, but the processing time was way too long to be exploited.The solution retained instead was a C++ code with pybind11 custom LIDAR that uses Embree. The module performs large-scale batched ray intersections against a static triangle mesh and returns hit positions together with object and semantic class annotations. It was designed to achieve high throughput, low-copy buffer handling for Python interoperability and flexible semantic mapping. The implementation exposes a `Lidar` class with the following responsibilities:

- Construct an Embree `RTCScene` from vertex and index arrays provided by Python.
- Store a per-primitive integer mapping (primitive index $\mapsto$ object id) as geometry user data.
- Provide batched ray intersection via a `scan_with_info` method that accepts arrays of ray origins and directions and returns an $R \times 5$ float32 array containing $(x, y, z, \text{object\_id}, \text{class\_id})$ per ray.
- Support per-primitive class ids and flexible rule-based semantic mapping (regex-based ancestry rules, explicit maps, legacy name-based rules).

- Use a reusable thread pool to parallelize ray processing across CPU cores.

Given vertices $V \in \mathbb{R}^{N\times 3}$ and triangle indices $I \in \mathbb{Z}_{\geq 0}^{M\times 3}$, the module allocates an Embree triangle geometry and uses `rtcSetNewGeometryBuffer` to obtain raw pointers for vertex and index buffers. It copies contiguous vertex and index memory into Embree's buffers using `memcpy` to avoid per-element overhead. And it attaches a pointer to an internal vector `prim_objid_storage` as geometry user data via `rtcSetGeometryUserData`. This allows, at intersection time, to retrieve the object id for a hit primitive by reading the stored array at the primitive index.

Each input ray is represented by origin $\mathbf{o} \in \mathbb{R}^3$ and direction $\mathbf{d} \in \mathbb{R}^3$. The intersection test follows the standard parametric form:

$$\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}, \qquad t \in [t_{\text{near}}, t_{\text{far}}].$$

For each ray the code initializes an `RTCRayHit` structure, sets `tnear`, `tfar`, mask and flags, then calls Embree's `rtcIntersect1`. Then, on hit, computes hit point coordinates $\mathbf{p} = \mathbf{o} + \text{tfar} \cdot \mathbf{d}$, retrieves the geometry via `rtcGetGeometry` and the primitive-to-object map via `rtcGetGeometryUserData`, and looks up the object id o for the reported primitive id. It calls the semantic resolution routine to produce a class label for the hit. And writes $(p_x, p_y, p_z, o, \text{class})$ into a preallocated contiguous float buffer. Missed rays are filled with `NaN`.

To reduce per-call overhead and maximize CPU utilization a fixed `ThreadPool` is created once with `std::thread::hardware_concurrency()` workers. For a scan of R rays, the rays are partitioned into approximately equal chunks ($\lceil R/T \rceil$ per thread, with T the number of worker threads) and each chunk is enqueued as a task; small R uses a single-threaded fast path to avoid thread management overhead. A reusable `std::vector<float>` serves as the output buffer (size $5R$). The buffer is exposed to Python as a NumPy array that shares the buffer memory; ownership is preserved by allocating a heap `std::shared_ptr` wrapped in a Python capsule so the C++ buffer remains valid until Python releases it.

The module supports multiple, ordered mechanisms for mapping hits to class labels (higher-precedence rules are applied first):

1. Per-primitive class ids: an array of length M may be supplied (`set_prim_class_ids`). If present for a hit primitive, its class id is considered first. This class id may be remapped via:
   - Flexible rules that bind face class ids to output labels.
   - A direct face-class $\mapsto$ label dictionary.
2. Object semantics: each object in a semantics table carries fields (id, name, parent, class string, class id). The object's `class_id` is considered next, with similar remapping via flexible rules or an object-class map.
3. Flexible regex-based ancestry rules can express constraints on the name ancestry (regular expressions for Parent, Parent2, and exclusions). Rules may also optionally require the source class id (from face or object) to match, this allows to map also the different elements generated within Blender. These rules are checked and the first matching rule yields the output label.

4. Legacy name-based ClassRule list: exact string matching of ancestor names is supported for backward compatibility.

5. If no rule applies, the result is `-1` (internally "unclassified"); a configurable `default_label` may be substituted for unclassified hits before returning to Python.

Ancestry traversal is supported by maintaining:

- A vector of `Semantic` entries and a `name_to_index` map.
- A `parent_index` array storing the parent index for each semantic entry, enabling efficient upward traversal to collect names used for regex matching.
- Geometry construction copies vertex and index buffers in $O(N + M)$ time and then commits the Embree scene.
- Each ray intersection cost is dominated by Embree's BVH traversal and triangle intersection; amortized per-ray cost is sublinear in M due to spatial acceleration structures.
- The per-call overhead is minimized by reusing the thread pool, reusing an output buffer, and avoiding Python-level allocations inside the hot loop.
- Thread-safety: inputs are read-only contiguous arrays; Embree intersection calls are per-thread with locally created `RTCRayHit` structures. No shared mutable state is modified during ray processing except for writing disjoint regions of the output buffer.

4.1.4 Output format and Python interop

The returned NumPy array has shape $(R, 5)$ and dtype `float32`, with columns ordered as:

$$(x,\ y,\ z,\ \text{object\_id},\ \text{class\_id}).$$

Missed rays are represented by `NaN` in all five columns. A Python-side capsule holds a `std::shared_ptr<std::vector<float>>` to preserve the lifetime of the underlying buffer until the NumPy array is garbage-collected.

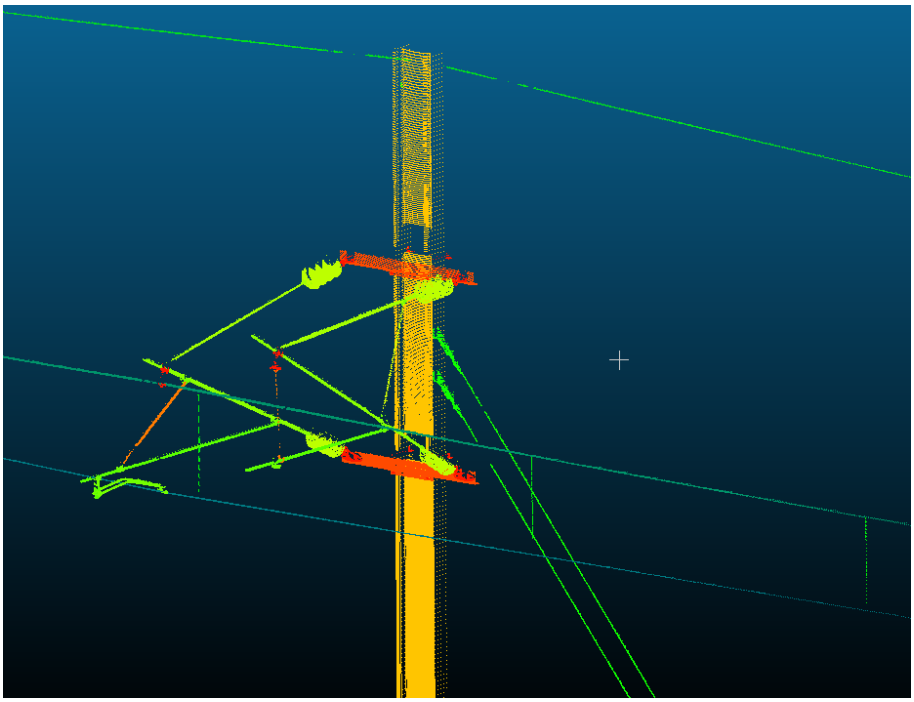

Figure 4.5: Segmented LIDAR simulation result on a support, with catenary.

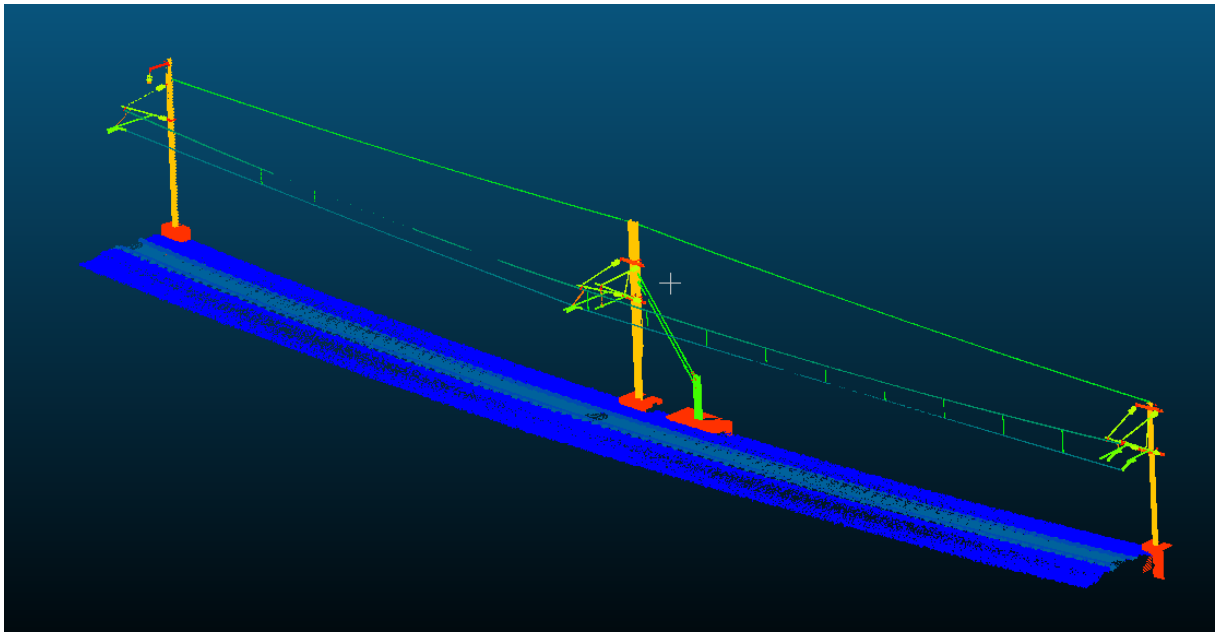

Figure 4.6: Complete segmented LIDAR simulation result for 1 track

The LIDAR output has modifiable parameters listed in table **4.3**. An example of the resulting segmented point cloud can be seen in Fig. **4.5** and **4.6**. It makes the simulation closer to the the real acquisitions, helping the model to generalize better.

Table 4.3: LIDAR Parameters — Values and explaination

Variable	Value	Explaination
HEIGHT_OFFSET	2000 mm	Height of the LIDAR above their positioning point (2 m).
MAX_RANGE	150000 mm	Maximal range of the sensor (150 m).
MIN_RANGE	200 mm	Minimal range of the sensor (0.2 m).
ACCURACY	2 mm	Precision of the measurments.
HORIZONTAL_FOV_MIN	0°	Minimal limit of the horizontal field of view.
HORIZONTAL_FOV_MAX	360°	Maximal limit of the horizontal field of view (complete scan).
VERTICAL_FOV_MIN	-70°	Minimal limit of the vertical field of view. (to bottom).
VERTICAL_FOV_MAX	90°	Maximal limit of the vertical field of view (to top).
HORIZONTAL_RESOLUTION	0.05°	Horizontal angular resolution (smaller = more points).
VERTICAL_RESOLUTION	0.05°	Vertical angular resolution (smaller = more points).
NOISE_FACTOR	0.1 mm	Standard deviation of the Gaussian noise added to simulate the error.
ADD_NOISE	True	Indicates whether realistic noise is added to the measurements.
SAMPLING_FACTOR	1.5	Subsampling factor (1.0 = full resolution; $value < 1$ reduces the number of points).

4.2 Model training for segmentation, tuning and selecting

The training was made from simulated dataset and 75% of the real dataset was used. The rest of the real dataset was used for the validation process. This allows us to better represent all the configurations possible while assessing correctly the training of the model, the simulated dataset not containing any other ground than the ballast and no vegetation and some other classes in the simulated data not represented within the real dataset. Another difference between the two dataset is the LIDAR, the simulated one is static, but for now all the real dataset is captured using a dynamic one. There are actually 30 classes for the segmentation, 4 more than the previous dataset. But this list will expand to include more classes useful for the design study like specific buildings or important terrain constraints (embankments, cliffs, etc...).

The trainings were made with 10 m * 10 m chunks of 8192 points and batch of 4 due to material limitations. The downsampling in the chunks is made by FPS in order to keep the important points and shapes. This made 3008 chunks used for training and 276 for validation from real data only to see if the model is indeed converging and does not overfit or underfit. From that,

the two models were trained with varying hyperparameters.
The hyperparameter search space was set for each model using Ray Tune for automated and efficient testing. This helped to reduce the number of tests and thus, the testing time. The concurrent testing and logging using MLFlow allowed to compare the models together with their runs.

Since a study was already made for training the previous model, some parameters were kept, avoiding a full new research. Loss function was kept as cross entropy, but the 3D-UMamba was also tested on different loss functions in order to study its impact on the new model. The optimizer used was Adam [29] due to the performances it achieved on the old model.

Table 4.4: Ray Tune search space

Parameter	Search space	Notes
Learning rate	log-uniform (1e-4, 1e-3)	continuous, sampled on log scale
Weight decay	{1e-3, 5e-3, 1e-2}	categorical (better regularization)
Betas	{(0.9,0.999), (0.9,0.95)}	categorical, Adam beta tuples
Gradient clip val	{0.5, 1.0, 2.0}	categorical, gradient clipping value

Tab. **4.4** represents the hyperparameter tuning for both of the models. The result of this experiment is given in Tab. **4.5**.

Table 4.5: Model Hyperparameters and Validation mIoU, best model of Ray Tune search space for 50 epochs

Model	Gradient clip val	Betas	lr	Weight decay	Validation mIoU
3D-UMamba	1.0	[0.85, 0.999]	0.00044	0.01	textbf0.39
PTV3	1.0	[0.9, 0.999]	0.00034	0.001	textbf0.33

From Tab. **4.5**, the hyperparameters were globaly the same for the two models but 3D-UMamba necessits a greater weight decay to learn properly. On this experiment Fig. **4.7**, 3D-UMamba converges to a higher validation mIoU. But it is also necessary to underline the fact that both models were having poor results, mostly due to the reduced architecture compared to the production model due to hardware limitations.

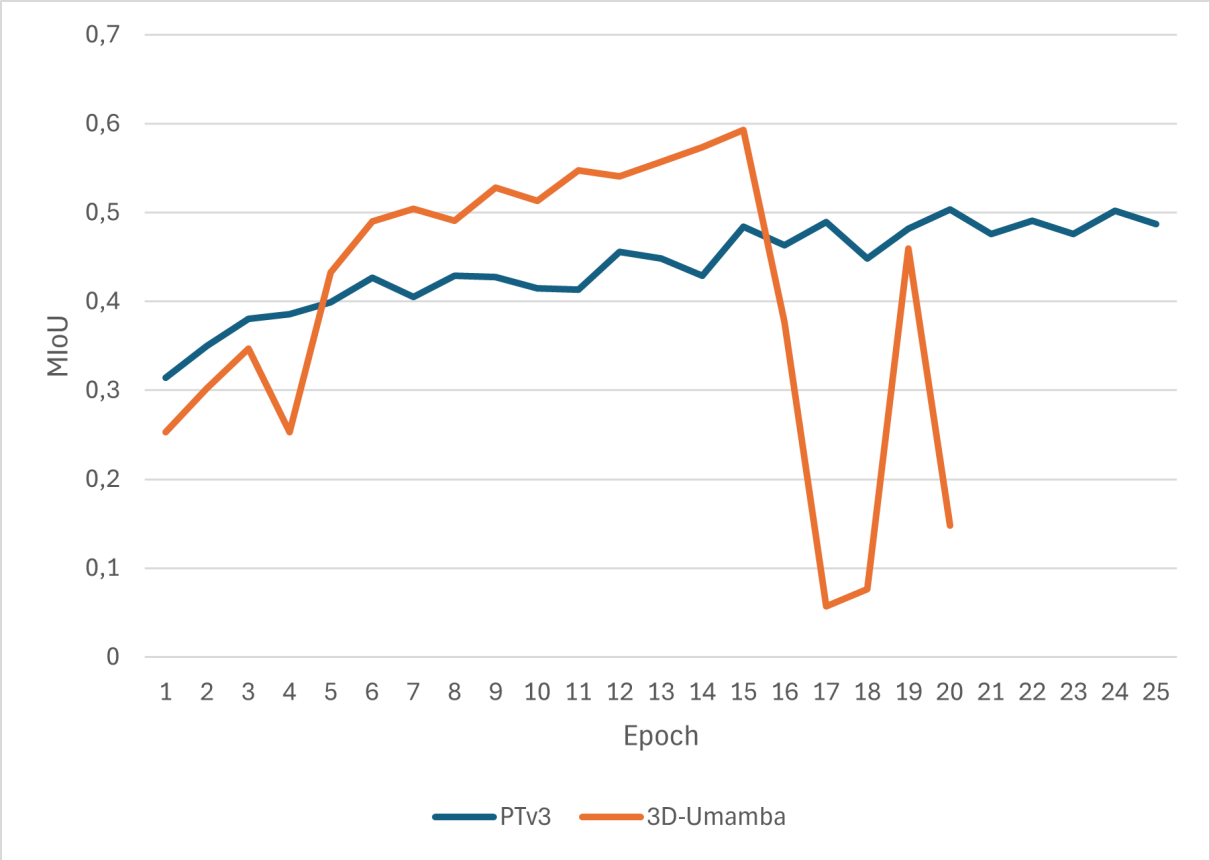

Figure 4.7: Validation mIoU results of the two best models for each Ray Tune search space

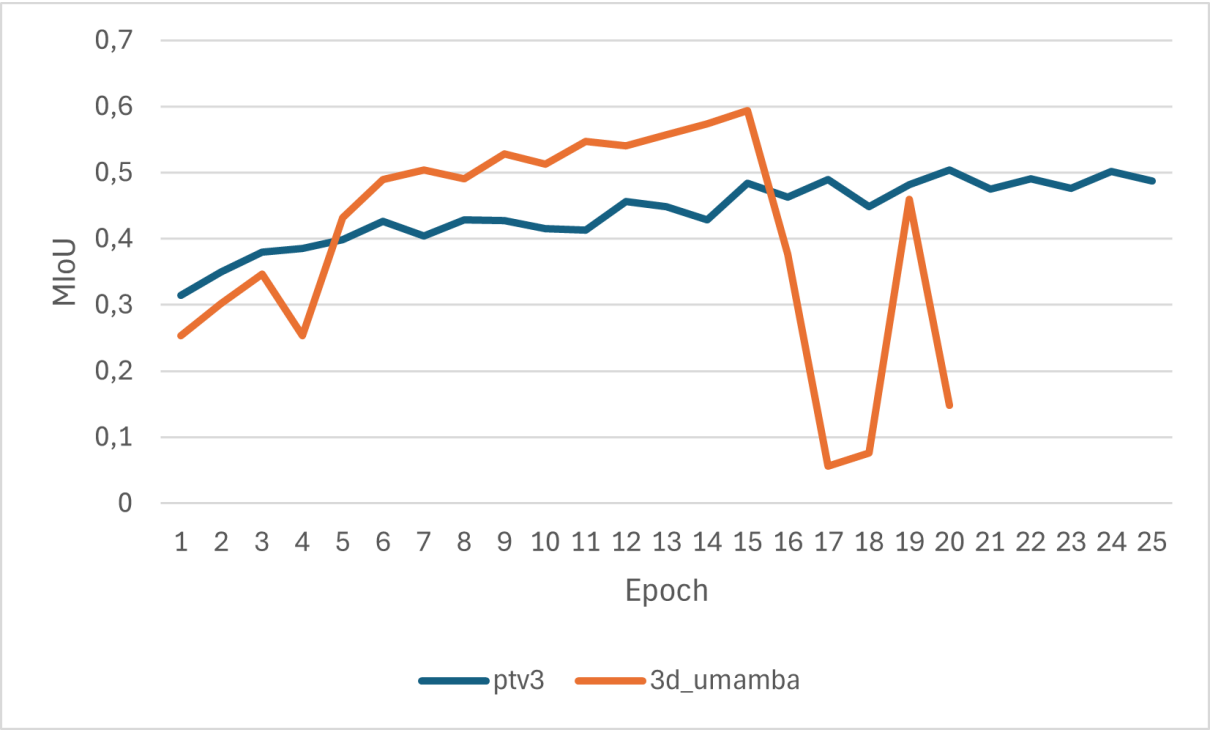

Figure 4.8: Validation mIoU with larger model parameters

In Fig. **4.8**, the PTv3 encoder and decoder patch size was augmented from 64 per step to 128, leading to and improvement in performances. The 3D-UMamba model was trained on a higher learning rate of 0.005 to avoid possible local minimum but the model was untouched due to a hard coded structure and a limitation of VRAM availability. While PTv3 showed a clear improvement, 3D-UMamba was overfitting, meaning that the model fails at generalizing for unseen data. But the model showed also an improvement in maximum validation mIoU, meaning that the model's architecture is probably good but data augmentation has to be improved and better hyperparameters have to be found.

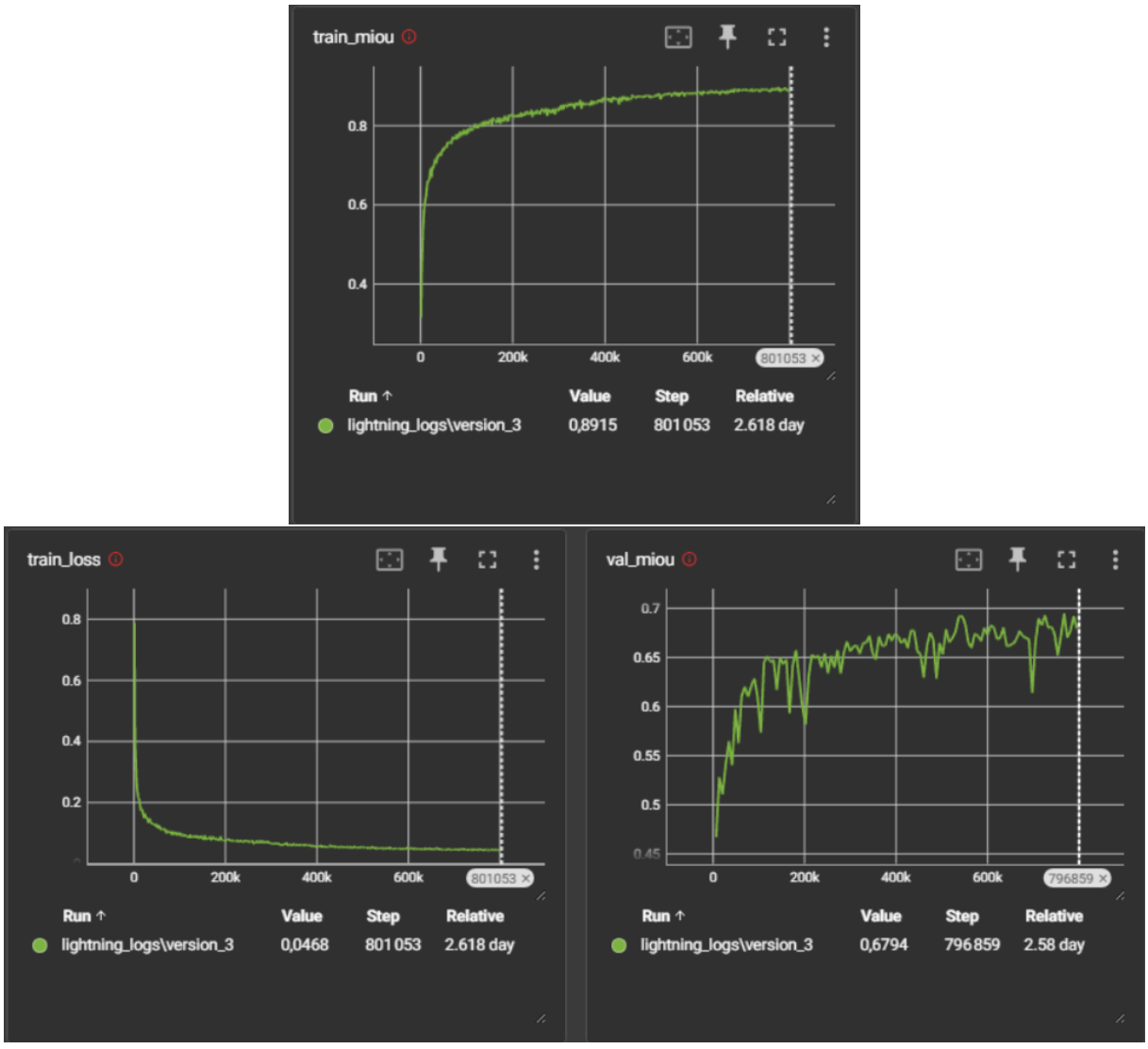

Figure 4.9: Training logs with the production model

Finally, the processing time difference is quite important but mostly due to the FPS done before each epochs for the model input, but also for the inference time that is measured as around 50 to 100% slower.
The resulting segmentation can be seen in Fig.**4.10** and shows great performance and accuracy on on this point cloud. But when introduced to another LIDAR type like in Fig. **4.11**, the model

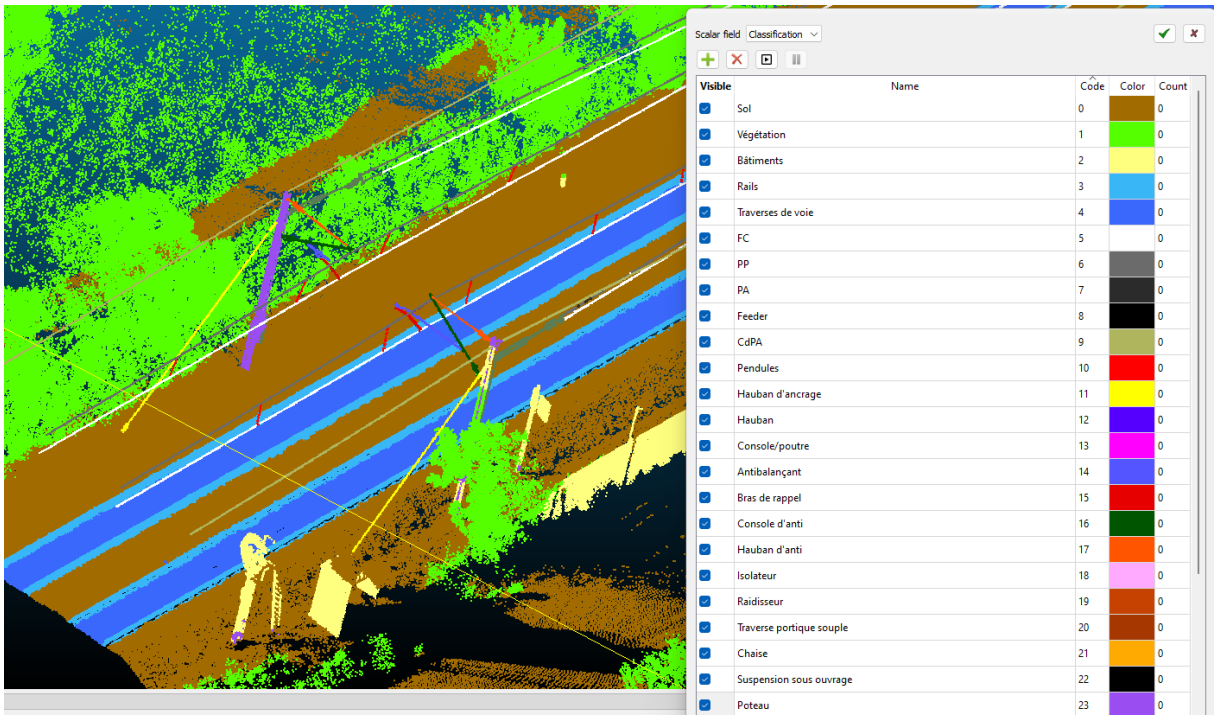

Figure 4.10: Segmentation with the production model

performs slightly worse. It emphasizes the need of enlarging the DB to more data of different origin and sensor.

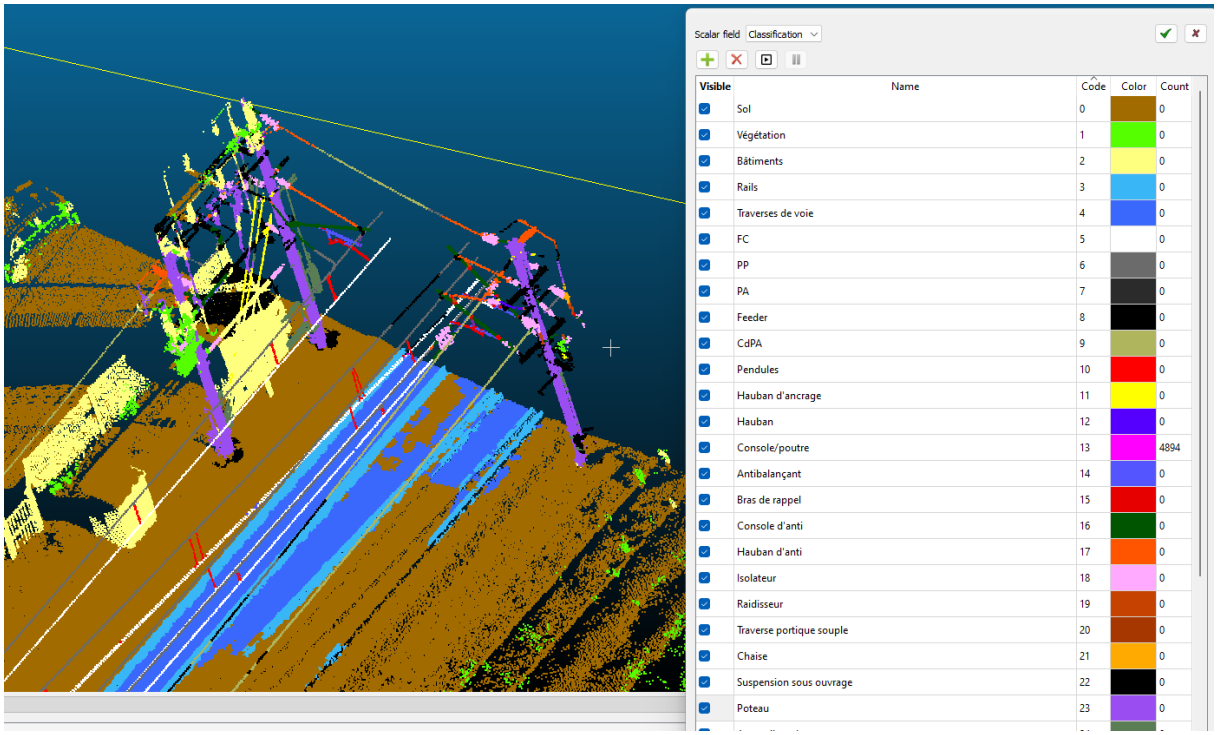

Figure 4.11: Segmentation on another type of LIDAR than training DB

5 Analysis and Discussion

5.1 Analysis of Blender dataset generation

The risk with generating artificial data is to generate too precise or specific and recurent data for the model to train on. This could lead to overfitting. But the gains in time and modularity to create and modify the dataset can be huge if the model does not overfit on the artificial data. On the current model, the dataset improves the detection and allows classes that were under represented to be better detected. This reduction on class imbalance and positive impact shows the power of this artificial point cloud generation.

But this implementation is still not optimal and the results must be weighted. Indeed, the track representation can be considered too perfect and simplistic. The data acquisition is only in static LIDAR type, which does not correspond to all the point clouds in real projects. Moreover, with a processing time of around 8-10 h for over 140 railway simulations, the implementation can still be improved in performance, realism and diversity of cases.

Additionnaly, the current LIDAR implementation has some limitations:

- The code assumes a static scene: geometry updates require reconstructing the Embree scene.
- Regex-based rules add CPU cost per hit for semantic resolution; heavy use of complex regexes may affect throughput.
- The module relies on the user to provide consistent per-primitive and semantics data; out-of-range primitive indices or malformed semantics may produce undefined classifications. The models hierarchy has to be

In addition to that, the 3D models are only for one type of catenary (25 kV), leading the model to not be able to reliably detect the supports for 1.5 kV catenaries with it's specific equipments. But overall, the data generation showed promising results for detecting more classes.

5.2 Analysis of model tuning and training

During tuning phases, PTv3 showed great learning stability, performances for a limited execution time. Unlike 3D-UMamba that showed an instable training across the tests. 3D-UMamba also had a longer inference time, making it more penalizing in case of future use on large point clouds. Nevertheless, it can be seen that the model is able to learn and achieve great results. Further tuning of more parameters like epsilon for the optimizer, lowering even more the learning rate and other will be tested in the future.
Along with that, different methods of data augmentation will also be tested on both models to

improve generalization.

For detecting more reliably, manual segmentation will have to be made to include old assemblies that are not represented in current BIM project. Moreover, using the current model, manual segmentation can be greatly assisted. Adding more real world data will also benefit the model generalizing to situations that are not always represented within the LIDAR simulation.

5.3 Discussion of the results

The current results are inline with the will of adding more classes, while being reliable. But currently more work has to be invested in order to detect reliably all classes for a maximum coverage of the catenary equipment.
The scene construction can still be improved to better represent reality and the diversity of cases but still gives really good results in detecting new classes the model was not exposed to with the old DB. So even if improvement can be made, the current result if sufficient for usage and due to the absence of BIM project representing older catenary types, point clouds will have to be manually segmented to improve the DB.
The current implementation of the LIDAR simulation is slow and restricted. Its enhancement can be good for model performance but along with that, real segmented data point clouds will be even more needed.
The training of the models has to be perfected too if the method has to be reliable for further assisting the work done in the design study.

Conclusion

This work aimed to improve the segmentation of point clouds within the railway catenary domain by generating synthetic data using Blender and training two different deep learning models. The main advance made was the implementation of a modular and parametric scene generation pipeline in Blender, allowing the creation of diverse railway catenary scenes with accurate geometry and LIDAR simulation. This synthetic data was used to augment the training dataset for two deep learning models: PTV3 and 3D-UMamba. Both models were trained and evaluated on the augmented dataset. The results showed that both models benefited from the synthetic data, with 3D-UMamba achieving higher validation mIoU. However, 3D-UMamba exhibited training instability and longer inference times compared to PTV3. Further tuning and data augmentation are needed to improve generalization and reliability. Overall, the work demonstrated the potential of synthetic data generation for enhancing point cloud segmentation in railway catenary applications. Future work should focus on adding more point clouds to the training dataset, improving the realism and diversity of the synthetic scenes, and refining the model architectures and training procedures. The current BIM projects do not represent all the catenary types in use, so manual segmentation of point clouds will be necessary to cover older assemblies. With continued development, the approach can provide a valuable tool for automating and improving railway catenary design studies.

The main contribution of this work will be to ease out the retrieving of information from point clouds in the catenary domain. Further work will be needed both to improve the segmentation results but also to extend the uses of the segmented point clouds.
Future perspectives include the retrieving of main geometric parameters from the segmented point clouds to automatize the design study even more. This will require further development of algorithms to extract features such as cable sag, support positions, and other relevant measurements from the segmented data. Additionally, integrating the segmentation and feature extraction into a streamlined pipeline for railway catenary design studies can enhance efficiency and reduce manual effort.

Acknowledgements

I first would like to express my deepest appreciation to Laurent ROLLY, Martial DURRIS, Ludovic CHUET and Rémi PEGHEON, for accepting me again in a very interesting internship, with nice future perspectives. But also for letting me be able to learn, try and do by myself while explaining and guiding wisely.

I am also grateful to all TSO Caténaires employees for being interesting, motivating and making this company a nice working place of continuous learning and improvement.

I will thank them all again for leading me to were I am today, supporting my studies and making me learn valuable things in so many domains, including catenaries.

I thank my parents, for supporting me in my internship and my thesis, and also for helping and supporting me. As well as my father and my brothers for helping me move in Lyon.

I am grateful to Tartu University, Heiki KASEMÄGI and Liina VIIKMAA, for letting us be able to do an internship in France, but also for 3 semesters of learning another face of engineering and feeding my curiosity.

I would like to extend my sincere thanks to ECAM university, enabling me to have all these experiences and making me able to learn, try, explore but most importantly, do things and giving me the tools to better learn by myself.

The use of GPT within this thesis was limited to format equations and tables in Latex, and for debugging within the code.

Bibliography

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *arXiv preprint arXiv:1612.00593*, 2016.

[2] X. Yu, L. Tang, Y. Rao, T. Huang, J. Zhou, and J. Lu, "Point-bert: Pre-training 3d point cloud transformers with masked point modeling," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[3] Y. Pang, W. Wang, F. E. Tay, W. Liu, Y. Tian, and L. Yuan, "Masked autoencoders for point cloud self-supervised learning," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part II*, pp. 604–621, Springer, 2022.

[4] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," 2019.

[5] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on $\mathcal{X}$-transformed points," 2018.

[6] LMD0311, "PointMamba (github repository - models folder)." `https://github.com/LMD0311/PointMamba/tree/main/models`, 2025. GitHub repository. Accessed: 2025-12-01.

[7] D. Lu, L. Xu, J. Zhou, K. Gao, J. Li, J. Li, *et al.*, "3dlst: 3d learnable supertoken transformer for lidar point cloud scene segmentation," *International Journal of Applied Earth Observation and Geoinformation*, vol. 140, p. 104572, 2025. Open access. Accessed: 2025-12-01.

[8] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[9] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences," in *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[10] V. K. A. Nina Varney and Q. Graehling, "Dales: A large-scale aerial lidar data set for semantic segmentation," 2016.

[11] D. Lu, L. Xu, J. Zhou, K. Gao, Z. Gong, and D. Zhang, "3d-umamba: 3d u-net with state space model for semantic segmentation of multi-source lidar point clouds," *International Journal of Applied Earth Observation and Geoinformation*, vol. 136, p. 104401, 2025. Open access. Accessed: 2025-12-01.

[12] d62lu, "3D-UMamba: 3d u-net with state space model (github repository)." https://github.com/d62lu/3D-UMamba, 2025. GitHub repository. Accessed: 2025-12-01.

[13] X. Wu, L. Jiang, P.-S. Wang, Z. Liu, X. Liu, Y. Qiao, W. Ouyang, T. He, and H. Zhao, "Point transformer v3: Simpler, faster, stronger," in *CVPR*, 2024.

[14] D. Lu, J. Zhou, K. Gao, J. Du, L. Xu, and J. Li, "Dynamic clustering transformer network for point cloud segmentation," *International Journal of Applied Earth Observation and Geoinformation*, vol. 128, p. 103791, 2024. Open access. Accessed: 2025-12-01.

[15] D. Liang, X. Zhou, W. Xu, X. Zhu, Z. Zou, X. Ye, X. Tan, and X. Bai, "Pointmamba: A simple state space model for point cloud analysis," in *Advances in Neural Information Processing Systems*, 2024.

[16] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.

[17] J. J. B. J. W. Z. W. W. Xiangcheng Hu, Jin Wu and P. Tan, "Ms-mapping: An uncertainty-aware large-scale multi-session lidar mapping system," 2024.

[18] J. J. W. Z. Xiangcheng Hu, Jin Wu and P. Tan, "Ms-mapping: Multi-session lidar mapping with wasserstein-based keyframe selection," 2024.

[19] L. D. B. W. J. L. Z. D. C. W. Z. M. B. Y. Bo Qiu, Yuzhou Zhou, "Whu-railway3d: A diverse dataset and benchmark for railway point cloud semantic segmentation," *IEEE Transactions on Intelligent Transportation Systems*, 2024. doi:blue10.1109/TITS.2024.3469546.

[20] Z. H. R. Y. A. B. R. Kharroubi, A.; Ballouch, "Multi-context point cloud dataset and machine learning for railway semantic segmentation.," *Infrastructures 2024, 9, 71. https://doi.org/10.3390/infrastructures9040071*, 2024.

[21] P. Marion, R. Kwitt, B. Davis, and M. Gschwandtner, "Pcl and paraview – connecting the dots," in *Proceedings of the International Workshop on Point Cloud Processing in Computer Vision (PCP '12), held in conjunction with CVPR 2012*, (Providence, RI, USA), June 2012. Workshop paper (accepted), June 16–21, 2012.

[22] S. Reitmann, L. Neumann, and B. Jung, "Blainder—a blender ai add-on for generation of semantically labeled depth-sensing data," *Sensors*, vol. 21, no. 6, p. 2144, 2021.

[23] B. O. Community, *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[24] M. A. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar, "Accelerating the Machine Learning Lifecycle with MLflow," *IEEE Data Eng. Bull.*, vol. 41, pp. 39–45, 2018.

[25] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow, Large-scale machine learning on heterogeneous systems," Nov. 2015.

[26] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," 2018.

[27] O. Yadan, "Hydra - a framework for elegantly configuring complex applications." Github, 2019.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

Appendices

An incomplete, public version of the source code is available along the thesis.

Non-exclusive licence to reproduce thesis and make thesis public

I, Lucas Johannes Adriaan Marais

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   "Improving Railway 3D Point Cloud Segmentation"

   supervised by Ludovic Chuet and Veiko Vunder

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Lucas Johannes Adriaan Marais
22.12.2025