Point Cloud Classification and Segmentation for Catenary Systems

By Zino J. Vieth – z.j.vieth@student.utwente.nl

BSc. Creative Technology, Faculty of EEMCS, University of Twente

Supervised by Faizan Ahmed

Critically observed by Champika Epa Ranasinghe

Client: Ambient Intelligence Research Group (Saxion University of Applied Sciences) and Strukton Rail

Client Supervisors: Jeroen Linssen, Bram Ton

18/02/2022

Abstract

The Dutch railway is transitioning towards digitization to improve the efficiency of maintenance and reduce material waste and system downtimes. Strukton Rail and the Saxion University of Applied Sciences are working on deep-learning-based methods to digitize the Dutch railway's catenary system. This paper explores the development of a data pipeline for the reconstruction of 3D models from point cloud-based railway scenes utilizing a provided CAD catalogue. We used a two step approach to achieve this goal: segmentation and object retrieval from the catalogue. In the first step, the deep learning-based segmentation method PointNet++ was used. We used a RANSAC and point pair features-based template matching implementation in the second step. Testing with three substantially different catenary components generates final region overlap scores ranging from 60% to 100%, depending on the quality of segmentation results, down-sampling parameters, and the number of RANSAC iterations. These results confirm the applicability of the approach. The manual steps of the current pipeline suggest a future need for the use of instance segmentation models, component-variant consideration, and automation of CAD model placement.

Acknowledgement

First and foremost, I'd like to thank my supervisor Faizan for giving me the opportunity to learn so much about machine learning and data science in this project, and for all his patience and guidance along the way. This paper's completion is also owed to all the help and answers to many questions, as well as inspiration in the form of his personal attempts at some of my challenges that Bram of the Ambient Intelligent Research Group has given me throughout the project. I thank Jeroen for his support as the project manager at AMI. I'd also like to thank my critical observer Champikah for taking the time to keep up with this project and hold me accountable. Also, the University of Twente, the Saxion University of Applied Sciences and Strukton Rail for providing me with the opportunity of this fun and challenging project.

Table of Contents

List of Figures and Tables

Chapter 1 – Introduction

The Dutch railway network is the pillar of public and ware transport in the Netherlands, making it an invaluable piece of infrastructure that many people rely on daily. With around 6830 km of track [1], monitoring and maintaining the catenary system of the network, including the overhead lines, arches, and equipment mounted on them, is a difficult task to handle manually, as it requires trained experts to identify broken or deteriorated equipment.

Strukton Rail [2], a railway company based in Utrecht, has begun digitalizing the Dutch railway network in cooperation with ProRail [3], a government task organization responsible for the maintenance and extension of the national railway network infrastructure, and the Saxion University of Applied Science's Ambient Intelligence Research Group [4], from here on referred to as AMI. AMI's task so far has been the development of a deep-learning model capable of automatically identifying and locating the different components of a catenary arch from high-density point cloud data recorded by a LiDAR sensor mounted on a train. The following project concerns the development of a deep-learning model to segment catenary arches from low-density point clouds and supplementary panorama images of the arches, and reconstructing the identified components into a digital twin based on a CAD catalogue provided by Strukton Rail. The focus of this thesis is the development of a prototype data pipeline capable of this task.

1.1 Preliminaries

The catenary system of the railway network consists of the overhead wires and arches along the railway track that are used to provide locomotive and light rail vehicles with electricity through pantographs. The wires themselves are not object of interest in this project, but the arches and mounted components responsible for holding the wires and maintaining mechanical and electrical tension. Overhead lines and arches are easily and frequently damaged by natural occurrences such as strong winds and storms carrying branches and other debris, as well as through the process of aging. Maintaining the system currently requires trained experts to visit the locations where damage is suspected and manually identify the deficiency before the required repair can be ordered. A digital twin, generated through deep-learning and consistently updated by the trains on the network may help streamlining this process and allow for a safer and more efficient railway system.

Deep-learning is a category of machine learning that refers to an approach of data analysis based on artificial neural networks. A deep-learning model commonly consists of a layered hierarchy of neurons which are connected through weights that modify values when they pass from one neuron to the next.

The first layer takes an input and the last layer represents the output, usually in the form of percentages, representing the confidence the model has that an input corresponds to a certain label or class. Deep-learning models are trained by comparing the generated output for a specific input with the ideal output, which is usually assigned ahead of the training through tedious manual labour, and adjusting the weights of the network accordingly and automatically through a process called backpropagation. In simple classification problems, usually an entire input (such as an image) will be assigned one label by the output based on the highest confidence it can generate, whereas in segmentation problems, such as the one tackled in this project, each input component (such as each pixel in an image) is assigned a label, allowing the model to differentiate and locate different objects in an image.

The primary data type used in this project are point clouds, unordered lists of 3D-coordinates (potentially containing additional information such as colour), which represent shapes in 3D space as they are recorded by a LiDAR sensor. LiDAR stands for "Light Detection and Ranging" and functions by sending out laser beams in pulsed waves and measuring the time it takes for the reflected light to return to the receiver. Using distances, estimated through the return-delay, and directional information, a 3D location for the reflection point relative to the sensor can be extrapolated and stored in a point cloud. LiDAR and point clouds are a promising technology in the field of computer vision for which much research is being done. Figure 1 shows an example of point clouds generated by various objects.

Figure 1: Example Point Clouds (from http://mgadelha.me/mrt/)

Digital twins refer to a virtual representation that serves as a real-time digital counterpart of a physical object. Virtual models of real-life objects allow for opportunities in planning, conceptualizing, and monitoring that would have previously been comparably complex to execute, depending on the level of complexity and size, as well as the abstract nature of an object.

CAD catalogues simply refer to a collection of technical 3D models each of which represents some sort of technical component of a system. In the case of this project, the CAD catalogue would contain 3D models of the different components of the catenary arches, such as poles, insulators, etc.

1.2 Research Objectives

The goal of this project is to provide Strukton Rail with a working data pipeline prototype which takes point clouds (and optionally panorama images) of catenary arches as an input and returns corresponding (in scale, location, and orientation) 3D scenes containing models retrieved from a provided catalogue. The target audience of this project are the data engineers and scientists of Strukton Rail. The primary research question is the following:

RQ: How to leverage deep learning-based point cloud segmentation models for constructing railway scenes as a 3D model/digital twin?

The research project can be divided into the following two phases:

1. Development of deep-learning models for segmenting catenary systems from low-density point clouds.
2. Development of a data pipeline for converting point-cloud-based railway scenes to 3D models leveraging the existing CAD libraries for the objects in the catenary system.

1.3 Thesis Structure

In the following chapters, the full process of preparation, implementation, and evaluation will be documented. Chapter 2 will outline the background research, containing the state of the art for each phase of the project and other relevant literature on the topic. Chapter 3 and 4 will concern the methodologies applied in this project, and the project specifications respectively, while chapter 5 tells the story of the realization/implementation. In Chapter 6, the results of the project are evaluated. Chapter 7 will outline the conclusions to this project, and chapter 8 will consider the opportunities for future research based on this project. Visual representations of the data used, supplementary code displays, and an ethical assessment of the project can be found in the Appendices.

Chapter 2 – Background Research

The goal of this background research is the investigation of existing deep learning models and their theoretical applicability to the given problem, as well as the retrieval of relevant information on the state-of-the-art approaches to converting point clouds into 3D models.

2.1 Literature Research Questions

The following four research questions were investigated in the scope of the background research. The first three focus on deciding on a suitable model for the first step of the project, while the last one aims to identify the state-of-the-art for the second part of the project.

RQ1. Which deep learning methods are available for the segmentation of point clouds?

RQ2. Which segmentation models are most applicable to low-density point clouds?

RQ3. How do different point-cloud segmentation models perform in classifying large or small objects?

RQ4. Which methods can be used to convert point-cloud-based scenes to 3D models?

2.2 Methodology

This background research is undertaken as a systematic literature review, loosely following Kitchenham's methodology [5].

The search process was a manual search in the largest computer science database called ACM Digital Library [6]. Relevant papers cited by articles found in this database have been searched for using the Google Scholar search engine [7]. The following search terms have been used in the ACM Digital Library for the respective research questions:

1.1 point cloud segmentation

1.2 outdoor point cloud segmentation

2.1 low density point cloud segmentation

2.2 segmentation of sparse point clouds

3.1 large object segmentation in point clouds

3.2 small object segmentation in point clouds

4.1 converting point cloud to 3D model

4.2 3D model reconstruction from point clouds

Due to the broad scope of this background research, very few criteria need to be mentioned. For papers concerning technical implementations, age is the primary inclusion criteria. Therefore, no papers older than 5 years have been included for the first three questions concerning deep-learning models, and no papers older than 10 years have been included regarding the fourth research question concerning 3D model reconstruction.

2.3 Results

2.3.1 Deep-Learning Models for the Segmentation of Point Clouds

To consider a point cloud segmentation model's viability for the specific task of the underlying project, the overall category of segmentation tasks is discussed. Then the categories of deep learning models will be explored before the considered models will be evaluated for criteria relevant to the application throughout the first three question. These criteria include performance, complexity, and common usage. Performance refers to the model's comparable accuracy to others on commonly used datasets. Complexity considers a model's implementation complexity, as well as its computational efficiency for the sake of this project's limitation in time. Lastly, the common usage criteria considers how proven a model is in a variety of applications similar to this project, and therefore its likely reliability to perform to a satisfactory level. Beyond accuracy and task-taxonomy (segmentation, classification, etc.), no standard criteria for applicability of point cloud deep-learning models to real-world applications exists, which may represent an opportunity for future research.

Semantic segmentation methods are viable enough for the scope of this project. There are two major types of segmentation methods as described by Guo [8]: those attempting semantic segmentation, and those attempting instance segmentation. Semantic segmentation is what is classically understood as segmentation, the labelling of each point with a corresponding class to which the point is supposed to belong. Instance segmentation on the other hand aims at not only labelling each point with a class but distinguish between instances of the same class within a point cloud. Instance segmentation is beyond the scope of this project.

Point-based semantic segmentation methods are the most promising approach available for this project. Guo divided semantic segmentation methods into projection-based, discretization-based, hybrid, and point-based methods [8]. Projection-based methods generally include those that project the 3D point cloud onto a 2D surface and apply segmentation models on those 2D views of the scene. Discretization-based methods turn the point cloud into a more regular grid-based datatype via for example voxelization.

Then convolutional neural networks can be applied to the 3D pixel grid, which works similarly to 2D segmentation using CNNs. Hybrid methods generally try to combine 3D CNN and 2D CNN streams of the same scene. Lastly, the point-based methods are meant to perform segmentation directly on the points of the point cloud, which is a newer and more promising approach according to Bello [9], as projection-based methods lose information, while discretization methods may unnecessarily increase the volume of the data [10].

Within point-based methods, pointwise MLP (multi-layer perceptron), point-CNNS, and graph-based methods are the most promising. Guo [8] further separated point-based methods into Pointwise MLP networks, convolution-based networks, and graph-based networks. Pointwise MLPs are fully connected neural networks, meaning each neuron of a layer is connected to each neuron of the next layer. These methods generally lack the ability to capture wider context for each point and learn richer structures. Bello [9] categorized the methods by their ability to perform this local region computation, but also to recognize local correlations. Convolutional-based networks were applied as a per-point method instead of their classical grid-based counterparts to solve the issue of local correlations. Lastly, graph-based methods generate a graph from the point cloud, usually consisting of vertices and edges. Learning from this graph helps capture the underlying shapes and geometric structures. While pointwise MLPs pioneered point-based point cloud segmentation, CNNs and graph-based methods are promising to achieve a better understanding of the local geometric features.

PointNet++ is the current state-of-the-art for point cloud segmentation. PointNet [11] was the pioneering method that most point-based methods are loosely based upon but lacks the ability to capture the local context of each point [9]. Its successor PointNet++ improves upon this point by dividing the point cloud into nested partitions on which PointNet is applied recursively [12]. While it is expensive computationally and many newer methods may report better accuracies for specified tasks, its common usage for tasks like the underlying project make it a reliable option.

PointNet++ variants are promising, but unproven and lack an applicability guarantee for this project. A variety of modified networks have been developed on top of PointNet++, which promise better performances. For example, a graph-based aggregation module is reported to improve segmentation accuracy (measured in mean intersection over union) for indoor objects [13]. This project focuses on outdoor scenes, making this specific modification unnecessary. Meanwhile, OG-PointNet++ uses a method called octree-grouping to improve the partitioning process and the overall performance speed of the network [10]. MVPNet (MultiView PointNet++) means to enhance PointNet++ by supplementing it

with multiple 2D images of which features are lifted to 3D space and fusing complementary geometry and image information in canonical 3D space [14]. While some of these modifications appear promising, they present an increased difficulty in implementation and required data preparation, as well as a lack of guarantee that they are sufficiently applicable to the current project. Although not always the best performing model, PointNet++ is the most tested de facto state-of-the-art across most segmentation tasks [8] [9].

Point-convolution networks may record higher accuracies but are more complex and potentially inefficient. According to Bello [9], the three convolution-based networks A-CNN, RS-CNN, and SSCN had the highest performance for segmentation tasks. The Annular-CNN (A-CNN) model leverages neighborhood information to better capture local geometric features of 3D point clouds, and is supposedly computationally faster than PointNet++ [15]. The Relation-Shape CNN (RS-CNN) focuses on the geometric relations of the point cloud to achieve contextual shape-aware learning by extending the regular grid CNN to the irregular configuration of point clouds [16]. The Submanifold Sparse Convolutional Network (SSCN) is a CNN tailored to processing sparse data by strictly operating on submanifolds [17], which refers to subsets of data mirroring properties of its superset. According to Graham et al. [17], CNNs are generally inefficient when applied to typically sparse datatypes such as point clouds, which the SSCN may solve. Yet, in comparison to PointNet++, they are relatively untested on a wide range of applications.

Graph-based methods like the SPG model can be viable implementations for this project. The Super Point Graph model has been previously tested for catenary arches using high density point cloud data by Dijkstra [18]. The Super Point Graph model [19] partitions the point cloud into simple shapes called superpoints which are geometrically homogeneous elements. These graphs may contain rich contextual relationships which may help a deep-learning model to better understand the geometry and relations of a scene. Additionally, a ground-aware framework using an attention mechanism has been developed which improves the understanding of long-range dependencies between objects and ground-points [20]. While this may not be applicable to this project, in which ground-points are usually removed before training [18], its ground-detection module may allow for the automation of the ground removal step previously executed manually, which may be interesting for further research.

A direct comparison of the SPG [19] and PointNet++ has shown that the SPG model performed better on smaller objects while PointNet++ performed better on larger objects [18]. This may be largely because PointNet in essence is limited in its ability to recognize fine patterns and the versatility of complex scenes [10]. While PointNet++ addresses that problem, its grouping methods are still limited, especially

when dealing with sparse point clouds. The addition of octree grouping in OG-PointNet++ aims to fix that but did not report data on segmentation accuracies of differently sized objects [10]. Kim et al. [21] suggests that starting with larger easier to match objects such as long poles might be helpful in the segmentation of smaller, connected objects afterwards.

Additionally, the ground-aware attention model by Wu et al [20] seems to perform even better on small objects than the vanilla SPG. Yet, its focus on autonomous driving presents with a different definition for small objects, more so referring to people further away, represented by lower fidelity point clusters. In this project, smaller objects refer to physically smaller objects at the same distance, for example insulators vs long poles.

DeepSets are a promising but unviable option for this project. DeepSets [22] also perform machine learning directly on the point cloud input set. While not specifically designed for point clouds, its ability to learn from unordered sets efficiently via permutation invariant functions makes it an interesting model to explore. Its key-feature of permutation invariance is important for point clouds which are inherently unordered and structureless beyond its coordinates. Additionally, it is projected to perform faster than graph-convolution as used by for example the SPG [22]. Unfortunately, no semantic segmentation implementation of DeepSets is currently available for public use. Implementing the DeepSets model for this task from scratch is beyond the scope of this project.

Low and inconsistent densities are a common problem with point clouds, which is why most models are already equipped to deal with it to some degree. Yet, a few methods seem to report relatively good numbers on their lower density tests, namely the aforementioned ground-aware attention module for the SPG [20], OG-PointNet++ [10], the RS-CNN [16], and the SSCN [17]. No additional information is available on the performance of DeepSets on low density point clouds, which is a research gap potentially fillable in this project.

2.3.2 Methods for Converting Point Clouds to CAD Models

The two primary methods for converting point clouds to 3D models are template matching based on segmented point clouds, and direct conversion of the point cloud into a polygonal model [23]. Since this project is meant to use a provided catalogue of CAD models to reconstruct the scene, the latter approach is not of interest. Additionally, a panorama image-based approach to reconstructing 3D scenes exists.

PanoContext [24] can reconstruct room layouts and objects in 3D from panorama images. Although this approach is not point cloud based, panorama images are part of the supplementary data provided for this project. It was specifically designed for indoor scenes and primarily divides the room into rectangular boxes. Catenary arches are of very different geometry and exist within an outdoor scene. Therefore, the applicability of PanoContext for the reconstruction of the arches as 3D models would require further research.

Template matching approaches generally use deep learning methods to segment the point cloud before replacing each segmented object with a CAD model in the scene. Kim et al. [21] used PointNet and MVCNN and structured their pipeline into the following steps: 1. Segmenting the point cloud, 2. Assigning labels to the segments, 3. Selecting the corresponding catalogue, 4. Placing the 3D shape from the catalogue in the same position and orientation as those of the segmented point cloud. They also pointed out that that it can be helpful to first identify easy objects (in their case long pipes, in the case of this project likely long, straight poles) and placing them in 3D space to improve the recognition of commonly connected objects.

The state-of-the-art approach for template matching in point clouds is Point Pair Feature matching, short PPF. Vock et al. [25] proposes a RANSAC and PPF-based approach. RANSAC (random sample consensus) is a looping algorithm with which the parameters of a machine learning model can be estimated. PPF (point pair features) matching is a 3D object detection and pose estimation method that leverages the relational features between each two points. In their approach, they first determine edge points for improved geometric understanding, then they generate a PPF hash-map for the object templates. Next, they run the sampled point pairs through a RANSAC loop to match them with template pairs to determine pair correspondence and generate transformation hypotheses. These transformations allow an understanding of the orientation. Next, the transformation hypotheses are scored using a voxel distance field. Lastly, they fine-align the highest scoring match candidates with the scene point cloud via voxel based ICP (iterative closest point algorithm, which minimizes the difference between two clouds of points), and remove the corresponding scene points, effectively replacing it with the template model. Then the RANSAC loop is continued for the next object. This approach is comparably efficient according to the authors.

2.4 Conclusion

Of the four research questions investigated in this background research, the first three focused on determining the most suitable deep learning model for the segmentation of low-density points clouds

regarding this specific project, considering performance, complexity, and common usage. The final question sought to deliver an understanding of methods with which point clouds could be converted into 3D models.

Resulting from the findings for the literature research questions, the models and methods that will be applied in the respective phases of the project will be:

[1] Deep-learning models for the segmentation of point clouds

- PointNet++

[2] Method for converting point cloud scene to CAD model using CAD catalogue

- Point Pair Feature Matching in a RANSAC loop

2.5 Further Research

The following further research opportunities have been identified throughout the background research:

- Taxonomy of point cloud deep-learning models in regard to potential field of application.
- Implementation of DeepSets for semantic segmentation applications.
- Applicability of PanoContext for the segmentation or reconstruction of catenary arches from panorama images.

Chapter 3 - Methodology

In this chapter, the general development life cycle, tools, and methods will be introduced, and the specific technologies used throughout the project explained in additional detail where necessary.

The goal of converting a scanned scene point cloud of a railway scene to a 3D model may be divided into two major phases, making up the overall data pipeline: the deep learning-based segmentation phase, and the template replacement phase. Within the template replacement phase, the transformations necessary to place the template CAD model in the same position as the scene object will be estimated via template matching. Then the CAD model can be placed in the scene. This two-step process must be completed for each segmented object of the input point cloud. From this point onward, the discussion of methods, specifications, realization, and evaluation will be considered separately for the two phases. Figure 2 shows the two main phases and their relevance to the planned pipeline.

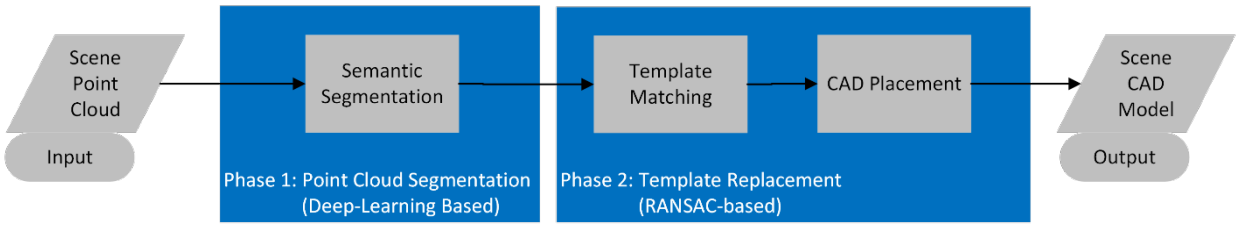

Figure 2: Project Phases and the Pipeline

3.1 CRISP-DM

Data science projects commonly follow the so-called CRISP-DM process, which stands for Cross Industry Standard Process for Data Mining [26]. It was developed as a European project to standardize the processes of data mining projects across industries. CRISP-DM entails six cyclical phases: business understanding, data understanding, data preparation, modeling, evaluation, and deployment. These phases will be applied in the specification phase of this project and serve as the underlying project architecture. Figure 3 shows these phases and the common cycle of progression of CRISP-DM.

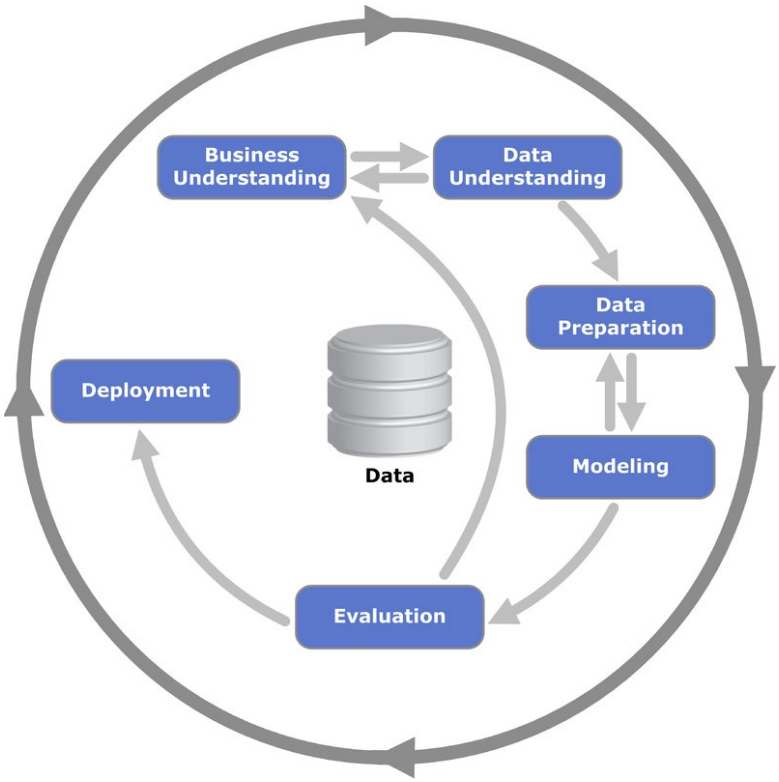

Figure 3: CRISP-DM, By Kenneth Jensen - Own work based on: ftp://public.dhe.ibm.com/software/analytics/spss/documentation/modeler/18.0/en/ModelerCRISPDM.pdf (Figure 1), CC BY-SA 3.0

In the first phase, called business understanding, the objectives and requirements of the project are to be specified. This includes the definition of success criteria, and assessment of available resources as well as risks and contingencies. In a commercial project, a cost-benefit analysis may take place at this stage, but this is not applicable at this stage, as will be explained later.

The second is the data understanding phase, during which the identification, collection, and analysis of data sets that may be useful for accomplishing the goal takes place. This may include the description of data via their surface properties like formats, row counts, and field identities, but also initial visualizations to identify relationships or preparation requirements.

The third phase is the data preparation phase. In this phase, useful data will be selected, cleaned, and if necessary sampled, constructed, integrated, or formatted to fit the input requirements of the model.

The fourth and fifth phase are the modeling and evaluation phases. During modeling, the algorithms are chosen and implemented, a test design generated, and the model built. In the case of the second phase of this project, no actual machine learning algorithm is trained. Therefore, this step will

primarily consist of implementation and testing for the second phase. In the evaluation step, the performance of these algorithms in consideration of the goal of the project will be considered. If found unsatisfactory, the cycle may be repeated, and other models and data sets applied.

In the case of satisfactory results in the evaluation phase, one might continue to the sixth phase: deployment. During this phase, deployment and maintenance of the project's outcome is planned, a final report produced, and the project reviewed.

This project contains at least two partial cycles, one for each of the two project phases, as they are considered as two separate stages with individual evaluation procedures. Using a standardized process such as CRISP-DM makes it not only easy to track and organize the progress throughout the process, but also to communicate it easily with supervisors and clients.

3.2 Development Environments

For the technical implementations necessary for this project Python was used as the primary development language in an Anaconda development environment installed on a server of the Saxion University of Applied Sciences. Anaconda is used primarily for ease of setup of the development environment and installations of python dependencies. For deep learning implementations, TensorFlow [27] was used, as the original implementation of the chosen PointNet++ model is part of TensorFlow [27].

3.3 Methodology for Deep-Learning-based Point Cloud Segmentation (Phase 1)

3.3.1 General Deep-Learning Techniques

Chapter 1 already introduced deep learning as a category of machine learning which utilizes artificial neural networks organized in multiple layers from the input layer, the intermediate hidden layers, and the output layer, containing the model's prediction for a given input in the form of confidence values (percentages). Each neuron is represented by a value container, which will receive its value from various, if not all, neurons' values multiplied by a weight value which is unique for each neuron-to-neuron connection. A bias is also subtracted from each of the values and serves as a minimum value for "the neural connection to activate" (to be larger than zero and therefore affect the neuron in the next layer at all). Commonly, a sigmoid function or ReLu (Rectified Linear Unit function), which is as of 2017 the most popular activation function for deep neural networks [28], is used as an activation function to determine the output of each neural connection. Using this function, values are passed from each input node to each output node through the various hidden layers of neurons.

Generally, this process represents decision making in feedforward (meaning one-directional) neural networks. During training, the output for each input can be compared to the manually provided output, and the weights and biases towards each output revised through a process called backpropagation. In this process, the weights and biases between neurons are adapted repeatedly. The trained model then consists of these iteratively adapted weights. After training, a model can be evaluated based on how accurately it predicts outputs for an input data set, for which manually prepared outputs are also available. Therefore, before training, the data original dataset is usually split into a training set and a testing data set, so that the model will not be tested on exactly the same data that it has been trained on. This is also important to detect a common problem in deep learning referred to as overfitting of a model, which occurs when the training data is too specific, and the model is not able to identify general patterns based on it.

3.3.2 Point Cloud Semantic Segmentation

Point clouds as a format have already been thoroughly explained in chapter 1.1 and semantic segmentation was introduced in chapter 2. At this point, it is worth taking a look at the tools and methods with which point clouds can be prepared for the task of training a semantic segmentation model. Data preparation will commonly consist of the following steps: cleaning, labelling, down-sampling, splitting, and augmenting.

During cleaning, irrelevant or false data will be removed from the data set, and during labelling, a class identifier will be added to each point of the point cloud which corresponds to the object that the point is a part of. Both of these tasks can be done manually using a software such as CloudCompare [29], which is the program predominantly used to create, edit, and visualize point clouds throughout this project.

Down-sampling can be achieved with a variety of simple methods such as random down-sampling (randomly select x amount of points from point cloud), uniform down-sampling (every x-points from the list), voxel-based down-sampling (only one point per cube in a 3D grid with cube volume x^3), but also via more sophisticated methods such as spectral decomposition [30] or adaptive hierarchical down-sampling [31]. According to a report by a M. Salahmand while working for AMI [32], data sets sampled via voxel down-sampling performed better than the of other tested sampling methods. Only the accuracy for particularly small objects suffered in the case of voxel-down-sampling. This deficit may be circumventable by reducing the cube volume x^3 of the voxel grid based on the bounding box diameter of the object point

cloud. Additionally, voxel-based down-sampling algorithms are easily implementable with the use of python libraries such as open3d [33]. Therefore, a voxel-based method will initially be used for down-sampling in this project.

Data splitting refers to the process of dividing a labeled dataset into two sets: a larger one for training, and a smaller one for testing, based on proportionality parameters.

Lastly, data augmentation is the process of creating additional data sets for training by transforming existing data sets in various ways. A student group working at AMI developed various augmentation methods and tested the resulting datasets for performance in point cloud segmentation [34]. The resulting methods will be applied in this project as well.

3.3.3 Point Cloud Segmentation Evaluation

The most commonly used method for evaluating machine learning projects is to split the dataset into a training and a testing set, commonly at 80/20 or 70/30 proportions. The larger training set is then used to train the model and the small testing set is used to evaluate the performance of the model by comparing the labels predicted by the models to those manually assigned beforehand.

A slightly different method, which has been specifically designed for datasets with a significantly lower data size, is called “leave-one-out-cross-validation”. In LOOCV, the learning algorithm is applied iteratively for each instance of the data set, which in this case refers to one labeled catenary arch. On each iterations, the model is trained using all other instances and then evaluated on the arch left out. This method is especially useful when the model is supposed to be able to make predictions on data that is not used to train the model, for example future scans of catenary arches. It is primarily useful for small datasets as it is computationally more expensive than the traditional method of splitting ahead of training. Additionally, it is effective at reducing the overfitting effect, as a model that performs well with LOOCV is able to predict for data it hasn’t seen before. Overfitting refers to a model being trained on too similar or too small of a dataset which causes it to primarily adapt to very specific patterns which are not useful for the more general predictions it is supposed to make.

A common heuristic measure for the evaluation of point cloud segmentations is the so called “mean intersection over union”, short mIoU. It refers to the average intersection of points with the manual and predicted labels divided by the total number of points that have the label in the manual and predicted set (excluding doubles). This way, the heuristic not only accounts for points it mistakenly did

not label, but also for points that it mistakenly did label. It is commonly expressed as a value between 0 and 1, or as percent, with a higher value signifying a higher model accuracy.

3.4 Methods for Scene Reconstruction (Phase 2)

3.4.1 RANSAC-based Template Matching

Chapter 2 introduced point pair features in a RANSAC loop as a highly efficient template matching method. At this point, these methods will be elaborated further.

RANSAC, or Random Sample Consensus, is a method for estimating a model's parameters via repetitive identification of outliers and estimation of the model after removal of these outliers. In essence, a RANSAC loop consists of three internal steps as shown in Figure 4: sampling, hypothesizing, and verifying. In the sampling step, data points are randomly sampled. Using these data points, a hypothesis of the model's parameters is created. In the last step, the hypothesis is scored, and compared to previous hypotheses and their scores. Of course, RANSAC does not guarantee the successful generation of an optimal or correct hypothesis due to its random sampling. By increasing the number of iterations of the loop, the probability to find a suitable hypothesis increases.

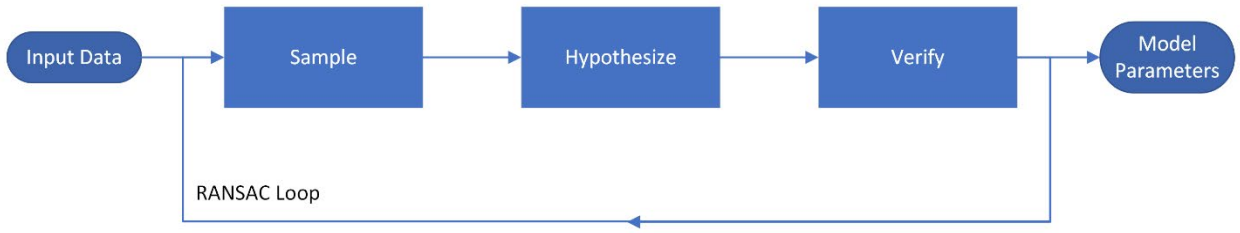

Figure 4: RANSAC Loop Concept

Point Pair Features are a commonly used method for object detection applications in 3D space. It utilizes the relations between two sampled points from a scene point cloud and compares them with the features of two points from a corresponding template. If the features match, the scene point pair is likely to belong to an object corresponding to the template. Features may include distances between the points, angles between the distance vector between the two points and their normal vectors, and the principal curvatures, among others.

Point pairs can be efficiently matched by indexing them using their hashed feature vectors. MurmurHash3 has been recommended for this task by Vock et al. [25]. Additionally, a library called mmh3

[35] is available to easily implement this function. Figure 5 shows the input and output of such a hash function in the context of point pair feature matching.

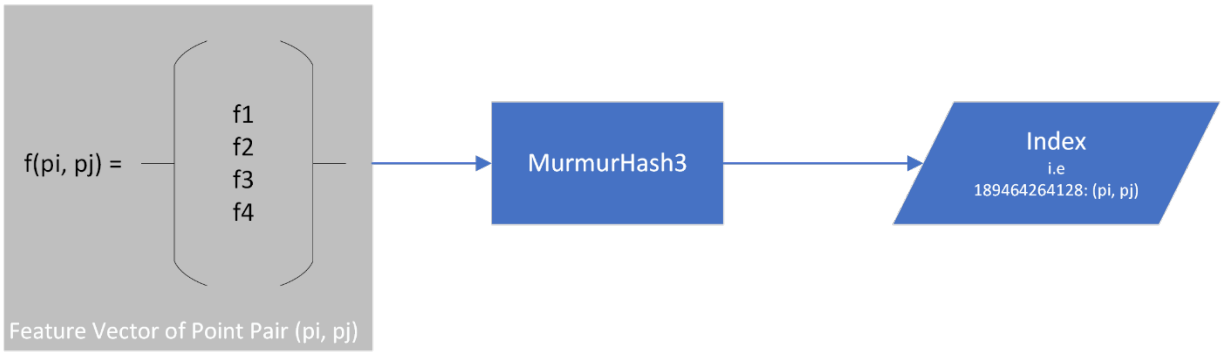

Figure 5: Point Pair Features Index Hashing

Linear algebra concepts can be applied to calculate the necessary transformation from a template point pair to its matched scene point pair.

Lastly, to score each hypothesis, a simple point region overlap algorithm can be applied, which will check the immediate neighborhood of each point of the transformed template for the presence of a point of the scene point cloud. The fidelity of this scoring system therefore heavily depends on the neighborhood-radius parameter. Alternatively, voxel-based overlap scores or graph-based scoring could be applied.

3.4.2 Methods for CAD Placement

There are a variety of CAD programs which could be used to manually build the final 3D models using the transformation data optimally obtained via template matching, but CloudCompare [29], despite it not being a CAD program, and FreeCAD [36] have been identifies as the two main methods applicable in this project.

CloudCompare requires the CAD models to be manually loaded in as .obj files. They can then be repositioned using CloudCompare's "Apply Transformation Matrix" function. This has the advantage of being able to directly compare the final CAD model to the point clouds as both point clouds and .obj files can be loaded and displayed in the same scene.

FreeCAD [36] comes with a Python API, which might enable the automation of the CAD placement step. Scene documents could be automatically created, objects loaded, and transformed via the obtained transformation matrices.

Chapter 4 – Specification

4.1 Business Understanding

4.1.1 Objectives

The precise purpose of this project is the creation of a prototype for a data pipeline capable of reconstructing 3D models from point clouds depicting catenary arches. It is part of the larger digitalization effort by Strukton Rail [2] for which they have contracted the Ambient Intelligence research group of the Saxion University of Applied Sciences to develop deep-learning models for the classification and segmentation of point clouds for catenary systems. Figure 6 demonstrates how this specific project fits into the overall work of AMI for Strukton. The green marked parts are those that are newly implemented in this project.

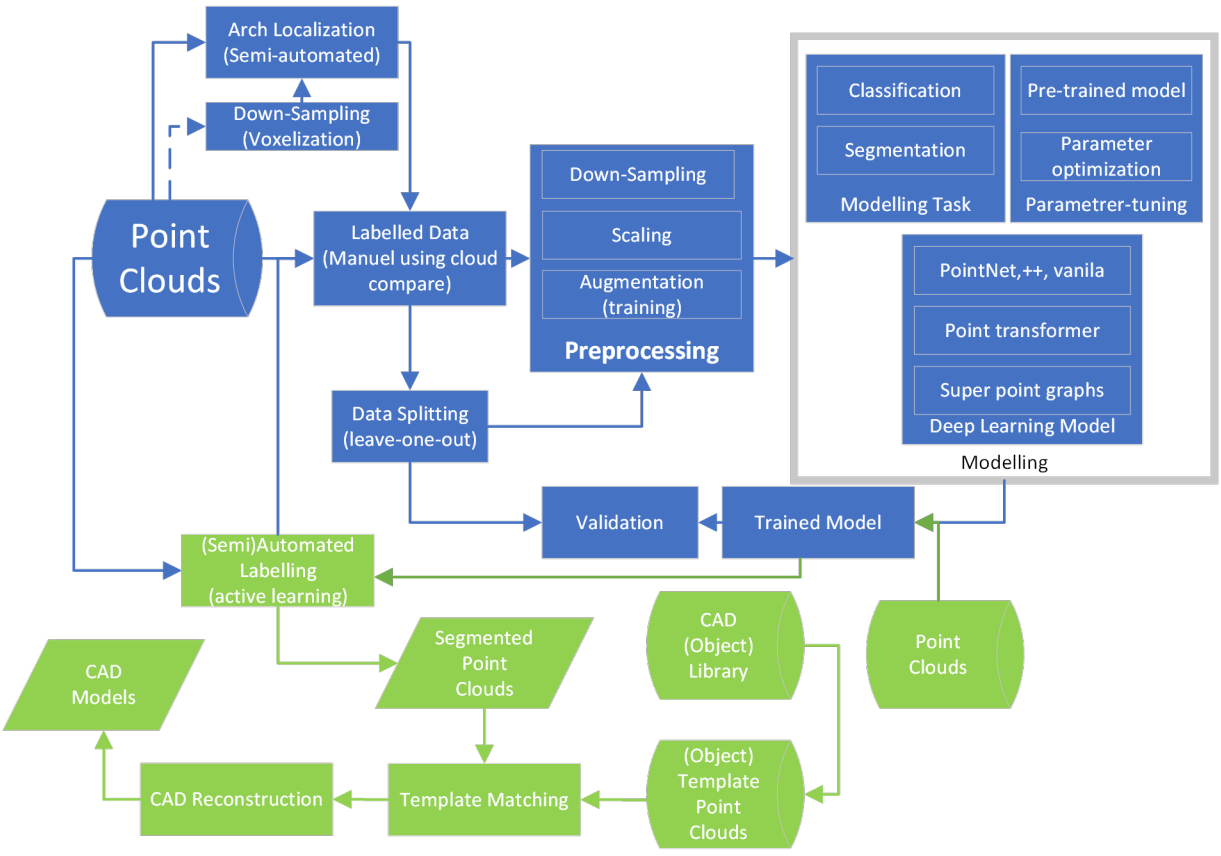

Figure 6: Relation between AMI (blue) and my (green) project.

This project can be considered successful, if the pipeline prototype shows successful results for at least two structurally different components of the catenary arches.

4.1.2 Resources

Strukton provided a dataset of 16 catenary arches for this project, of which 13 were deemed usable and have been preprocessed and labeled by AMI interns and researchers in previous projects. Additionally, tested data augmentation methods and a modified PointNet++ algorithm is provided by the researchers at AMI, which can be implemented for this project. Additionally, an archive of catenary arch component CAD models has been provided by Strukton.

A server with extensive computational power and the anaconda software is provided by the research group for testing and running of code. Lastly, the supervisors and researchers at AMI offered their assistance for the completion of this project in the form of technical experience and know-how.

4.2 Data Understanding

4.2.1 Point Clouds of Catenary Arches

The data set provided for this project consists of 13 labeled catenary arches. They are stored in the “las/laz” format, which is the industry standard for archiving LiDAR scanned point clouds. They have all been manually labelled. Figure 8 shows an unlabeled catenary arch point cloud and a labelled one, where each color represents a label corresponding with a component type. It also must be noted that the catenary arches can differ severely in structure and attached components as can be seen in Figure 7, which shows a direct comparison of three arches with different components, structures, and environment points left in the dataset. A visualization of all 14 arches can be found in Appendix C. These point cloud arches have an average of approximately 4 million points per arch (3,942,666), amounting to over 50 million points in total. While this is a large amount of data in pure physical storage space required, these points only represent 13 arches, which is very little to train a solid model.

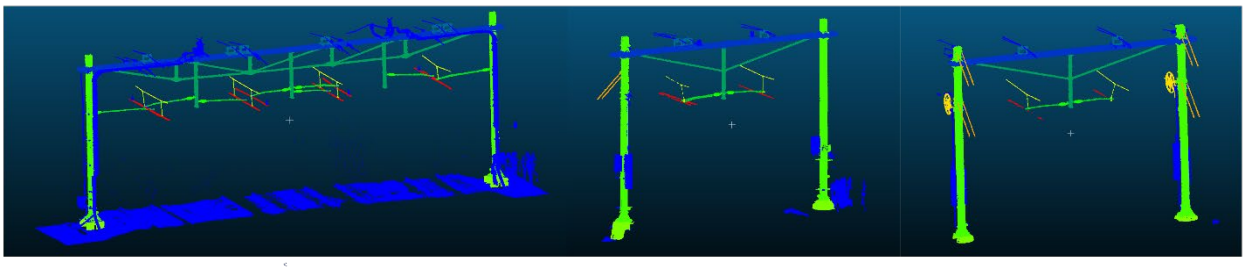

Figure 7: Different types of Catenary Arches from Strukton's dataset.

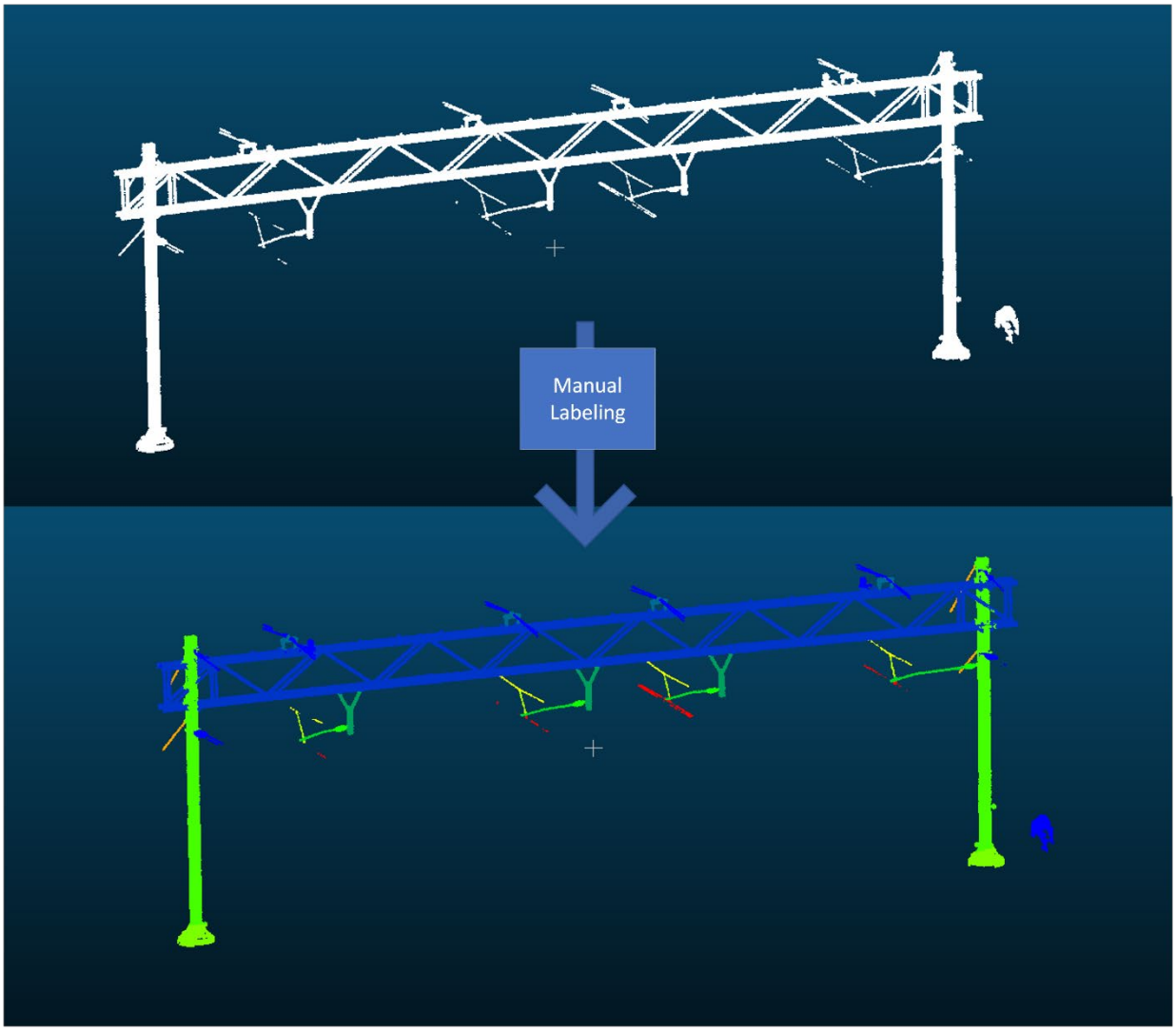

Figure 8: Manual Segmentation via Labelling.

The following labels are available in the data set, each, with the exception of “Unlabeled”, corresponding to a component type commonly found on catenary arches. It has to be noted that the real-life components corresponding with these labels come in many different versions and sizes, which puts additional emphasis on the deep-learning model for segmentation and the data pipeline as a whole being a prototype.

ID	Label/Component	Colour Name	RGB	CAD Available
0	Unlabeled	White	255, 255, 255	-
1	Portal	Orange	255, 95, 0	No
2	Messenger Wire Support	Yellow	255, 255, 0	Yes

3	Drop Post	Red	255, 0, 0	No
4	Steady Arm	Green	0, 255, 0	Inconsistent Shapes
5	Insulator	Fuchsia	255, 0, 255	Yes
6	Pole	Blue	0, 0, 255	Yes
7	Pole Foundation	Acqua	0, 255, 255	Not Isolated
8	Dropper	Light Yellow	255, 255, 135	No
9	Stitch Wire	Pink	255, 175, 255	No
10	Wheel Tensioning Device	Light Blue	215, 255, 255	No
11	Tension Rods	Dark Green	135, 175, 0	No
13	Tension Rods Foundation	Blueish	135, 175, 255	No
14	Contact Wires	-	-	-
15	Top Tie (single arch top bar)	-	-	-
16	Bracket (45 deg bar, single arch)	-	-	-

Table 1: List of Catenary Components and their representative properties.

4.2.2 CAD Catalogue

Strukton also provided a catalogue of 59 CAD models in the “stp” format, also referred to as STEP-files. STEP is a “manufacturer-independent format for describing computer-generated 3D models” (www.gom.com). Unfortunately, these 59 files do not directly correspond to each of the labeled components. In fact, multiple labels are not present in this archive at all, while for some labels, up to 18 different CAD models are available. An overview of which labels have corresponding CAD models available can be found in Table 1, and the chosen models for each label can be viewed in Appendix D. Figure 9 shows one of the CAD models in isolation when opened using CAD software such as FreeCAD [36].

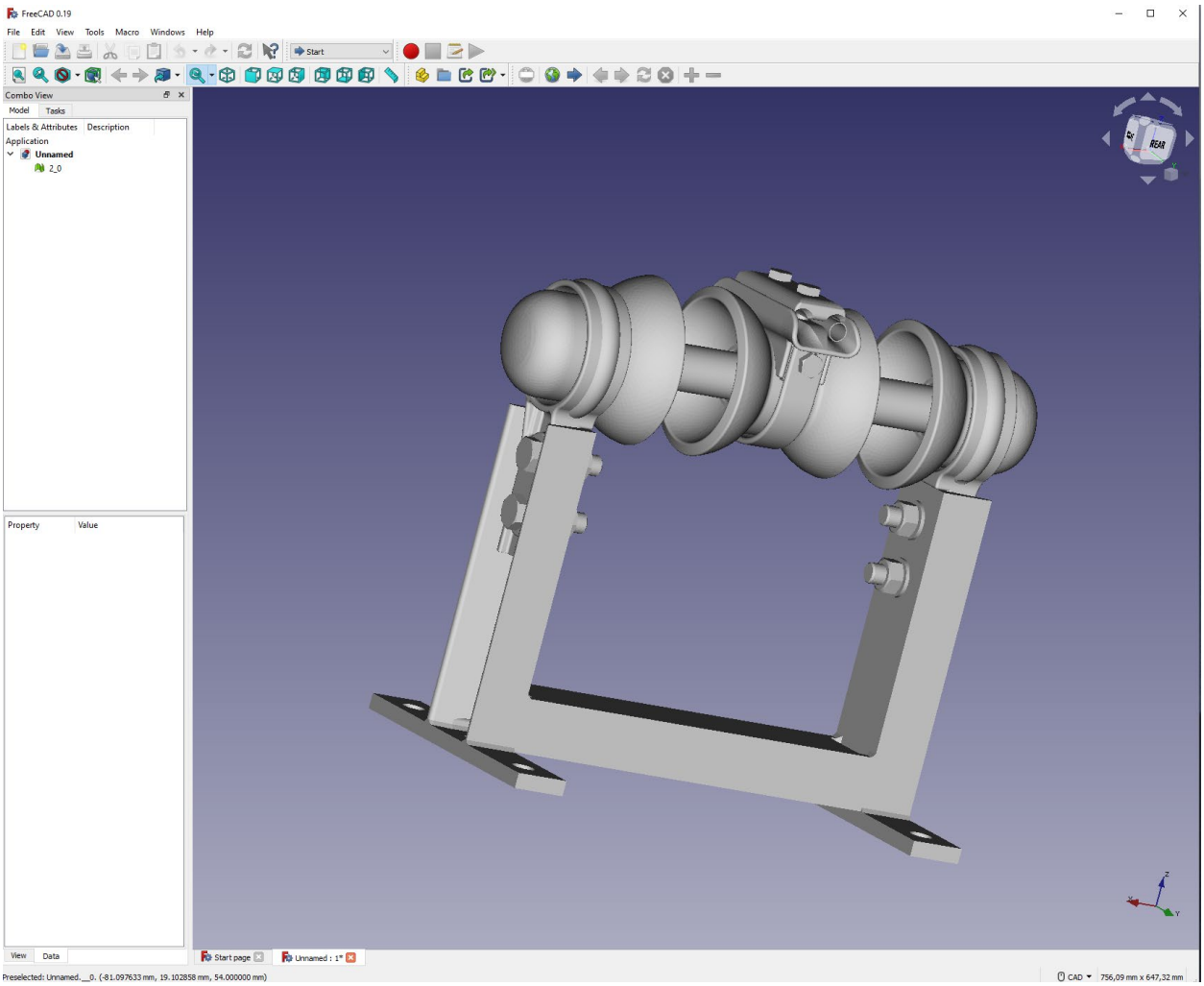

Figure 9: Messenger Wire Support CAD Model in FreeCAD.

4.3 Pipeline Architecture

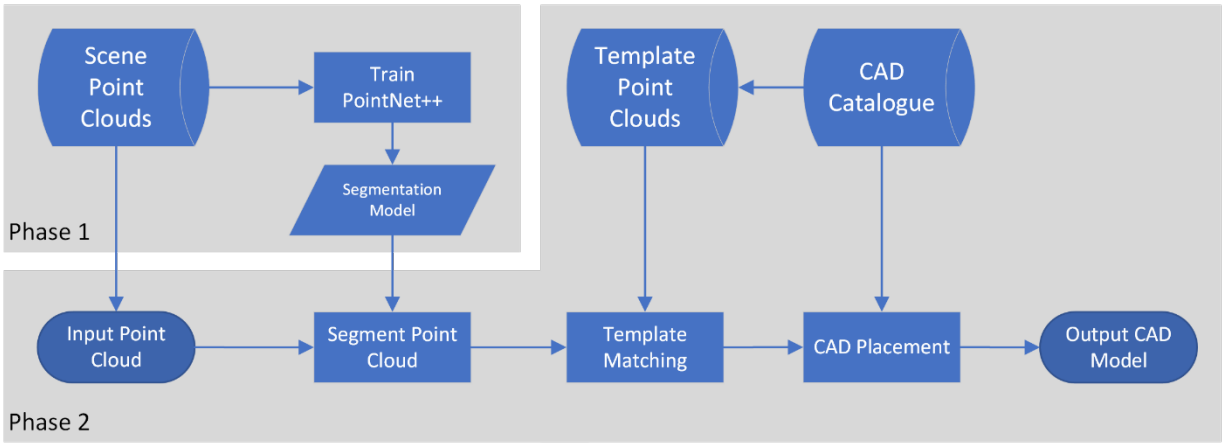

Figure 10: Planned Pipeline Architecture

4.3.1 Planned Pipeline

Figure 10 shows the broad pipeline layout as is planned for the implementation. During phase 1, a semantic segmentation model will be trained using PointNet++, which can then be used to segment a LiDAR scanned point cloud of a catenary arch. Once segmented, each identified component can be matched against template point clouds which will be generated from the CAD models provided in the catalogue. This matching will result in a verified transformation hypothesis, which can then be used to place the actual CAD model into the scene. Doing this for all segmented components will result in one CAD model containing the entire arch, in theory.

4.3.2 Expected Limitations

There are a multitude of certain and possible limitations with this project design. First of all, if the segmentation model fails to perform at acceptable accuracies, phase 2 will have to commence with the manually labelled point clouds instead. Secondly, there aren't CAD models available for all the components the segmentation model may be trained to identify. These will have to be skipped, which means that this prototype will in no case be able to output a fully reconstructed catenary arch CAD model. Lastly, the PointNet++ segmentation model is known to be significantly more accurate when segmenting larger objects such as poles, and much less accurate when segmenting small objects like insulators [18]. Since the segmentation step is the first in the reconstruction pipeline, it is likely that the pipeline will perform better on replacing large objects than small ones as well.

4.4 Phase 1: Training a Point Cloud Segmentation Model

The goal of phase one is to generate a semantic segmentation model capable of segmenting the components of catenary arches well enough so that its output may be used for the accurate replacement of each component with a CAD model. How to define well enough is very ambiguous when considering the scope of this phase by itself. The model itself can be evaluated using the mIoU heuristic explained in chapter 3.3.3. At this point in time, it is unclear whether the template matching process of phase 2 will perform perfectly fine or won't work at all with low accuracies in the segmentation model.

The following figure 11 shows the flowchart for phase 1, with three CRISP-DM steps color-coded.

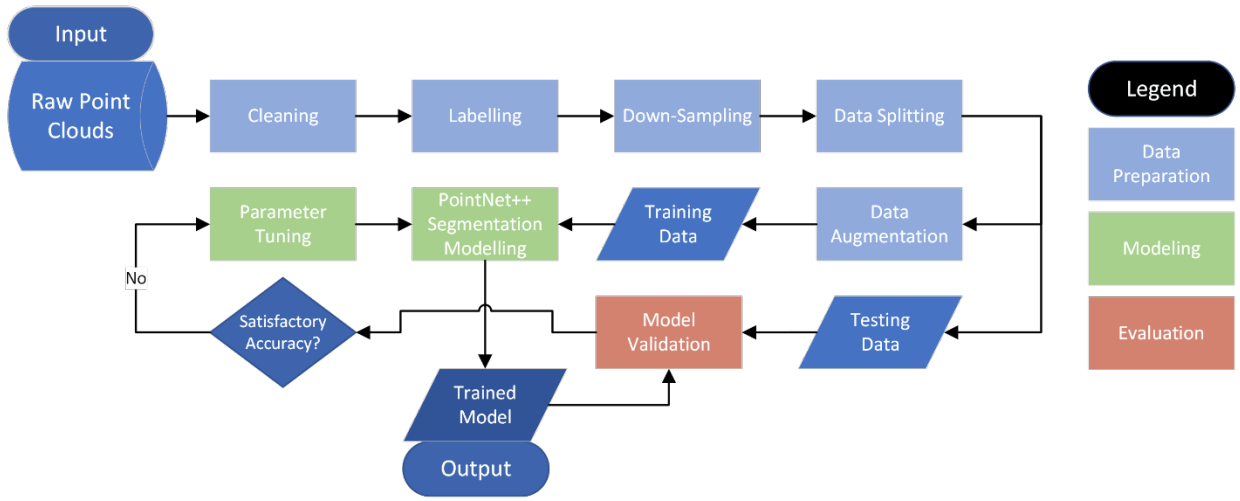

Figure 11: Generalized pipeline for the training of a deep learning-based segmentation model.

4.4.1 Data Preparation for Point Cloud Segmentation

The data preparation step has been mostly completed ahead of this project. A student group cleaned and labeled all the arches [34], and a script for data splitting has been provided by a researcher of AMI. Similarly, data augmentation [34] and down-sampling [32] methods have been implemented for the use with a PointNet++ model. At this point it is important to note, that PointNet++ takes a specific number of points as input, equaling 2^n, where a lower n is likely to perform worse [11], but is computationally faster.

Additionally, the PointNet++ model provided by the researchers at AMI does not take raw laz files as input, but instead an hdf5, which serves as an aggregate file storing the entire training and testing sets. A script is available to load the laz files into this format.

4.4.2 Modeling: PointNet++

As so often in data science projects, the modeling step simply consists of running the training command for the PointNet++ model with the previously created training set as input. Parameter tuning is not an expected step in this stage, as the model itself has been previously tuned for this input dataset by the researchers at AMI.

4.4.3 Segmentation Evaluation

The used implementation of PointNet++ uses the leave-one-out-cross-validation methodology for training, introduced in chapter 3.3.3, meaning that instead of splitting the entire dataset prior to training, the model is trained iteratively with one arch being left out of the training set on each iteration. This arch is then used to evaluate this iteration's results. Since the dataset used in this project is relatively small for a deep learning project, this method is the most suitable, despite its computational cost.

The model's accuracies for each component are computed via the mIoU heuristic introduced in chapter 3.3.3. Unexpected results are to be investigated for errors in the implementation. Otherwise, the resulting accuracies are to be reported as part of the prototype pipeline's documentation, and the model itself can then be used to segment point clouds of catenary arches for the reconstruction of CAD models in phase 2.

4.5 Phase 2: Reconstruction of Point Cloud Scenes as CAD Models

The goal of phase two is the development of the actual pipeline prototype for the reconstruction of CAD models from point cloud-based catenary scenes.

As input, the pipeline should take any unlabeled point cloud depicting a catenary arch, and the output should be a CAD model representing the input arch. No hard requirements for the format of the output CAD model have been set. Given that the CAD catalogue contains CAD models in the STEP-format, the output should preferably be in the same format to avoid unnecessary conversions.

The following figure 12 shows the intended pipeline with added detail in the template matching, and the three CRISP-DM phases color-coded:

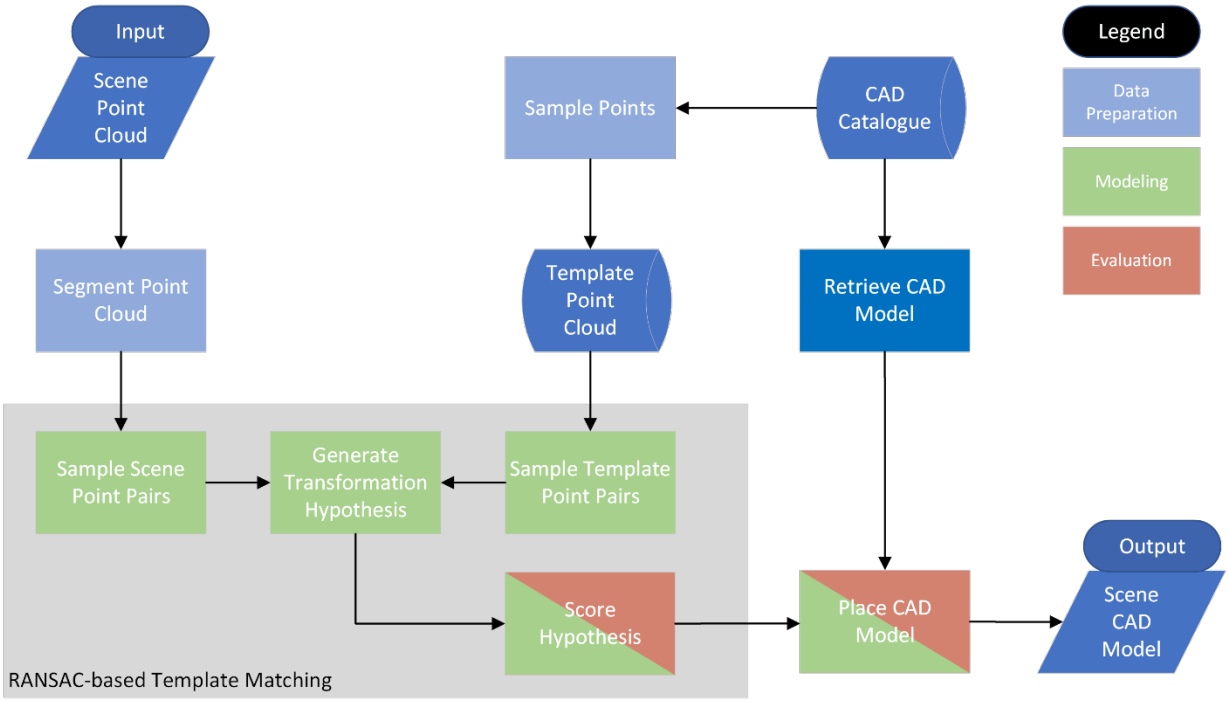

Figure 12: Projected Pipeline for the RANSAC-based template matching and CAD reconstruction.

4.5.1 Data Preparation for Template Matching

The first step in this phase's data preparation process is the collection of applicable CAD models from the archive provided by Strukton. The models are not labeled in correspondence with the labels used for the segmentation model, so they must be manually compared to the scanned and labeled point clouds. One CAD model should be selected for each labeled component. Since the segmentation model does not currently differentiate between different types of the same component, there is no need to implement type specific CAD model retrieval for the scope of this project.

Once a model for each labeled component, if available, has been selected, they must be converted into point clouds to serve as templates in the template matching process. CloudCompare [29] can be used for this process.

Then, input point clouds must be automatically labeled by the previously trained semantic segmentation model. Although this may not truly be considered a data preparation step as it occurs after the input in the pipeline, the modeling of this CRISP-DM iteration is meant to focus on the CAD reconstruction part of the pipeline, which is why it is included here.

Lastly, the segmented component point clouds and the template point clouds should be sampled using the same parameters to decrease the random sparsity factor of the input point cloud. In chapter 3.3.2 I discussed using the voxel-based sampling method for this purpose.

4.5.2 Modeling: Template Replacement

The modeling of phase 2 consists of the development of a RANSAC-based template matching approach capable of estimating template-to-scene transformations for the placement of CAD models into the scene. The various methods that should be used to implement this RANSAC-loop have been explained previously in chapter 3.4.1.

The feature vector suggested by Vock et al. [25] for template matching consists of the following:

More specifically given $(p_i, p_j) \in \mathcal{E} \times \mathcal{E}$ with corresponding edge directions $t(p_i), t(p_j) \in \mathbb{R}^3$ we compute the feature vector

$$f(p_i, p_j) = \begin{pmatrix} f_1(p_i, p_j) \\ f_2(p_i, p_j) \\ f_3(p_i, p_j) \\ f_4(p_i, p_j) \end{pmatrix} = \begin{pmatrix} \|p_j - p_i\|_2 \\ \angle(p_j - p_i, t(p_i)) \\ \angle(p_j - p_i, t(p_j)) \\ \kappa_{\min}/\kappa_{\max} \end{pmatrix},$$

where $\kappa_{\min}$ and $\kappa_{\max}$ are the principal curvatures at p_i and

$$\angle(x, y) := \tan^{-1}\left(\frac{\|x \times y\|_2}{|\langle x, y \rangle|}\right) \quad \forall x, y \in \mathbb{R}^3,\ x, y \neq 0,$$

Figure 13: Point Pair Feature Vector and Equations, taken from Vock et al. [25].

The feature vector suggested consists of the following four values: 1. the absolute distance between the two points; 2. and 3. the two angles between the directional vector between the points and each point's facing direction (normal vector); and 4. the principal curvatures at one point. The principal curvatures essentially measure the bending of the shape at the given point, and are not as essential for the functionality of the point pair features as the first three, according to Vock et al [25].

Additionally, Vock et al. [25] provides the formulas necessary for the calculation of transformation matrices from two matching point pairs. Vector and matrix math is efficiently implementable even on large data sizes using the numpy [37] library for python.

$$u_{ij} = \frac{p_j - p_i}{\|p_j - p_i\|_2}, \quad v_{ij} = \frac{v'_{ij}}{\|v'_{ij}\|_2},$$

$$v'_{ij} = (\mathbb{1} - u_{ij}u_{ij}^T)\ t(p_i),$$

where $\mathbb{1} \in \mathbb{R}^{3\times3}$ is the identity matrix and

$$R_{ij} = \begin{pmatrix} u_{ij} & v_{ij} & (u_{ij} \times v_{ij}) \end{pmatrix} \in \mathbb{R}^{3\times3}$$

$$T_{ij,kl} = \begin{pmatrix} R_{kl}R_{ij}^T & p_k - R_{kl}R_{ij}^T p_i \\ 0 & 1 \end{pmatrix} \in \mathbb{R}^{4\times4}$$

Figure 14: Equations for the calculation of transformation matrices from two feature-matched point pairs. Taken from Vock et al. [25].

By calculating R_{ij} and R_{kl} using the point pairs from the scene and the template respectively, we can calculate a 4x4 transformation matrix which includes rotation, scaling, and transformation. Scaling and rotation transformations are part of the $R_{kl}R^T_{ij}$, while the vertical vector p_k-$R_{kl}R^T_{ij}p_i$ corresponds to the translation.

Once a transformation hypothesis is calculated it should be scorable by applying it to the template and comparing the transformed template's points to the points of the scene object. The more points of the transformed template overlap with the scene component, the better must the transformation hypothesis be in theory. To rate the level of overlap, write an algorithm which checks the direct neighborhood of a certain radius of each template point for any existing scene component points. If a scene component point is in the neighborhood of the template point, this point is considered to overlap well. The fidelity of this algorithm would depend heavily on the neighborhood threshold, so different values should be tested. Theoretically, the mIoU principle could be applied to this as well by also adding all scene points which are not within the neighborhood of a template point to the denominator in the percent calculation. This is expected not to be beneficial in this case though, as it would cause mistakes by the semantic segmentation model to heavily deflate the scores of the template, if points outside the real shape of a component are falsely labeled as part of that component.

After obtaining an optimal transformation matrix for each segmented object, the respective CAD model can be placed into a scene manually using CloudCompare [29]. Then the transformation matrix can

be directly applied to the CAD model and the meshes can be saves as a single scene. The automation of this step via the FreeCAD [36] API is not considered a necessity for the success of the prototype pipeline.

4.5.3 Evaluation of Template Matching

The evaluation of this phase takes places primarily via the point-region-overlap transformation-scoring heuristic introduced in chapter 3.4.1, but the output of the pipeline should also be very easily visually evaluable without an additional heuristic. Based on how well the CAD model visually represents the input scene point cloud one can approximately estimate the success of the project. Yet, the point region overlap scores for each tested component should be reported as the official evaluation.

Chapter 5 - Realization

5.1 Phase 1: Implementing PointNet++ for the Segmentation of Point Clouds

The main purpose of phase 1 of this project is the implementation of an existing deep-learning model. To succeed in this task, I had to setup a development environment, install all the necessary dependencies of the model, load and prepare the dataset, and finally write code to run the model and evaluate it.

5.1.1 Setting Up the Development Environment

The Saxion University of Applied Sciences provided me with an Anaconda Jupyter Hub account on their server as my primary development environment. The machine that this Anaconda environment runs on had python 3.8 already installed. I created a fresh virtual environment with this python installation to start off.

Since the PointNet++ implementation I'd be using for this project is based in TensorFlow [27], I began with the installation of TensorFlow. To do this, I followed the instructions of the PointNet++ TensorFlow GitHub. The only difficult step in this process was the compilation of C-based dependencies using CMake [38]. The complications mostly stemmed from undocumented version incompatibilities. After researching the various errors and eliminating them over the course of a week, TensorFlow was correctly installed with GPU-support using CUDA [38]. CUDA was already installed on the machine in two different versions, which complicated the correct installation of the TensorFlow version I'd need for the correct PointNet++ implementation. CUDA [38] is a parallel computing platform developed by Nvidia which allows programs to use the GPU for extensive computing operations, such as training a deep-learning model.

5.1.2 Preparing the Dataset

The dataset had already been compiled and prepared for training using the PointNet++ model. It consisted of 11 laz files containing one catenary arch each. AMI provided me with shell and python scripts to down-sample the catenary arches for the correct input size of PointNet++ of 131072 points, and augment it to generate a larger dataset, as well as to compile it into an hdf5 file, which serves as a composite file that can be more easily loaded into the training model via the SaxionDataset python class provided to me.

Multiple difficulties surfaced during this process. Primarily, the python scripts provided were written using the deprecated pylas [39] library, which I had to replace with its successor laspy [40] library. The reason for this change being necessary is not exactly clear to me at this point, as multiple function calls in the original script seemed to correspond to a mixture of laspy and pylas implementations. This is most likely due to a difference in the setup of the virtual environment between the original writer of the script and mine. By working with the laspy documentation, I debugged the code to work with my version of laspy until the errors disappeared. Additionally, a few data and dependency paths had to be corrected, which was not particularly difficult.

5.1.3 Running the Model

After compiling the dataset to train the segmentation model, I began writing the code to train the model. Since there would be little code for me to write, I simply adapted the existing training code provided to me and altered it where necessary. Once again, I had to change the input and output paths, and fix dependencies. Then I ran the model.

At this point, I had to go back to work out some version incompatibilities between CUDA [38] and TensorFlow [27] as mentioned previously. After finding a workaround to the version problem using linux' syslink function to create system links to the necessary CUDA file in the places that TensorFlow required it, I could finally train the model successfully. The trained weights of the neural net were saved as a pickle binary dump in my data folders. Figure 15 shows some of the console output while training the model.

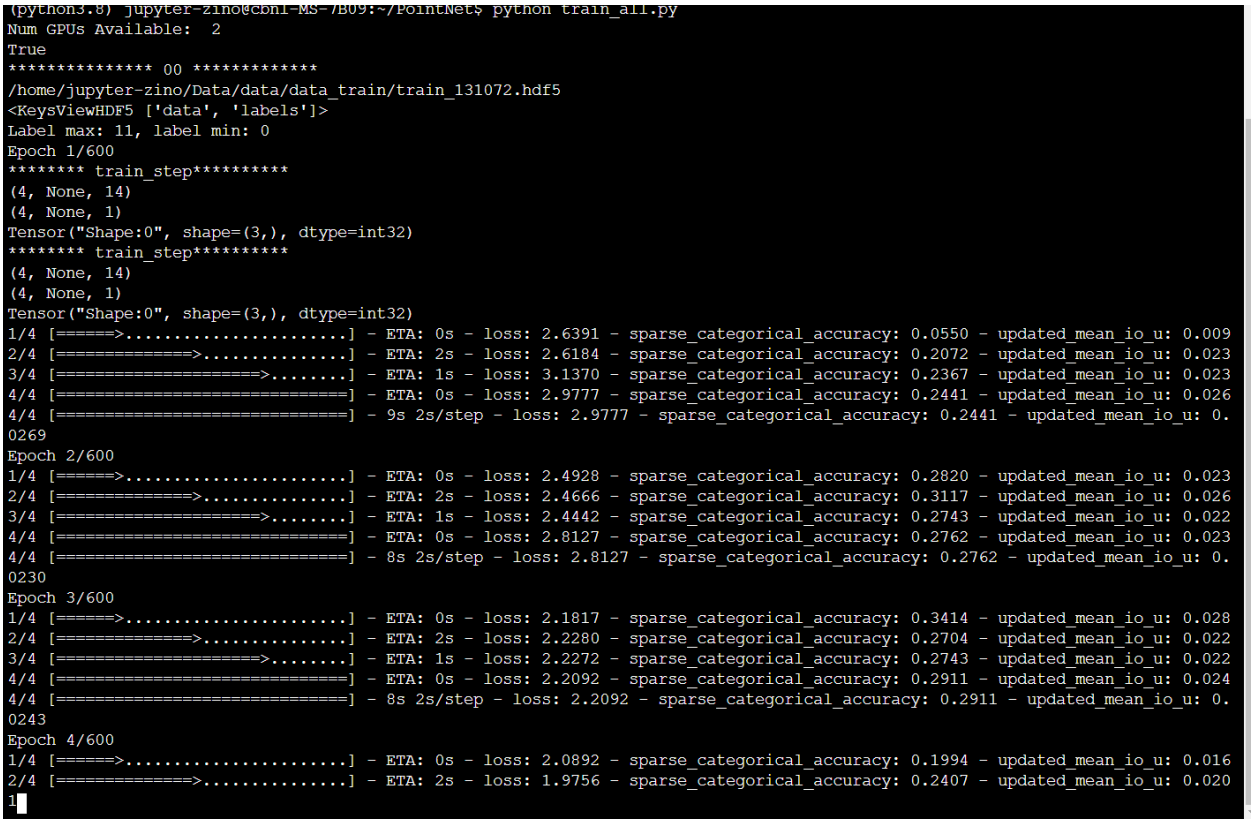

Figure 15: PointNet++ Model Training Console Output.

5.1.4 Applying and Evaluating the Model

During the process of training, the model already reported accuracies via the leave one out cross validation method. For further testing I chose one of the catenary arches, the file “bram_02_03.laz” to run an automated segmentation on. The model completed the segmentation with a mIoU of 0.83, which is considerably high, given the expectation that it would not perform well on smaller objects such as insulators. While it did not do a perfect job on the insulators, neither on the differentiation of poles and pole-feet, it is good enough to continue working with in phase 2 of this project. The segmented point cloud can be seen in figure 16.

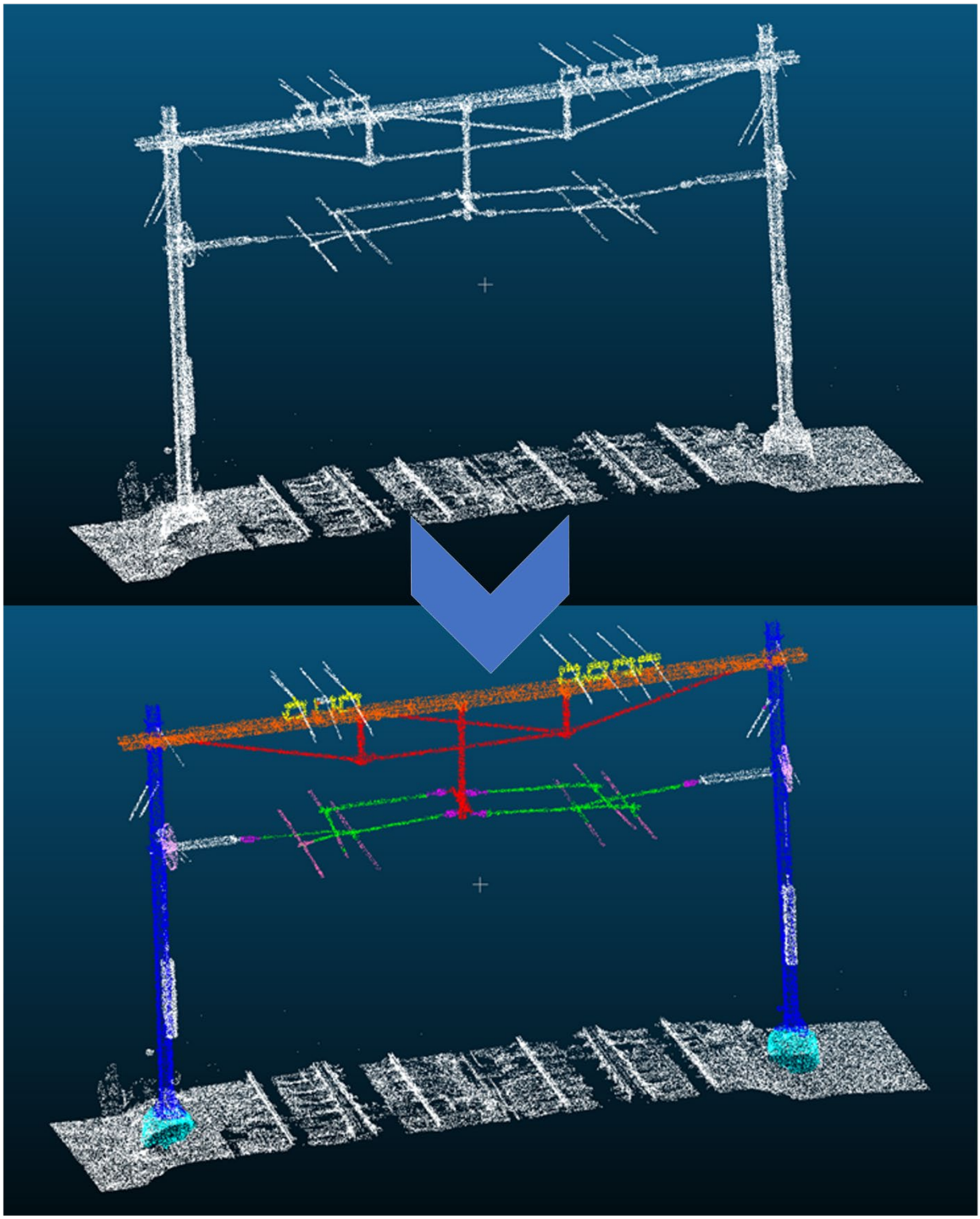

Figure 16: Automatically labelled/segmented point cloud.

5.2 Phase 2: Implementing the CAD Reconstruction Pipeline

Phase 1 left me with the ability to automatically label point clouds of catenary arches for a certain set of components. This automatic labeling is the first step in the final CAD reconstruction pipeline. Before I can continue implementing the next step, the template matching process, the CAD catalogue and template point clouds need to be prepared. During this chapter, I will include code excerpts of my implementations. For an overview of majority of the code used for the template matching step, see Appendices E and F.

5.2.1 Preparing the CAD Catalogue

As mentioned in chapter 4.2.2, the CAD archive provided by Strukton Rail [2] contains 59 individual CAD models, from which I am to select those corresponding to the labeled components the segmentation model has been trained for. While manually opening each of the 59 models in FreeCAD [36], it quickly becomes clear that these CAD models and the segmentation labels were not exactly provided with the other in mind. There are 18 different CAD models for poles, of which multiple have a pole foundation attached. Yet, there is no separate CAD model for the pole foundation itself. This would not necessarily be a major problem for the replacement process, as the pole foundation would be correctly transformed alongside the pole itself after matching the pole, the foundation existing in the pole's template point cloud may lead to problems or at least misleadingly low point region overlaps. For the start and the scope of the prototype, it is enough to consider the CAD models directly corresponding to each label. The extracted CAD models can be found in Appendix D.

After preparing the CAD catalogue, template point clouds must be generated. I intended to use CloudCompare [29] for this, as it has a tool that allows me to sample points from a mesh, but CloudCompare cannot import stp-files. Thankfully, STEP-formatted CAD objects can be exported as obj-files, which can be loaded into CloudCompare. By loading both the CAD model and an example of an automatically labeled point cloud scene, I can pre-scale the CAD model to the size of the segmented point cloud's components before sampling points, as can be seen in figure 17. I sampled a sufficiently large number of points to capture the shape of the CAD model in the point cloud, as can be seen in figure 18, and save it as a "laz"-file named based on the label corresponding to the template ("[l].laz", where [l] is the label).

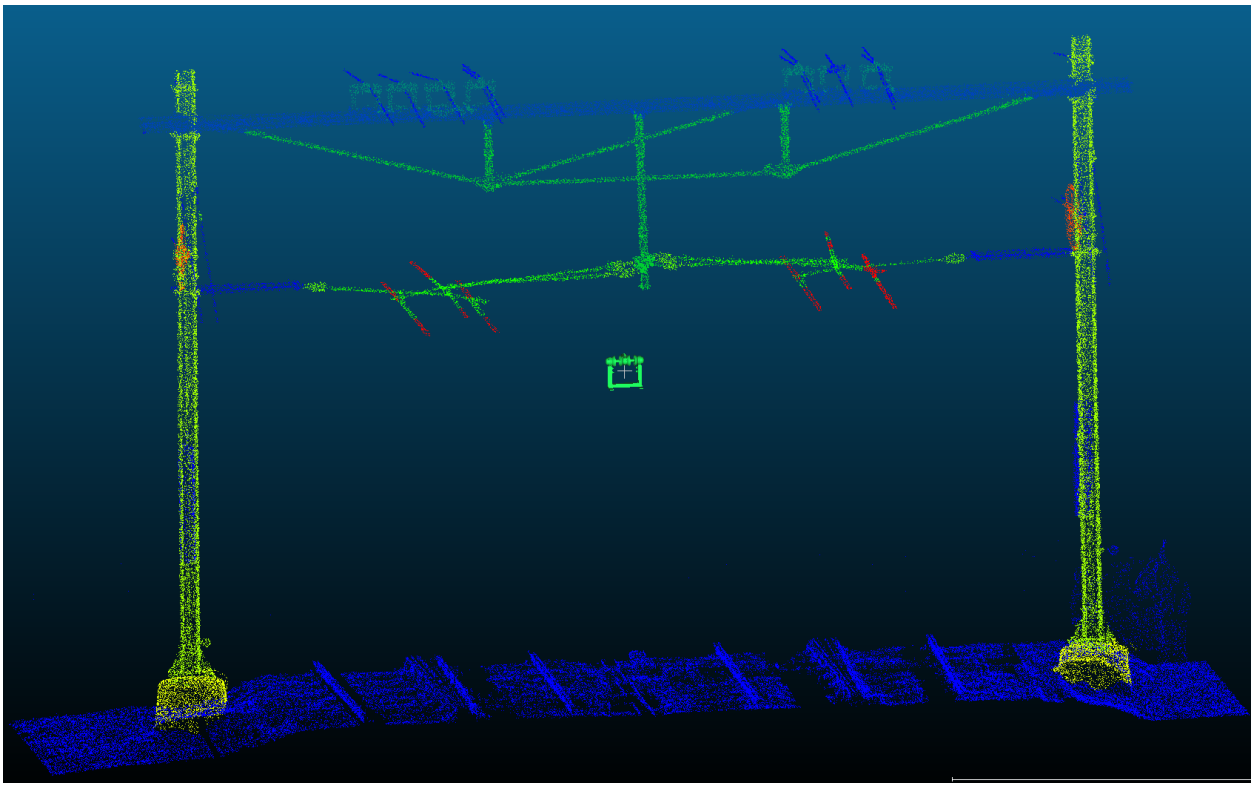

Figure 17: Scaled CAD model in relation to segmented point cloud.

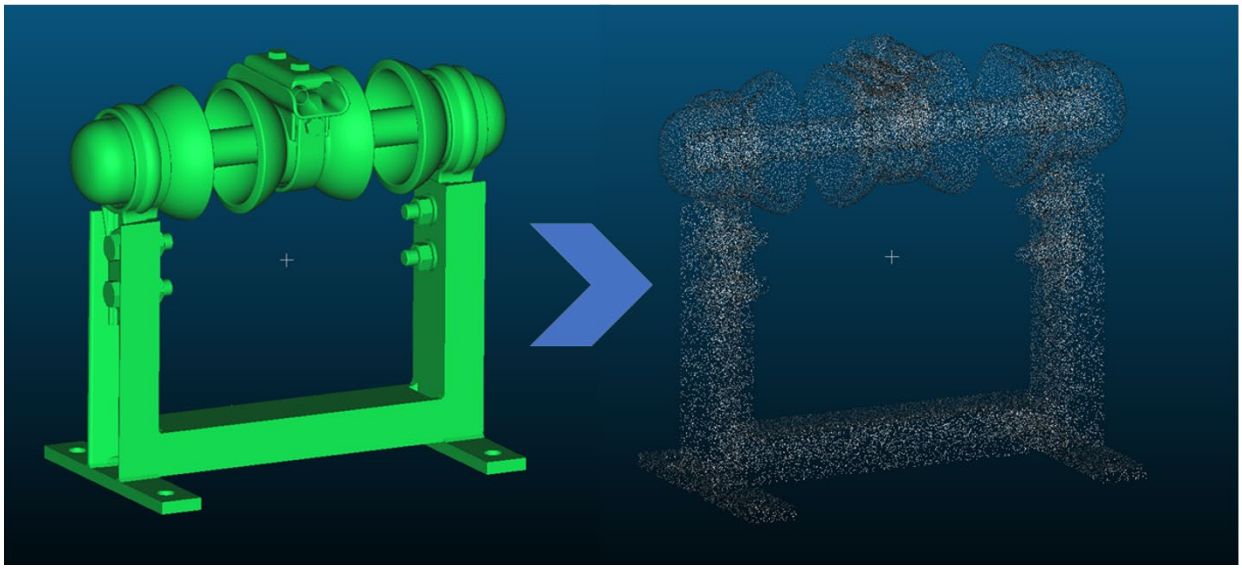

Figure 18: Point Cloud sampled from a CAD model.

5.2.2 Sampling Segmented Component and Template and Calculating Normals

The next step is to down-sample both the segmented component and the template point cloud using the same voxel-grid dimensions. Open3D [33] has an inbuilt method for voxel-based down-sampling,

so I converted my point cloud list into open3D geometry object to be able to apply voxel-based down-sampling to it, the implementation and example console output of which can be seen in figure 19 and 20.

```
151
152 ''' Voxel-based Downsampling
153 '   Samples points from input point list based on voxel grid.
154 '   Similar method used in prediction, so recommended to use.
155 '''
156 def voxel_down_sample(points, voxel_size=0.025):
157     pcd = o3d.geometry.PointCloud()
158     pcd.points = o3d.utility.Vector3dVector(points)
159
160     return np.asarray(pcd.voxel_down_sample(voxel_size).points)
161
```

Figure 19: Open3D-based voxel-based down sampling implementation.

```
---------------------------------------
--Preparing template features from folder: /home/jupyter-zino/Data/data/templates/
-> Loaded templates for labels [6, 2, 5]
---------------------------------------
--Attempting to load scene from: /home/jupyter-zino/Data/data/data_train/03_02_bram_pred.laz
--Total points: 131072
--Found labels: [ 0  1  2  3  4  5  6  7 10 11]
--Sorting objects:
Sorted  0 with length 69199
Sorted  1 with length 15896
Sorted  2 with length 2873
Sorted  3 with length 7842
Sorted  4 with length 3366
Sorted  5 with length 866
Sorted  6 with length 23202
Sorted  7 with length 5056
Sorted  10 with length 1526
Sorted  11 with length 1246
---------------------------------------
---------------------------------------------------------------------
--Running template matching and pose estimation for object with label 0
-> No template was found for label 0
---------------------------------------------------------------------
--Running template matching and pose estimation for object with label 1
-> No template was found for label 1
---------------------------------------------------------------------
--Running template matching and pose estimation for object with label 2
Found resolution of 0.0027984043804067664 in 1 ms.
Sampled 690 points for this object.
```

Figure 20: Console output of voxel-based down sampling using the implementation of figure 19.

Besides down-sampling, it is also necessary to calculate the normal vector of each point, which essentially represents the direction this point is facing. We need these vectors to calculate the features of the point pairs in the template matching step later on, as is described in chapter 4.5.2. Open3D [33] also provides a function for this task, which makes implementing it relatively straight-forward.

```
137
138 def generate_normals(points):
139
140     pcd = o3d.geometry.PointCloud()
141     pcd.points = o3d.utility.Vector3dVector(points)
142
143     pcd.estimate_normals(search_param=o3d.geometry.KDTreeSearchParamHybrid(radius=0.1, max_nn=30))
144 #     pcd.orient_normals_consistent_tangent_plane(15)
145
146     pts = np.asarray(pcd.points)
147     nms = np.asarray(pcd.normals)
148
149     return np.stack([pts[:,0], pts[:,1], pts[:,2], nms[:,0], nms[:,1], nms[:,2]], axis=1)
150
```

Figure 21: Method for calculating the normal vectors of a point cloud using open3D.

5.2.3 Instance Segmentation

While preparing for the template matching step, a problem became immediately noticeable. The template and the segmented scene component are very different. Primarily, this is because there will almost always be more than one of each component in the segmented point cloud, while the template generally consists of a singular component. This problem is visualized in the following figure 22.

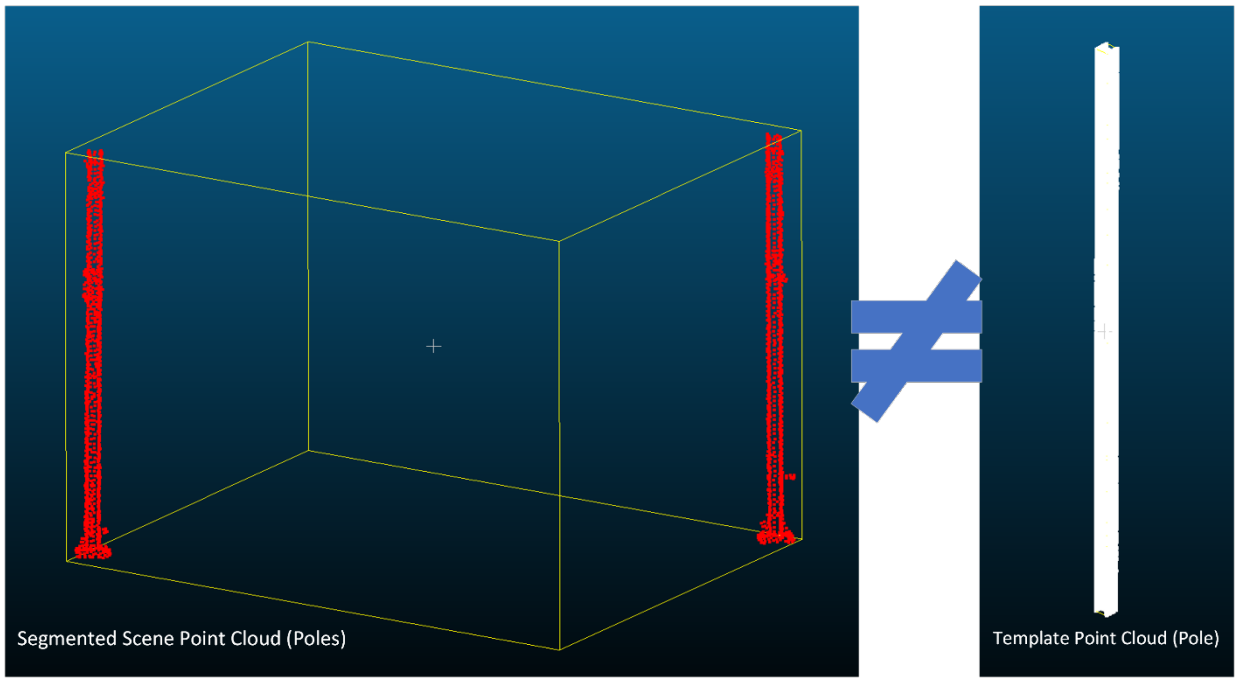

Figure 22: Incoherence between segmented scene and template point clouds.

I came up with two potential ways to get around this problem. The first is to apply some kind of instance segmentation algorithm to the already segmented point cloud to simply separate the two poles and then replace them individually. The second is to manually rebuild the pole template using CAD models

so that it would represent the segmented point cloud with two poles. While the first option might be computationally less efficient, since the template matching will have to be run for more instances in total, it might also perform better when the point clouds the template matching is done on would be less complex. Additionally, reconstructing templates for all the different combinations of poles, as can be seen in Appendix C, would take a lot of extra time and would be less scalable than an instance segmentation approach. Optimally, a deep learning model should be trained to perform the instance segmentation but considering the scope of this project and the point clouds I am working with, a point-neighborhood-based algorithm will do the job.

My prototype implementation is based on a simple algorithm that will continuously add points to a cluster. To add a point to the cluster, it must be within a certain range of the cluster's current centroid. After adding the point to the cluster, its centroid will be recalculated. If a point does not fall within a cluster, it will start its own cluster for which future points will checked. Overall, this approach is not very scalable, as it requires drastically different neighborhood-radius parameters, depending on how far instances are apart from each other, and would not work at all for long, parallel components with very little distance between them, such as cables, but it works fine for majority of poles and insulators that it has been tested on. In the future, this implementation would need to be replaced with an actual deep-learning-based instance segmentation model or some kind of graph-based algorithm.

```
227
228 ''' Segment Instances
229 '   Method that separates same object points from one point list into a list of point lists
230 '   for each instance of that object. Uses spherical clusters to determine instances.
231 '''
232 def segment_instances(points, diameter, th, axis=0):
233     start = time.perf_counter()*1000
234
235     centers = []
236     instances = []
237
238     for p in points:
239
240         # check if the current point belongs to an existing instance
241         added = False
242         for i, center in enumerate(centers):
243             if dist(p, center) < th:
244                 instances[i].append(p)
245                 centers[i] = np.average(instances[i], axis=0)
246                 added = True
247                 break
248
249         if added:
250             continue
251
252         # otherwise add a new instance
253         instances.append([p])
254         centers.append(p)
255
256     runtime = time.perf_counter()*1000 - start
257 #     print("Segmented", len(instances), "instances in", int(runtime),"ms.")
258     return instances
259
```

Figure 23: Simplistic instance segmentation algorithm.

After instance segmentation, the instances of the poles for example are automatically separated as can be seen in the following figure 24, with red and green representing the different instances.

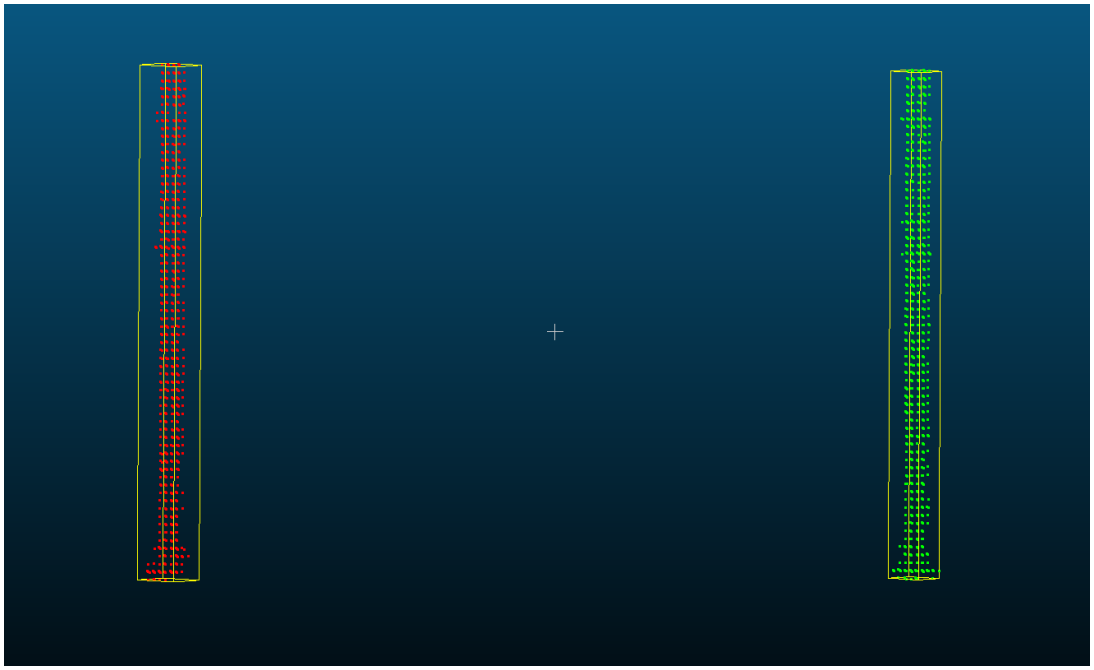

Figure 24: Instance segmented poles.

In the case of the segmented messenger wire support components, this algorithm meets its limitation, as the segmentation results in figure 25.b) show. In this case, the scope of this project suggested the manual instance segmentation of these components so that they could still be used for testing of the pipeline. The results of manual instance segmentation for the messenger wire supports can be seen in figure 25.c).

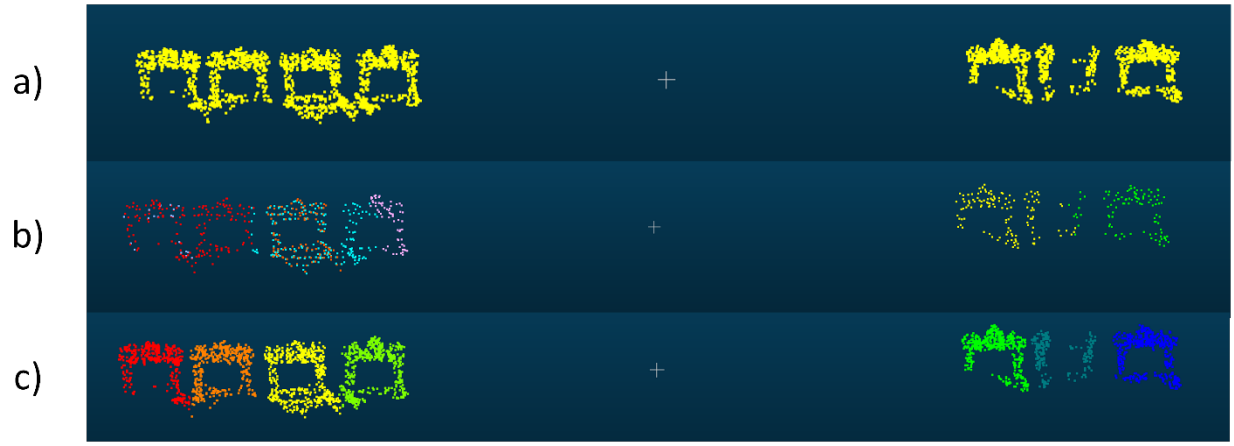

Figure 25: Segmentation of messenger wire supports. a) Automatically labelled messenger wire supports by the PointNet++ model. b) Automatic instance segmentation attempt. c) Manually instance segmented messenger wire supports.

5.2.4 General RANSAC Implementation

The next step is to implement the RANSAC loop. For this project it will follow the following pseudo-code, where the green text represents the actual RANSAC loop:

```
1.  For each component, repeat:
2.      For each instance of that component, repeat:
3.          var best_hypothesis
4.          var best_score
5.          Repeat x times: # where x is the maximum number of ransac iterations allowed
6.              Sample scene point pairs and create a hashmap using their hashed features
7.              Sample template point pairs and create a hashmap using their hashed features
8.
9.              Match scene and template point pairs
10.
11.             For each match, repeat:
12.                 Calculate transformation hypothesis
13.                 Score transformation hypothesis
14.                 Update best_hypothesis and best_score if applicable
15.
16.         Place CAD model into the scene
17.         Apply the best transformation hypothesis to the CAD model
18.
19. Save Scene
```

As the pseudo-code suggests, I decided to simply set a maximum for the number of RANSAC iterations allowed per component instance. Vock et al. [25] suggested the use of a probability framework that could estimate the number of iterations most likely needed to find a suitable hypothesis but playing with this max-iterations parameter will be effective enough for the prototype, as the template matching itself is not going to be particularly computationally slow.

The main program code for the template matching including my RANSAC implementation can be found in Appendix E.

5.2.5 Point Pair Sampling and Feature Calculation

The first step in the RANSAC loop itself is the sampling step. I decided on a parameter for the total amount of point pairs I want the RANSAC loop to sample on each iteration. Initially, I'll go with around 2000 point-pairs sampled. The sampling process is relatively simple: First I create an empty dictionary into which the point pairs will be saved, with their hashed features as key. Then, 2000 times, I'll select a random point from the instance point cloud, and the select another point, that is not the same as the first point. Then I calculate the feature vector for the point pair as described in chapter 4.5.2 and hash it to generate this pair's index. The purpose of hashing the feature vector is to very quickly be able to identify other point pairs that are relationally the same (or at least extremely similar to, as the features will be rounded slightly) as this point pair. This is because comparing one integer value with another is much faster than comparing four decimal values with four others. Lastly, the point pair is inserted into the dict with its hashed features as its key/index. Figure 26 visualizes this process.

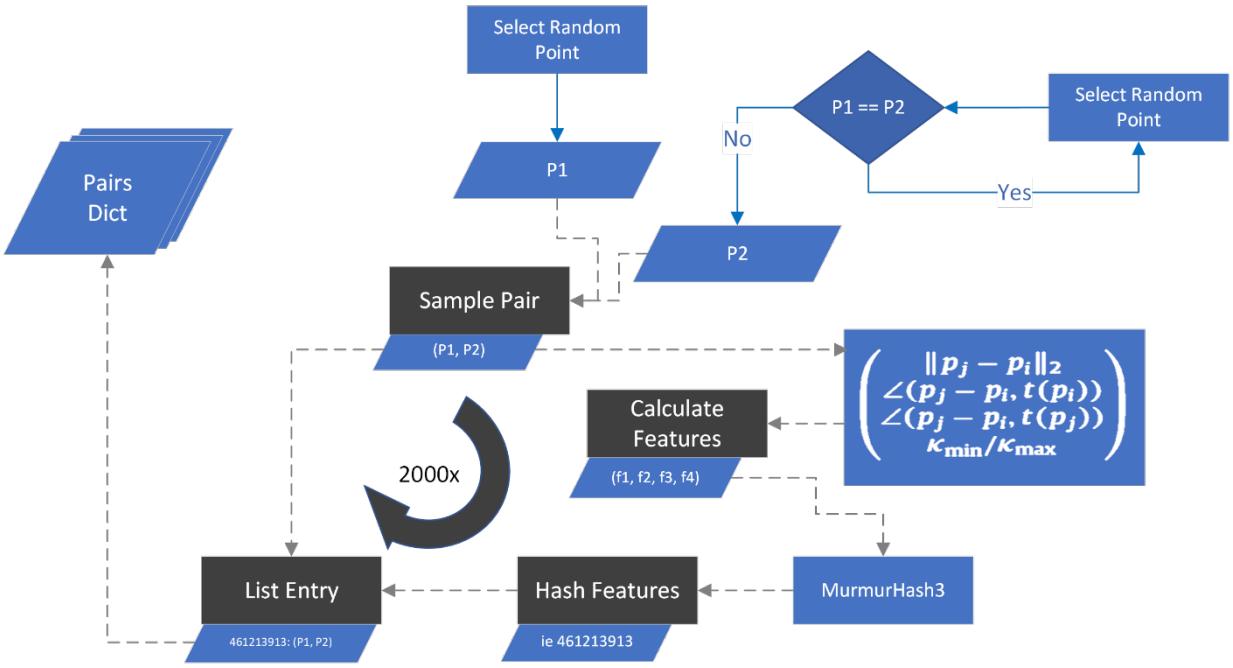

Figure 26: Point Pair sampling diagram.

In the code below you find the implementation of this loop and the calculations of point pair features. According to Vock et al. [25] the principle curvatures, or feature four, are not necessarily of importance for all types of objects, which is why I did not initially implemented it and set it to 0 in this implementation. Since the point pair matching worked to a satisfactory, even if considerably slow degree, I focused on improving other areas first. Reimplementing feature four for the point pair features might help optimizing the computational performance by decreasing the amount of matches per sampling while raising the quality of matches, which would allow for a higher number of iterations with potentially better transformation hypotheses.

```python
''' Point Pair Feature Retrieval
'   Method that samples point pairs from a point list and hashes their features.
'   Returns a dict with hashed features as indices and point pairs as values.
'''
def retrieve_ppf(points, pc_diameter, sample_count):
    pairs = {}

    for i in range(sample_count):

        # sample a random p1
        p1 = points[random.randrange(len(points))]

        # sample a random p2 that is within the neighbourhood of p1
        p2 = None
        while p2 is None:
            p2 = points[random.randrange(len(points))]

            if (p1 == p2).all() or dist(p1, p2) > pc_diameter:
                p2 = None

        # Get an index from the pairs features
        hash_val = mmh3.hash_from_buffer(features(p1, p2))

        # Add it to the hashmap
        pairs[hash_val] = (p1, p2)

    return pairs
```

```python
''' Calculate Features
'   Method that calculates the actual point pair features for two input points and
'   returns a tuple containing the features.
'''
def features(p1, p2):

    m1, n1 = pan(p1)
    m2, n2 = pan(p2)

    # Feature 1: distance between point 1 and point 2
    f1 = round(dist(p1, p2))

    vec_dis = m2 - m1

    # Feature 2: angle between distance vector and p1's directional vector (normal)
    f2 = round(angle_between(vec_dis, n1))

    # Feature 3: angle between distance vector and p2's directional vector
    f3 = round(angle_between(vec_dis, n2))

    # Feature 4: Kmin / Kmax, where Kmin and Kmax are the principal curvatures at p1
    f4 = 0

#     print([f1, f2, f3, f4])
    return np.array([f1, f2, f3, f4])
```

Figure 27: Point Pair sampling implementation.

One conscious decision at this point was to not only resample scene point pairs, but also template point pairs. This might seem like a waste of computational power, but I found that the computational power required for the sampling step itself is negligible. Resampling template pairs as well does not actually decrease or increase the probability of matches but ensures that the program will never fail to execute its task simply because one batch of sampled template point pairs was not suitable for matching at all.

The selection of matched point pairs is straightforward. By looping through all dict keys of the scene point pairs, one can easily check in python whether that key also exists in the dict of template point pairs. Alternatively, a union function could be applied to the scene and template dict keys, returning the union of those to lists. Using numpy [37] for this task is likely the most computationally optimized solution, but is unlikely to make a significant difference in the total runtime in comparison to more intensive tasks like the scoring of the transformation hypothesis.

5.2.6 Transformation Hypothesis Generation

The process of calculating transformation hypotheses is essentially a python version of the linear algebra equations introduced in chapter 4.5.2. First the point pairs need to be unpacked and the point and normal vectors separated. Then they can be applied to the formulas as can be seen in figure 28. My implementation of the formula that calculates the translational vector (cvec in figure 28) appears a little different than in the provided formulas. This is most likely because of a flipped coordinate system or a difference in the normalization of the dataset between Vock et al. and this project.

$$u_{ij} = \frac{p_j - p_i}{\|p_j - p_i\|_2}, \quad v_{ij} = \frac{v'_{ij}}{\|v'_{ij}\|_2},$$
$$v'_{ij} = (\mathbb{1} - u_{ij}u_{ij}^T)\, t(p_i),$$

where $\mathbb{1} \in \mathbb{R}^{3\times3}$ is the identity matrix and

$$R_{ij} = \begin{pmatrix} u_{ij} & v_{ij} & (u_{ij} \times v_{ij}) \end{pmatrix} \in \mathbb{R}^{3\times3}$$

$$T_{ij,kl} = \begin{pmatrix} R_{kl}R_{ij}^T & p_k - R_{kl}R_{ij}^T p_i \\ 0 & 1 \end{pmatrix} \in \mathbb{R}^{4\times4}$$

```python
''' Calculate Hypothesis
'   Generates a transformation hypothesis based on matched scene and template pairs.
'   INCOMPLETE
'''
def generate_hypothesis(scene_pair, template_pair):
    pi, pj = scene_pair
    pk, pl = template_pair

    mi, ni = pan(pi)
    mj, nj = pan(pj)
    mk, nk = pan(pk)
    ml, nl = pan(pl)

    uij = (mj - mi) / np.linalg.norm(mj - mi, 2)
    vpij = np.matmul((np.identity(3) - np.matmul(uij, uij.T)), ni)
    vij = vpij / np.linalg.norm(vpij, 2)

    ukl = (ml - mk) / np.linalg.norm(ml - mk, 2)
    vpkl = np.matmul((np.identity(3) - np.matmul(ukl, ukl.T)), ni)
    vkl = vpkl / np.linalg.norm(vpkl, 2)

    rij = np.asarray([uij, vij, np.cross(uij, vij)])
    rkl = np.asarray([ukl, vkl, np.cross(ukl, vkl)])

    rijkl = np.matmul(rkl, rij.T)

    cvec = mk + np.matmul(rijkl, mi)

    tijkl = np.asarray([
      np.asarray([rijkl[0, 0], rijkl[0, 1], rijkl[0, 2], cvec[0]]),
      np.asarray([rijkl[1, 0], rijkl[1, 1], rijkl[1, 2], cvec[1]]),
      np.asarray([rijkl[2, 0], rijkl[2, 1], rijkl[2, 2], cvec[2]]),
      np.asarray([0, 0, 0, 1])
    ])

    return tijkl
```

Figure 28: Transformation Matrix Hypothesis Implementation with direct representation of formulas.

Figure 29 shows all the hypotheses generated by matched point pairs over one RANSAC iteration as applied to the template in yellow. It quickly becomes obvious that around half of these transformations are transforming the position of the template in a completely wrong direction. It is possible to exclude these transformations from being tested on, by simply checking if the translation vector of the transformation matrix is within a certain range of the directional angle between a point of the template and the scene. Simply returning "None" instead of a valid transformation matrix allows to easily exclude them from the validation step. The results of the removal can be seen in figure 29 in blue and the code used for it in figure 30 below.

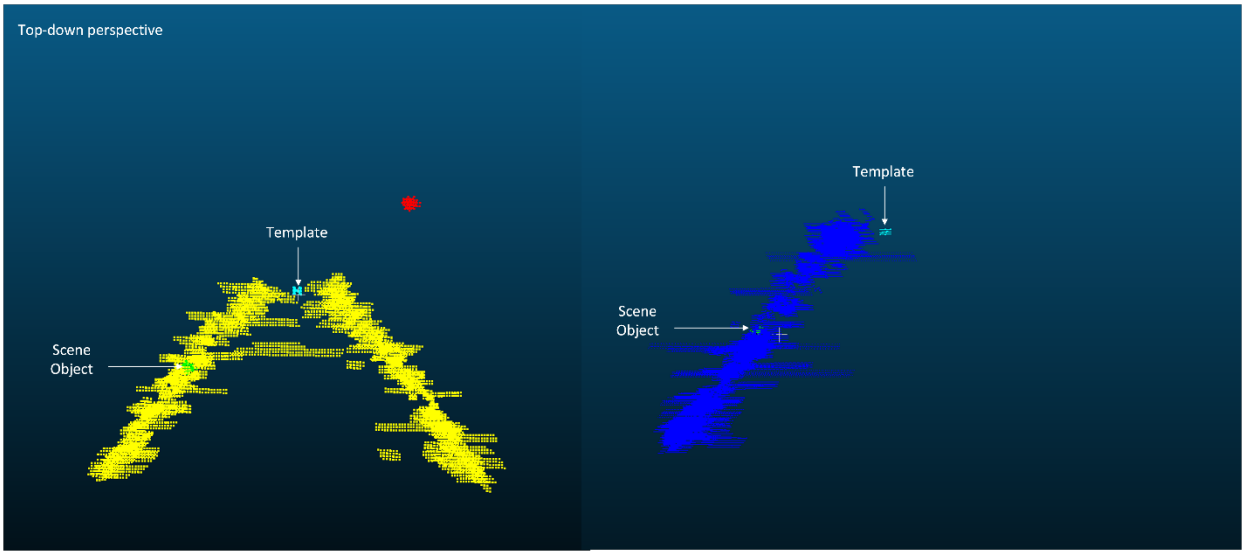

Figure 29: Exclusion Visualizations: Collections of applied transformation hypotheses of the pole component. In yellow, collection of applied transformation hypotheses before exclusion. In blue, collection of applied transformation hypotheses after exclusion.

```
344
345     dir_angle = np_angle([1, 0, 0], (mi - mk))
346     hypo_angle = np_angle([1, 0, 0], cvec)
347     if hypo_angle < dir_angle-np.pi/8 or hypo_angle > dir_angle+np.pi/8:
348         return None
349
```

Figure 30: Hypothesis Exclusion Implementation.

Lastly, considering some of the transformation results, I found it worth testing whether removing scaling and rotations from the transformations could have a positive effect on the result for some components, such as poles, and messenger wire supports, which -based on the available dataset- have generally the same orientation. While it isn't guaranteed that locking rotations (and possibly scaling) for some components will make sense in the long run, I found it worth exploring for the scope of the

prototype. The following change in code allows me to lock the rotation and scaling of the transformation matrix. The results of this alternative are discussed in chapter 6.

a) Unlocked

```
354
355     tijkl = np.asarray([
356      np.asarray([rijkl[0, 0], rijkl[0, 1], rijkl[0, 2], cvec[0]]),
357      np.asarray([rijkl[1, 0], rijkl[1, 1], rijkl[1, 2], cvec[1]]),
358      np.asarray([rijkl[2, 0], rijkl[2, 1], rijkl[2, 2], cvec[2]]),
359      np.asarray([0, 0, 0, 1])
360     ])
```

```
Matching for instance 2/2 of object 6.
Fine Alignment produced a score of 0.84 in 68496ms.
- RANSAC complete with best score 0.84.
Transformation Matrix:
0.9986176001185973 -0.03382425061558211 -0.028075283260066187 0.38699790946093
0.010624154814777617 1.0 6.938893903907228e-17 0.30371726354576
0.02807528326006619 1.7184604433895245e-17 0.9989769541936312 0.05676175927817706
0.0 0.0 0.0 1.0
```

b) Locked

```
361
362     tijkl = np.asarray([
363      np.asarray([1.0, 0.0, 0.0, cvec[0]]),
364      np.asarray([0.0, 1.0, 0.0, cvec[1]]),
365      np.asarray([0.0, 0.0, 1.0, cvec[2]]),
366      np.asarray([0.0, 0.0, 0.0, 1.0])
367     ])
368
```

```
Matching for instance 2/2 of object 6.
Fine Alignment produced a score of 0.72 in 76135ms.
Fine Alignment produced a score of 0.96 in 65242ms.
- RANSAC complete with best score 0.96.
Transformation Matrix:
1.0 0.0 0.0 0.37060721716726946
0.0 1.0 0.0 0.30553450513423336
0.0 0.0 1.0 0.02502122311108919
0.0 0.0 0.0 1.0
```

Figure 31: Implementation and output difference between rotation/scaling locked and unlocked transformation matrices.

5.2.7 Transformation Hypothesis Scoring

In order to score and evaluate a transformation hypothesis, the hypothesis must first be applied to the template. This can easily be done by extending the point vector by one dimension to make it eligible to multiply with a 4x4 transformation matrix, which is exactly what is done then. Figure 32 shows the numpy-based [37] code used to apply the transformation matrix to a point list.

```
462
463 ''' Apply Transformation
464 '   Method that applies a transformation to a point list. Returns a point list.
465 '''
466 def apply_transformation(points, trans_matrix):
467     data = np.hstack((points, np.asarray([[1]]*len(points))))
468     return np.matmul(trans_matrix, data.T).T[:,:3]
469
```

Figure 32: Applying a transformation matrix.

Once a template point cloud has been transformed, the corresponding transformation matrix can be scored by comparing the transformed template to the scene's component instance. To generate a score, I wrote what I call a "point-region-overlap" algorithm. It iterates over the points of the template and checks whether a scene point is within its immediate vicinity. Since this is a considerably computationally unoptimized algorithm, it is slow enough to significantly impact the runtime negatively. I therefore instead sampled 100 random points from the template and scored using these 100 points. Additionally, to improve performance, I stopped the scoring process once it was guaranteed not to beat

the previous best score. In hindsight, this choice may not be optimal, as even slightly lower scores can produce better results via fine-alignment later on.

The most important parameter of the scoring algorithm is the distance threshold that determines the point's immediate vicinity/neighborhood. I initially experimented with a fixed parameter, but then opted to use the scene component point cloud's resolution, which is essentially the point cloud's average distance of each point to its nearest neighbor, of which my implementation can be found in figure 33. This makes sense here, for one because there are no outliers in the instance segmented point clouds, and for another because it suggests to limit the positional shift of scene to template at a max of one point distance in the case of an "optimal" transformation. My point-region overlap algorithm is shown in figure 34.

```
181
182 ''' Point Cloud Resolution
183 '   Calculates the resolution of an input point point cloud, approximated as the average nearest neighbour
184 '   distance.
185 '''
186 def pc_resolution(points):
187     start = time.perf_counter()*1000
188
189     distances = []
190     for (point, i) in zip(points, knn(points, 1)):
191         distances.append(dist(point, points[i][0]))
192
193     output = sum(distances) / len(distances)
194
195     runtime = time.perf_counter()*1000 - start
196     print("Found resolution of", output, "in", int(runtime), "ms.")
197
198     return output
199
```

Figure 33: Point Cloud Resolution Implementation.

```
470
471 ''' Point Region Overlap
472 '   Tests point of a test point cloud against proximity to points in a control point cloud and returns the
473 '   percentage of test points that are in proximity with control points.
474 '''
475 def point_region_overlap(test, control, best_score, th=1):
476     start = time.perf_counter()*1000
477
478     miss_limit = int((1-best_score)*100)
479     n = 0
480
481     indices = np.random.randint(len(test), size=100)
482     for i, tp in enumerate(np.array(test)[indices]):
483         for cp in control:
484             if dist(tp, cp) < th:
485                 n += 1
486                 break
487
488         if i - n > miss_limit:
489             break
490
491     runtime = time.perf_counter()*1000 - start
492 #     print(f"Performed region overlap in {int(runtime)} ms with result {n/100}.")
493
494     return n/100
495
```

Figure 34: Point Region Overlap Scoring Algorithm.

5.2.7 Fine Alignment

After finding a hypothesis with a promising score, which is a heuristic I set individually for each component throughout testing to achieve workable results, each transformation would be fine aligned. The threshold depended primarily on the shape of the component. For example, the score threshold for fine aligning the poles was substantially higher than for insulators, because a low score for the poles generally suggested a significantly wrong rotation, and the higher scores suggested an already near optimal solution that would be significantly less computationally difficult to fine-align. Obviously, an argument could be made to attempt to fine align some of the lower solutions as well, but since the fine alignment algorithm employed required continuous rescoring using the computationally inefficient point region overlap, the computational cost would be too great, and it was overall more efficient to adapt the thresholds as described.

The fine alignment algorithm I wrote only attempted to realign the template and scene via translations, ignoring rotations and scaling. In the future, it is suggested to also fine align rotations and scaling, but those would be significantly harder to implement than translations, which is why they were ignored in the scope of this project. Additionally, the fine-alignment algorithm employed is incapable of retracing its steps once it found a slightly better solution, which while computationally less time consuming, is likely to cause it to miss more optimal solutions for more complex components such as messenger wire supports. The algorithm was originally intended to function well on simply shaped components such as poles and insulators. A future iteration of this algorithm would need to be able to retrace its step and iteratively try to fine align the template using a different order of translations.

In my implementation, it will first attempt to fine align the template, on the x-, then y-, and z-axis respectively. For each axis, it would try to translate the template by increasingly shrinking values starting from both 1 and -1. Whenever it finds an improved score, it will attempt the same translation again until the score can no longer improve, upon which it would shrink the translation value by 90% and try again until a certain threshold is reached. Originally, I set the shrinking to 50%, but 90% resulted in significantly better aligned end results, although it was computationally costly. Figure 35 shows examples of the algorithm attempting to fine align multiple templates. The blue point cloud is always the transformed template before fine-alignment, and the white one is after fine alignment. As is clearly visible, none of these are particularly good fine alignments, but they very nicely demonstrate some of the weaknesses of the prototype fine-alignment algorithm I implemented. The poles in 35.a) cannot correct for faulty rotations, the insulators in 35.b) don’t find back towards the scene object, once they have settled at a

slightly better position while putting themselves into a position from which the fine-alignment algorithm cannot recover, and the shape of the messenger wire support in 35.c) is too complex for the fine-alignment to recognize a good path towards the optimal solution. Yet, with the right starting hypothesis, this fine-alignment algorithm can still reach very high overlap percentages on each of these components. Hence, improving the fine-alignment algorithm is primarily a question of computational efficiency, as less iterations would be needed to find a hypothesis that the fine-alignment can optimize.

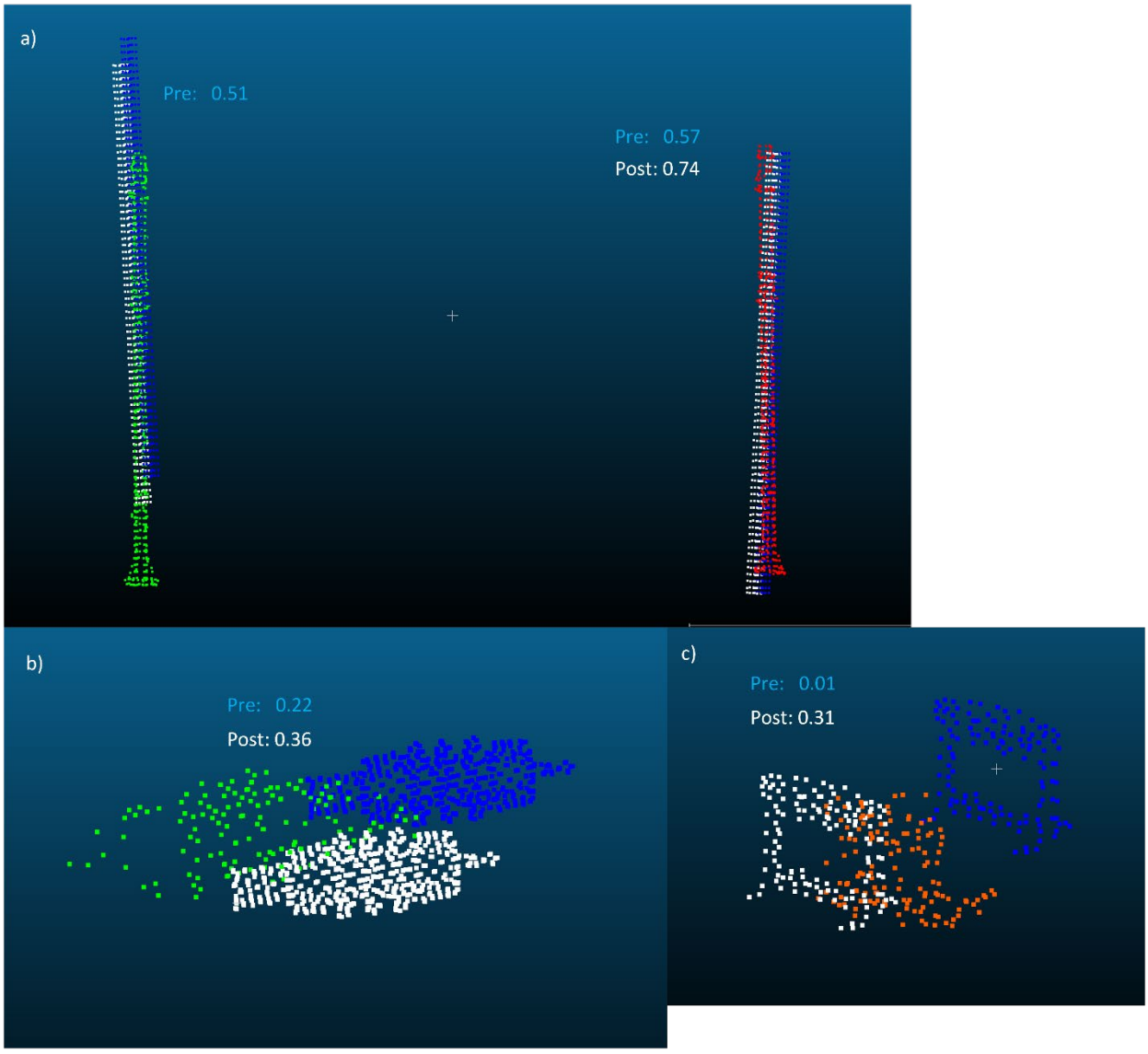

Figure 35: Fine alignment algorithm failing.

Throughout fine alignment, whenever an improvement is found, the transformation matrix would also be updated by adding the translation step undertaken by the fine alignment algorithm. My prototype implementation consists essentially of the following for-loop three times, for each axis one loop.

```
370
371     output = input
372     score = init_score
373
374     step_th = 0.001
375
376     fine_align_matrix = x_matrix_trans(0)
377
378     # x translation
379     for i in [-1, 1]:
380         step = i
381         while abs(step) > step_th:
382
383             change = apply_transformation(output, x_matrix_trans(step))
384             new_score = point_region_overlap(change, control, 0.0, th)
385
386             if new_score > score:
387                 output = change
388                 score = new_score
389                 fine_align_matrix = add_translation(fine_align_matrix, x_matrix_trans(step))
390                 continue
391
392             step /= 1.1
393
```

Figure 36: 1/3 fine alignment implementations.

5.2.7 CAD Placement

The final step of the pipeline is the actual placement of a CAD model into the scene for each component using the correct position, rotation, and scaling. For the scope of this project, it was determined unnecessary to automate this step, as it has little technical implications on the feasibility of the pipeline. Instead, CloudCompare [29] was used to load the CAD model, in the form of an .obj file, corresponding to each component point cloud template. Then, the transformation matrix returned by the template matching for each instance of a component could be applied to the model, placing the model in the same position as the transformed template point cloud. Figure 37 shows this process in visualized steps as well as the resulting CAD model scene. The scene could then be saved as a single .obj file and converted into the original .std format using FreeCAD [36].

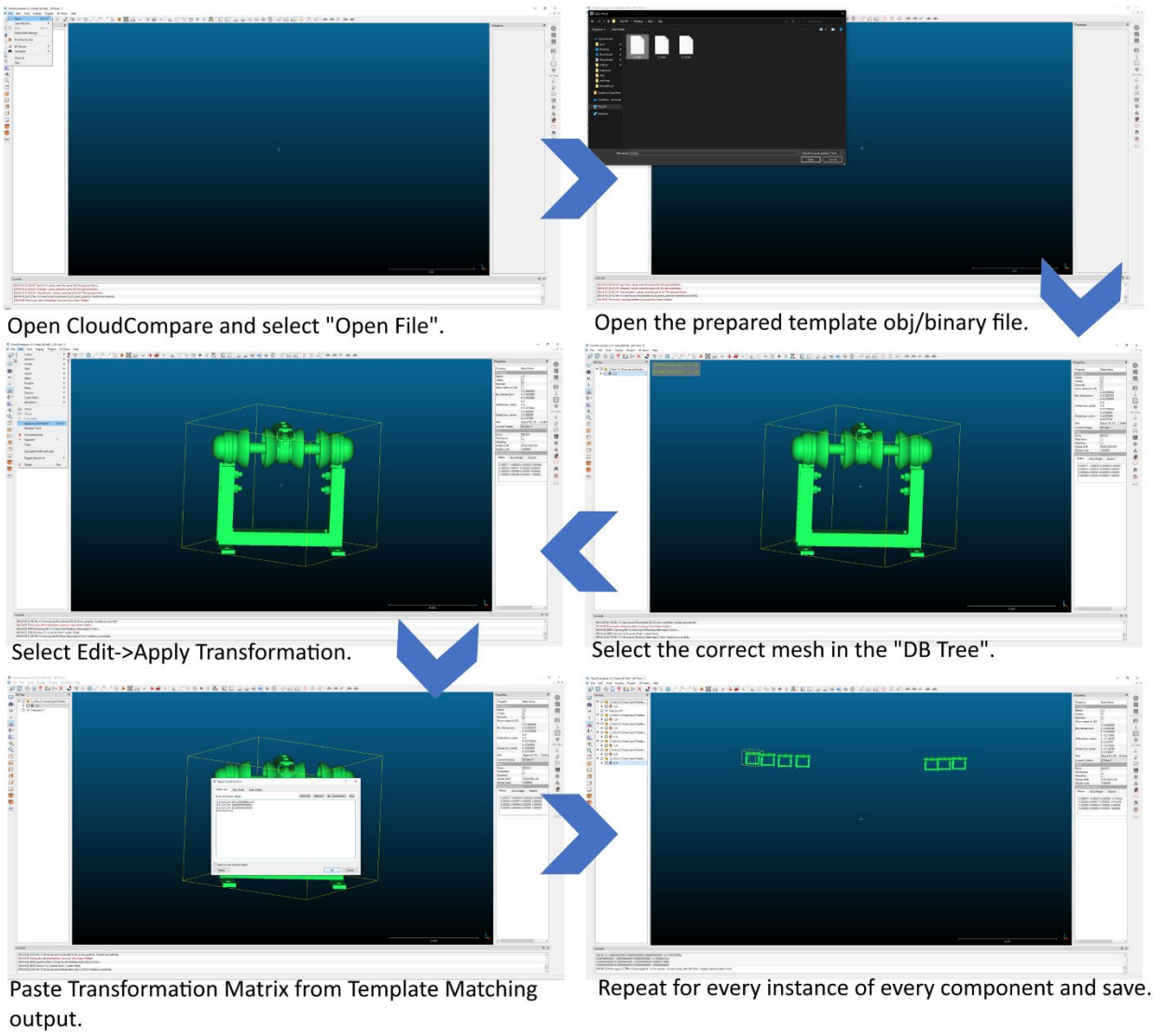

Figure 37: CAD Placement step-by-step using CloudCompare.

5.2.8 Final Pipeline as Implemented

The following figure 38 shows the specific pipeline I implemented for this prototype as explained in this chapter. It corresponds to the general pipeline introduced in chapter 4.3. It contains both the full pipeline for the general implementation and training of a semantic segmentation model, with the data preparation steps unspecific to PointNet++ or this implementation, and the actual pipeline for the reconstruction of 3D models, which uses the output of the deep learning pipeline as one of its building blocks.

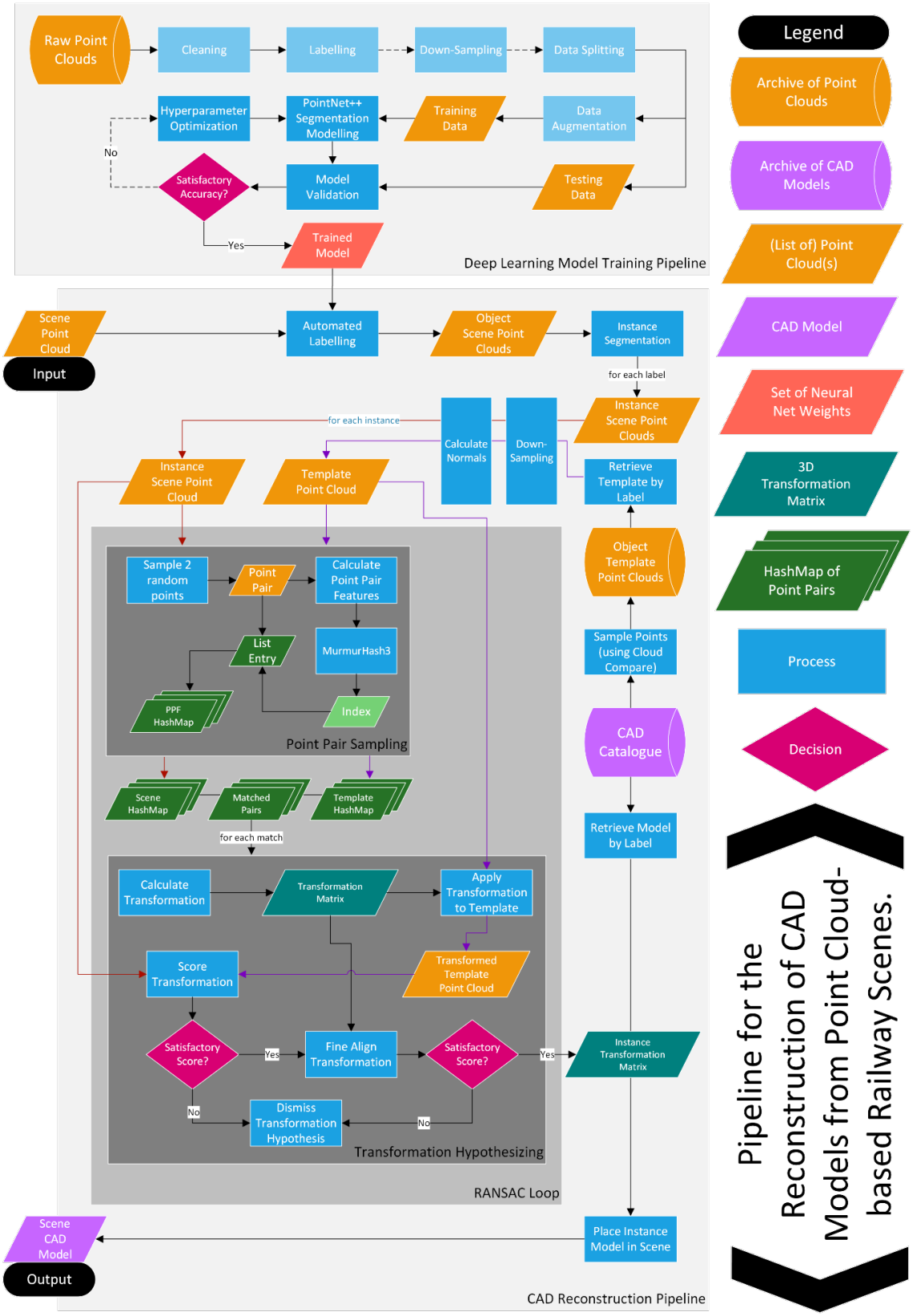

Figure 38: Detailed pipeline mirroring the implementation.

Chapter 6 - Evaluation

In order to evaluate the pipeline, the point-region-overlap heuristic is used, comparing the transformed template to the original scene objects. While these scores are heavily dependent on the shape of each component itself, the shape parity between template and scene object, and the quality of the segmentation results in an earlier step, they do provide information about which components the prototype generally struggles to align.

In the following table, the average alignment scores of best transformations before and after fine-alignment for each tested component can be found. Additionally, a comparison between the fully calculated transformation matrix and transformation matrices with locked rotation and scaling is included as well.

Component	**Pre-Fine-Align (Unlocked)**	**Post-Fine-Align (Unlocked)**		**Pre-Fine-Align (Locked)**	**Post-Fine-Align (Locked)**
Poles	0.78	0.95		0.78	0.98
Insulators	0.66	0.83		0.67	0.77
Messenger Wire Supports	0.33	0.78		0.33	0.70

Table 2: Average point region overlap of best hypothesis after 100 RANSAC iterations for each component, with and without rotations and scaling locked and before and after fine alignment. Data from a single pipeline run using one catenary arch.

It is curious, but not entirely illogical that the averages for pre-fine-aligned scores are so similar between the rotation/scaling locked and unlocked tests. With enough RANSAC iterations, the algorithm should almost always be able to come up with a hypothesis that seems plausible enough. The similarities in scores between these two groups suggests that each component has an approximate limit to the possible scores without fine alignment. This limit likely depends on the complexity of the shape of the component and its relative position to the initial template position. Messenger wire supports have by far the most complex shape out of the three components tested as well as the lowest direct resemblance between scene objects and template, so the lower score makes sense.

For both the insulators and messenger wire supports, the post-fine alignment results of the rotation/scaling unlocked group are considerably better than for their locked counter parts, but the sample size is hardly significant, given that this is the average data of one run of the pipeline. For both components, it is not unlikely that they have benefitted from slight down-scaling transformations, which would push more of the template points to overlap with the points from the scene object. While this could

also be true for the poles, its lengthy shape causes it to drop significantly more overlap percentages even at small rotations on any but the vertical axis, which would explain why the rotation locked group was able to outperform the unlocked group marginally. While one would generally expect poles to be standing vertical all the time, this does not necessarily have to be true in real world scenarios, especially in the case of severely damaged catenary arches, which is why this marginally better performance of the locked group is not particularly significant.

Since these values do not provide a full understanding of the quality of the results of the prototype, an additional visual evaluation is in order. By comparing the original point cloud to the segmentations and the resulting CAD model, the applicability of the approach can be estimated. Figure 39 is supposed to provide this visual evaluation, by visualizing all the major steps of the pipeline: semantic segmentation, instance segmentation, template matching, and CAD placement.

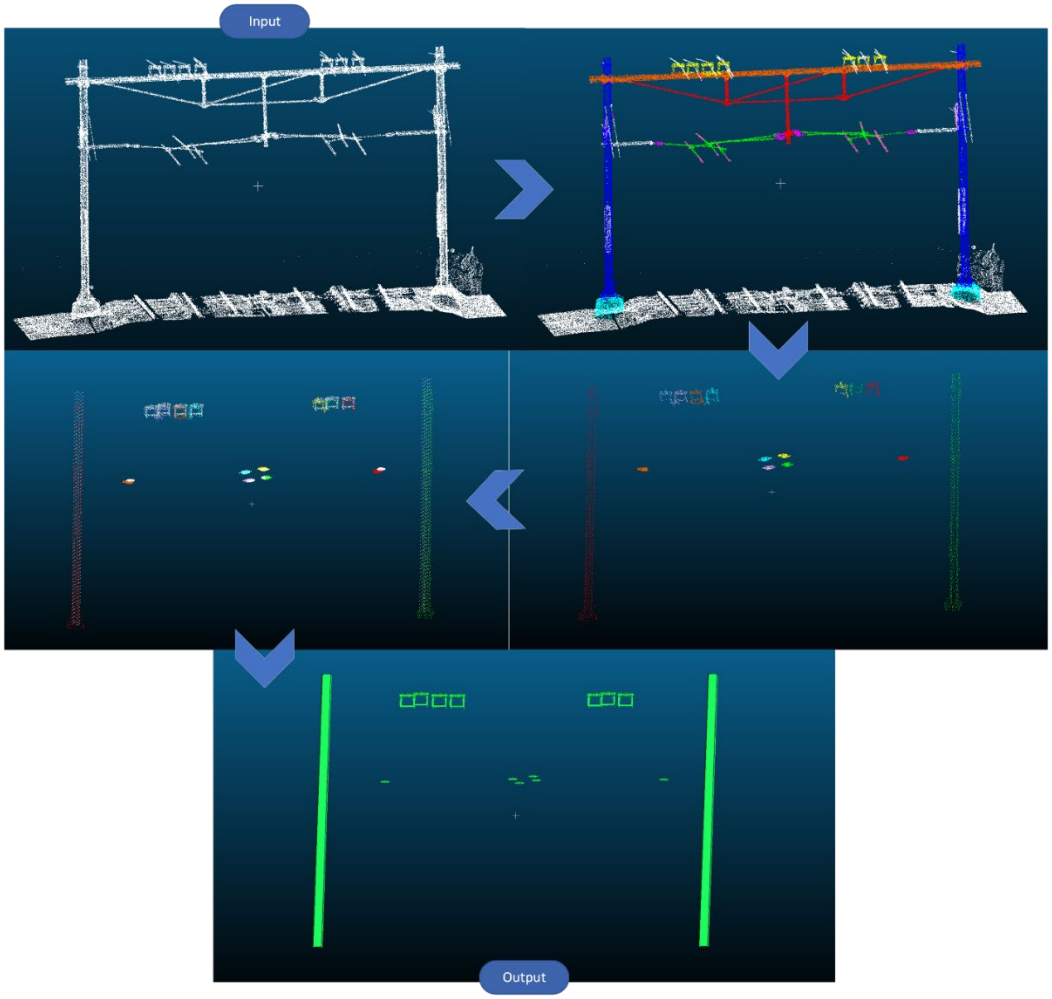

Figure 39: Visualized steps and result of the pipeline prototype.

As can be seen in Figure 39, the resulting CAD model is not complete, as the prototype was primarily tested using three components: the vertical poles, insulators, and messenger wire supports. This was mostly due to a lack of available CAD models and time constraints as a result of the scope of this project. At the same time, the prototype showed that it can vaguely place a CAD model in the position of each scene component instance, which is promising. It clearly struggled with the fine alignment of the messenger wire supports, and the 180-degree rotations of the insulators, which all face the same direction. Both problems can be solved via an increased fidelity in the template matching and improved fine-alignment. Additionally, besides scaling, the prototype was almost able to perfectly place the poles into the scene. The only improvement to make here would be to include scaling in the fine-alignment algorithm.

The general model pipeline introduced in chapter 4.3 holds true and can be considered the successful result of this project, alongside the detailed pipeline introduced in chapter 5.2.8. It is reiterated here in figure 40.

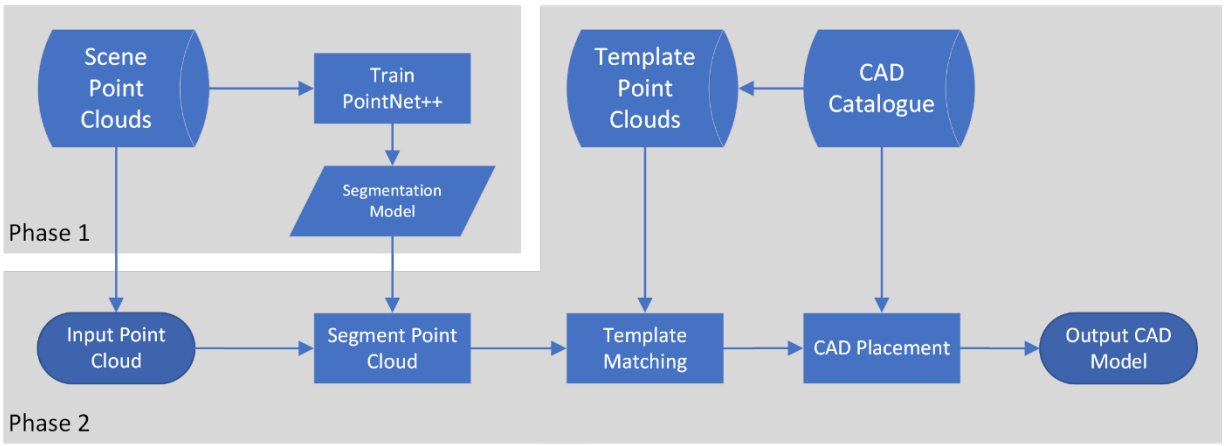

Figure 40: Confirmed generalized pipeline for the reconstruction of CAD models from point cloud-based railway scenes.

Chapter 7 - Conclusion

The pipeline developed throughout this project is a successful prototype for the task of reconstructing 3D models from point cloud-based railway scenes. The requirement was for the pipeline to be able to perform on both the largest and the smallest component of the catenary arch, which it did to a degree where future improvements within the frame of the existing pipeline may lead to production-quality results. The pipeline consists of three primary steps: the segmentation of the catenary arches (semantic as well as instance segmentation), iterative template matching for each segmented component instance to retrieve an optimal transformation matrix, and finally the CAD placement to generate a complete scene 3D model using the transformations.

The modified PointNet++ developed by Bram Ton of the Ambient Intelligence Research Group was a suitable segmentation model for this prototype, as its results generally allowed for the template matching and CAD placement steps to work. It only struggled partially with insulators, the smallest components of catenary arches, which required a few adjustments in the data used for the template matching.

The use of locked rotations and scaling as part of the transformation hypothesis generation step of the template matching has shown to have merits in the performance of replacement of certain components. Depending on the expected features of the components and the reliability of the scan angle and data normalization processes, the future use of locked rotations and scaling for specific components such as messenger wire supports is recommended, due to their regular position and orientation within catenary arches.

The conclusion is that the chosen approach of using a point-based deep learning segmentation model, alongside point pair feature and RANSAC based template matching to reconstruct 3D models from point cloud-based railway scenes using a provided CAD catalogue has been successful.

Chapter 8 – Future Work

Opportunities for future work have been hinted at throughout the project. They include the improvement of the semantic segmentation model or testing of other models, the extension of the CAD catalogue, the replacement of the instance segmentation algorithm, and the improvement of algorithms used throughout the template matching step, as well as the added automation of multiple steps of the pipeline.

Firstly, improving the semantic segmentation model would be one of the greatest benefactors to the performance of this pipeline. To correctly match the template and scene objects, the scene object must be correctly segmented in the first place, which the currently used model struggles with for multiple components. A higher accuracy in the segmentation model also allows the scoring of the template matching step to be more reliable, leading to better results more easily, as the thresholds for fine-alignment and acceptable hypothesis can be tightened.

Secondly, the current CAD catalogue does not include all components and variants of components that can be found in catenary arches. Beyond just extending the CAD catalogue with more models, an approach to detecting and handling component variants should be developed towards later stages of the improvement of the currently presented pipeline.

One of the initial assumptions, that of a semantic segmentation model being satisfactory, has proven incorrect, or at least partially so. Ideally, in the future, an instance segmentation model should be added or placed in the semantic segmentation model's stead. Alternatively, various instance segmentation algorithms could be introduced depending on their reliability for the different components. Given the performance of such algorithms throughout this project, I suggest a deep learning or point pair feature based approached for the instance segmentation.

As mentioned in chapter 5.2.5, it may be worth considering the implementation of the fourth feature for the point pair feature-based template matching. It is currently set to zero and implementing it may raise the quality of matches while lowering the overall quantity, allowing for a larger number of iterations and therefore a higher chance to get good results for less computational power.

An improved scoring heuristic for the template matching step should be tested or developed. The current template matching scoring algorithm is computationally slow and is not neutrally interpretable between components, as the relationship between the score and the quality of the transformation hypothesis is currently heavily dependent on the accuracy of the semantic segmentation model as well as

the instance segmentation. To reliably automate this pipeline in the future, a more consistent method should be introduced.

Finally, the prototype pipeline is not fully automated in its current implementation. Due to the unreliability of the instance segmentation, this step is not automated for all components, and would have to be replaced as mentioned previously. Additionally, the actual CAD placement step using optimal transformation matrices has also not been automated in the scope of this project. It might be possible to do so using FreeCAD or potentially using other 3D scene manipulation libraries for python or other languages.

References

[1] D. Hofland, "125 jaar Amsterdam Centraal," Sanoma Media Netherlands, Amsterdam, 2014.

[2] Strukton, "Strukton Rail," [Online]. Available: https://strukton.com/en/rail. [Accessed September 2021].

[3] ProRail, "ProRail," [Online]. Available: https://www.prorail.nl/. [Accessed September 2021].

[4] Saxion, "Ambient Intelligence," Saxion University of Applied Sciences, [Online]. Available: https://www.saxion.edu/business-and-research/research/smart-industry/ambient-intelligence. [Accessed September 2021].

[5] B. A. Kitchenham, "Procedures for Undertaking Systematic Reviews," Software Engineering Group, Department of Computer Science, Keele University, Keele, 2004.

[6] Association for Computing Machinery, "ACM Digital Library," Association for Computing Machinery, [Online]. Available: https://dl.acm.org/. [Accessed September 2021].

[7] Alphabet Inc., "Google Scholar," Google, [Online]. Available: https://scholar.google.com/. [Accessed September 2021].

[8] Y. Guo, H. Wang, H. Liu, L. Liu and M. Bennamoun, "Deep Learning for 3D Point Clouds: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 2019.

[9] S. Bello, S. Yu, C. Wang, J. M. Adam and J. Li, "Review: Deep Learning on 3D Point Clouds," *Remote Sensing,* vol. 12, no. 11, p. 1729, 2020.

[10] X. Yao, J. Guo, J. Hu and Q. Cao, "Using deep learning in semantic classification for point cloud data," *IEEE Access,* vol. 7, pp. 37121-37130, 2019.

[11] C. R. Qi, H. Su, M. Kaichun and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, HI, USA, 2017.

[12] C. R. Qi, H. Su, M. Kaichun and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *Neural Information Processing Systems*, Long Beach, CA, US, 2017.

[13] X. Li, S. Guan, X. Li, J. Jin and J. Zhang, "A Local Feature Aggregation PointNet++ Network Based on Graph Network," in *The 4th International Conference on Image and Graphics Processing*, Sanya, China, 2021.

[14] M. Jaritz, J. Gu and H. Su, "Multi-view pointnet for 3d scene understanding," in *IEEE/CVF International Conference on Computer Vision Workshops*, 2019.

[15] A. Komarichev, Z. Zhong and J. Hua, "A-cnn: Annularly convolutional neural networks on point clouds," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[16] Y. Liu, B. Fan, S. Xiang and C. Pan, "Relation-shape convolutional neural network for point cloud analysis," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[17] B. Graham, M. Engelcke and L. Van Der Maaten, "3d semantic segmentation with submanifold sparse convolutional networks," in *IEEE conference on computer vision and pattern recognition*, 2018.

[18] E. Dijkstra, "Segmenting small objects in large point clouds using the Super Point Graph model," Ambient Intelligence Research Group, Saxion University of Applied Sciences, University of Twente, Enschede, 2021.

[19] L. Landrieu and M. Simonovsky, "Large-scale pointcloud semantic segmentation with superpointgraphs," in *IEEE Conference on Computer Vision and Pattern Recognition*, Salt Lake City, Utah, US, 2018.

[20] J. Wu, J. Jiao, Q. Yang, Z. J. Zha and X. Chen, "Ground-aware point cloud semantic segmentation for autonomous driving," in *27th ACM International Conference on Multimedia*, Nice, France, 2019.

[21] H. Kim, C. Yeo, I. D. Lee and D. Mun, "Deep-learning-based retrieval of piping component catalogs for plant 3D CAD model reconstruction," in *Computers in Industry*, 2020.

[22] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov and A. Smola, "Deep sets," in *Neural Information Processing Systems*, Long Beach, CA, US, 2017.

[23] F. Remondino, " From point cloud to surface: the modeling and visualization problem," in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2003.

[24] Y. Zhang, S. Song, P. Tan and J. Xiao, "Panocontext: A whole-room 3d context model for panoramic scene understanding," in *European conference on computer vision*, Zurich, Switzerland, 2014.

[25] R. Vock, A. Dieckmann, S. Ochmann and R. Klein, "Fast template matching and pose estimation in 3D point clouds," *Computers & Graphics,* vol. 79, pp. 36-45, 2019.

[26] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, C. Shearer and R. Wirth, "CRISP-DM 1.0: Step-by-step data mining guide," 2000.

[27] 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, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, 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,* 2015.

[28] P. Ramachandran, B. Zoph and Q. V. Le, "Searching for Activation Functions," arXiv preprint arXiv:1710.05941, 2017.

[29] D. Girardeu-Montaut, *CloudCompare,* 2020.

[30] M. Labussiere, J. Laconte and F. Pomerleaur, "Geometry Preserving Sampling Method Based on Spectral Decomposition for Large-Scale Environments," University Clermont Auvergne, University Laval, 2020.

[31] E. Nezhadarya, E. Taghavi, R. Razani and B. L. J. Liu, "Adaptive Hierarchical Down-Sampling for Point Cloud Classification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[32] M. Salahmand, "Classification and Segmentation of Catenary Arches - Internship Final Report," Enschede, 2021.

[33] Q.-Y. Zhou, J. Park and V. Koltun, "Open3D: A Modern Library for 3D Data Processing," *arXiv:1801.09847,* 2018.

[34] B. Bakir, J. Jonkers, J. Teunissen, T. Tunc and R. Ahmed, "Research Report: Point Cloud Classification of Railway Structures," Enschede, 2021.

[35] A. Appleby and H. Senuma, "mmh3".

[36] J. Riegel, W. Mayer and Y. van Havre, *FreeCAD,* 2002.

[37] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant, "Array programming with NumPy," *Nature,* vol. 585, p. 357–362, September 2020.

[38] Nvidia, *CUDA,* 2007.

[39] T. Montaigu, *pylas,* 2018.

[40] G. Brown and T. Montaigu, *laspy,* 2012.

[41] Kitware, *CMake,* 2000.

Appendix A: Ethical Assessment Paper

1. Introduction

The purpose of this paper is the ethical assessment of the graduation project of student Zino J. Vieth of the University of Twente in the Winter semester of 2021/22. The topic of the graduation project is named as “Point Cloud Classification and Segmentation for Catenary Systems” and the research focus is the development of a data pipeline for the reconstruction of 3D models from a point cloud-based railway scene for the creation of a digital twin.

First, the project will be introduced in detail and key concepts will be explained. Then, starting in chapter 3, overarching ethical dilemmas will be discussed regarding their relevance to the specific project, before design decisions and their moral basis in the form of ethical codes and principles is investigated. The project will then be put in relation to the UN’s sustainable development goals. Lastly, the project’s impact, limitations, and implications for the future will be briefly concluded based on the previous discussions.

2. Project Description

The Dutch railway network is the pillar of public and ware transport in the Netherlands, making it an invaluable piece of infrastructure that many people rely on daily. This reliance makes failures and delays within this system all the more impactful and noticeable to the quality of life, but also financial ventures of Dutch citizens. With around 6830 km of track as of 2014 [1], the maintenance and monitoring of the entire system has become a mammoth task, difficult to handle manually, as it requires trained experts to identify broken equipment, or those that might be at risk of faulty behaviour in the near future. To improve the efficiency and safety of the network, the Dutch railway companies ProRail [2] and Strukton Rail [3] have begun an extensive digitalization program.

One of the fields of focus for Strukton Rail is the digitalization of the catenary system, which refers to the system of overhead electricity lines and the corresponding arches and equipment. The long-term

goal is the creation of a digital twin[1] of the entire catenary system, which would help with the documentation and if consistently updated via real-world scans with the monitoring and maintenance of the system. Strukton Rail tasked the Saxion University of Applied Science's Ambient Intelligence Research group (AMI) [4] with the development of relevant technologies for this goal. The graduation project in question lies within the scope of the development of these technologies.

The graduation project's goal is the development of a data pipeline[2] for the reconstruction of 3D models from point cloud[3]-based railway scenes generated via LiDAR[4] scans. This project can be divided into two phases: First, the implementation of a deep-learning[5] model for segmenting[6] the point clouds of catenary arches into the various components of the arch; Second, the replacement of each object in the point cloud with a correctly aligned 3D model provided by Strukton. Together, these major steps form a data pipeline which takes a LiDAR scan of a catenary arch as input and returns a 3D model of the same arch as an output. Given the fidelity of the data provided for this project and the state of the art of 3D point cloud segmentation models, which is still in a developmental phase, this project aims to deliver a prototype able to show that the chosen methods are applicable for the creation of a digital twin on this scale. Appendix B shows one of the primary outcomes of this project: the draft of the data pipeline for the reconstruction of 3D models from point cloud-based railway scenes. The other outcomes are in the form of the code actually implementing this pipeline, and the associated report detailing the workings of the pipeline.

---

[1] Digital Twin – "a virtual representation that serves as the real-time digital counterpart of a physical object or process" (Wikipedia, as of Jan 2022). In this case, a digital representation of the Dutch railway's catenary system via 3D models.

[2] Data Pipeline – "a set of data processing elements connected in series, where the output of one element is the input of the next one" (Wikipedia, as of Jan 2022).

[3] Point Cloud – "a set of data points in 3D space" (Wikipedia, as of Jan 2022).

[4] LiDAR (Light Detection and Range) – "a method for determining ranges by targeting an object with a laser and measuring the time for the reflected light to return to the receiver" (Wikipedia, as of Jan 2022). Strukton used 360° LiDAR scanners mounted on trains to capture point clouds of railway scenes including 13 catenary arches to be used for this project.

[5] Deep Learning – "a part of a broader family of machine learning methods based on artificial neural networks with representation learning" (Wikipedia, as of Jan 2022). Essentially, a method by which a computer is allowed to learn the features and patterns of data by providing it with a lot of it. Based on the generated understanding of these patterns, a trained deep learning model is able to predict an output for a specific input.

[6] Semantic Segmentation – "the process of assigning a label to every pixel in an image" (beyondminds.ai, as of Jan 2022), or every point in a point cloud, so as to associate that point with a specific object, for example the poles of the catenary arches, or other components such as insulators, wires, etc.

On one hand, the client of this project is Strukton Rail, and the data pipeline is more specifically developed for use by their data scientists. On the other hand, this project aims to deliver a prototype of a data pipeline, the feedback of which the researchers of AMI can use to continue developing the best possible technologies for the digitalization of the railway's catenary system. It is therefore not the purpose of this project to deliver a final product, interface, or program testable for usability or ethical impact.

In a previous project, AMI has begun the development of a PointNet++-based semantic segmentation model and tested it against various other state of the art models, such as the Super Point Graph model and the Point Transformer model. In that process, various preprocessing steps have been performed on the dataset provided by Strukton Rail, such as cleaning[7], isolating the thirteen catenary arches into separate files, labelling[8], down-sampling[9], and augmenting[10] the data.

In my part of the project, I am working with this preprocessed data, and reimplement the deep learning model developed by AMI into the data pipeline in the first phase of this project. In the second phase, completely new technologies are introduced and tried for their applicability to the 3D model reconstruction pipeline. I work on this project, the entire data pipeline, autonomously as an intern at AMI, with the supervision and assistance of their researchers where required. Appendix A shows the structure of the project at AMI and where my work connects to theirs.

My personal interest in this project stems from its technical nature and its relevance to public infrastructure. I believe that public transport is going to be one of the public sectors with the most exciting and observable changes in the future, which is why I decided to dip my toes into this vast field through this project. While this project is very much under the hood and does not include any significant user interactions, the changes that deep learning technologies will bring to transportation will greatly influence how people behave and move around their cities and countries.

From a Creative Technology standpoint, this project is a little bit atypical, as it does not really contain a human-centered component, but I decided it is best to take the opportunity to look into a more

[7] Data Cleaning – the process of removing irrelevant or faulty points from the point clouds, in this specific example, such as the scanned trees and foliage next to the tracks.
[8] Data Labelling – the manual process of assigning a label to each point in the point cloud. Each label represents an object, such as a pole or insulator.
[9] Down-Sampling – the usually heuristic-based process of selecting data points to actually be used by the machine learning algorithms, as to reduce the computational load in comparison to using the entire dataset.
[10] Augmentation – the generation of additional, slightly modified data sets from existing data to enhance the training phase of a model by providing it with more unique arches to learn from.

technical topic while I am still at university, as I've always been most interested in the technical components of previous projects, but rarely had the chance to go into depth with it.

3. Discussion

In this chapter, ethical and philosophical issues related to this graduation project will be articulated, their impact on the design process discussed, and possible solutions analyzed. Additionally, this projects correspondence with the UN Sustainable Development Goals will be considered.

Generally, the lack of public end consumers and the nature of a data pipeline prototype would suggest relatively little direct personal impact. Yet, the larger ramifications and consequences of the project and the field of digitalization that it underlies, as well as the project's impact on the client, the data scientists of Strukton Rail and researchers of AMI will be considered.

3.1 Ethical Dilemmas

Commonly, the topic of artificial intelligence raises a variety of notoriously debated ethical dilemmas, few of which it is feasible to discuss to a satisfactory degree while maintaining the scope set by this graduation project. Yet, there are two for which the nuances presented by this application of artificial intelligence are worth discussing: the dilemma of automation and artificial intelligence replacing humans in the workforce, and that of the AI black-box. Beyond those, I will discuss this project's potential economic ramifications, and the pitfall of documentation in the form of malevolent use of it.

First, artificial intelligence and automation technologies have been reducing the need for manual human labour and pushed numerous professions to extinction. According to McKinsey [5], hundreds of millions of jobs will be lost to automation by 2030, and the improvement of artificial intelligence and computer vision technologies that this project relies on are certainly an important factor in that development. The task of manually monitoring the catenary system of the Dutch railway is likely to become obsolete as the result of the greater vision that this graduation project belongs to. Yet, this task was previously done on a considerably low and inefficient scale as the railway simply does not have access to enough experts to monitor the entire network sufficiently. At the same time, the experts that would previously manually monitor the catenary system would still be required to do so in the future, simply using the Digital Twin of the catenary system as a tool to do this task more effectively and on a larger scale. While it is not impossible that the railway companies may have a decreased need for catenary

system experts, this profession would not be entirely replaced by the outcome of this project. Concurrently, new jobs for data scientists and IT personnel may be generated to improve and maintain the digital twin. Therefore, I conclude that the upsides of this project, improved efficiency and safety of the railway network and generation of jobs for data scientists and IT professionals far outweighs the minimal loss of manual labour jobs.

Second, the dilemma of the AI black-box has fairly little ethical implications for this project. The AI black-box problem is defined by the lack of interpretability or understanding of an outsider in the operations of an artificial intelligence model. For one, this problem has severe implications on the public's trust and acceptance of artificial intelligence in their everyday life [6], but also on the question of accountability for an AI's failure. If the AI is the only entity able to understand the reasoning of its decision, who but the AI itself should be held responsible for this decision? A detailed discussion of this problem is fortunately not applicable to this project, as the AI involved in this project, in the form of a semantic segmentation deep-learning model, does not make actually impactful or end-result deciding decisions. It acts merely as a single process in a data pipeline to decrease the amount of manual labour required. This quickly points at a design need for the data pipeline: it needs to be designed to allow data scientists to correct for inaccurate predictions by the AI and manually add or modify missing parts in the digital twin. This can be easily done by comparing the scanned catenary arch (which also serves as the input for the data pipeline) and compare it visually to the output 3D model. Additionally, spatial approximation algorithms can automatically detect whether certain regions of the input point cloud have not been replaced by a template 3D model and alert the data scientists operating the pipeline as such. Therefore, the AI black-box dilemma has no real ethical ramifications, as the data-scientists do not need to rely on trusting the AI's predictive ability in this specific application of AI.

The third notable ethical consideration relevant to this topic in general is that of the economic consequences of improving the reliability and efficiency of the railway system and ware transport in general. Generally, it could be argued that improving the transport infrastructure may benefit already large corporations who can easily corner the market in local communities due to their advantage of production efficiency, if the transportation barriers are further removed. At the same time, research by Gaus and Link [7] as well as Polyzos and Tsiotas [8] suggests that high quality and quantity of transportation infrastructure positively effects regional economies and their potential for growth. While concerns for the sustainable development of a region are warranted, according to Prus and Sikora [9], transportation and globalization are trends bound to continue developing, and this project or even the

digitalization project overall are unlikely to cause any notable negative consequences, if at all. On the other hand, the quality of life improvements and financial opportunities that will be generated by a more efficient and reliable railway system are likely to be significant. Therefore, this project is unlikely to affect the overall trend of development of transportation infrastructures, but aims to simply improve the existing system, which will have next to no negative economic ramifications.

The last ethical dilemma to discuss is the argument that improved documentation may make the railway network more accessible for malevolent intentions, for example the more effective manipulation or damaging of railway equipment, in this case overhead electricity equipment. While it is true that a well-documented system may be easier to damage should individuals with malevolent intentions get hold of this documentation, but the system is also more easily repairable, and damage can be more quickly identified through digitalization. Additionally, the documentation, in the form of a digital twin is, as far as the development team is concerned, not meant to be a public asset, but to be used exclusively by the companies in charge of monitoring, maintaining, and developing the network. This fact alone suggests that this is more of a security issue than an ethical dilemma. Almost any tool designed to help with a task may be used to harm in the hands of malevolent individuals. Therefore, the ways in which the digital twin may be used malevolently are neglectable and likely to be negated by its positive effect on maintenance and monitoring.

Finally, it is worth noting that the LiDAR scanners employed to capture the catenary arches in this project do not pose a privacy threat, as they are not of photographic nature and individuals that are possibly captured by the scanners while near the tracks are not identifiable. Hence, there is no ethical privacy issue to be discussed.

3.2 Ethical Codes and Moral Principles

Ethical codes are a useful tool for decision making and design processes in engineering projects. Given the discussion of ethical dilemmas, the project itself may not seem to be of significant ethical consequence, but there is still a need to consider ethical behaviour and decision making in many minor aspects of the project. While the ethical dilemmas highlighted overarching ethical questions and the ethical soundness of the intention of the project, the employment of an ethical code and key moral principles may help to make important decisions within the project itself. Though first one must choose a code that reflects the nature of the project well.

This engineering project can be considered to lie within the fields of data science and software engineering, for both of which specific codes exist. When comparing a general engineering code such as the IEEE Code of Ethics [10] and field specific codes such as the joint ACM/IEEE-CS Software Engineering Code [11] or the Data Science Association's Data Science Code of Professional Conduct [12], it quickly becomes clear that the field specific codes consider more typical activities of their field more closely, while the IEEE code is very general about upholding "the highest standards of integrity, responsible behaviour, and ethical conduct in professional activities" [10]. The software engineering code for example highlights the importance of adequate testing, debugging and reviewing of software, as well as documentation of code and tracking of errors and bugs [11]. The data science code of professional conduct on the other hand outlines the correct use of data science terminology and statistical methodology [12]. Both of these specifications are significant for the ethical completion of this graduation process, as correct and accurate documentation as well as truthful and precise reporting of statistical results are paramount to the client–student relationship of this project, as will be further discussed by the use of moral principles.

The majority of code of ethics are based on a certain set of moral principles which can be applied in different ways to specific professional fields. One of the original works on moral principles by Kitchener [13] outlined the five principles of autonomy, non-maleficence, beneficence, fidelity, and justice, with veracity being introduced by later works, such as that of Meara, Schmidt and Day [14]. In consideration of the Data Science Code of Professional Conduct [12] and the Software Engineering Code [11] I determined all but justice to be a moral principle worth discussing in the scope of this project.

The principle of autonomy refers to respecting a person's right to make his or her own decisions. In this project, this mostly translates to my responsibility as an engineer to provide the client with code and documentation which allows them to autonomously manipulate and change the data pipeline to their needs or change in requirements at a future date. It is therefore important to deliver well documented and flexible code, especially in terms of data transformations performed throughout the pipeline.

The principle of beneficence refers to the intention of acting to the benefit of another. Since the researchers of AMI and data scientists at Strukton Rail are the target audience/users of the project's outcome, they should be the primary recipients of this beneficence. The data pipeline should therefore be designed in a way that makes their life easier. This may manifest itself by providing dynamic input options, clear documentation, but also by not requiring specific and manual-work intensive data formats for inputs and outputs.

The principle of non-maleficence has a similar nature of that of beneficence, but more specifically states the developer's intention to do no harm. In the case of this project, this can be achieved by considering each design decision from the perspective of the target audience and ask "How will this decision affect the data scientist or researchers?". For the most part, this has affected design decisions regarding inaccuracies and failure cases. For example, if the AI is unable to detect a certain number of points for a component that would normally have a lot more points, the data scientists need to be informed of potential inaccuracies in the detection of this component. Similarly, if no 100% accurate transformation hypothesis can be generated for the placement of a 3D model in the 3D scene, this inaccuracy must be clearly labeled to make sure that the data scientists have the opportunity to correct it.

The principle of fidelity refers to the faithfulness of the developer to the client. It is important that the client's requirements are considered with at least as much importance as the developer's own opinions and possible conflicts are discussed and resolved with an agreement on an appropriate course of action. In the case of this project, very few concrete requirements have been provided by the client besides the general idea of the data pipeline, its input and output, as well as the data given for training and testing. It is therefore mostly necessary to stay faithful to the format of the provided data and not modify to a degree that cannot be feasibly incorporated in the data pipeline itself.

The principle of veracity refers to truthfulness. In the case of this project, truthfulness is especially important in the documentation of the pipeline's limitations. Potential quirks and inaccuracies in the various employed algorithms and heuristics need to be detailed, to ensure that future work is equipped to deal with them. Similarly, the different use cases and the pipeline's performance on each of them has to be truthfully reported, as do its tested or untested limitations on specific datasets or components of the catenary arches. For example, majority of the development for phase two has been done with the pole component in mind. The potential limitations when using this pipeline on significantly differently shaped components must be clarified.

3.3 Ethical Analysis

3.3.1 The Accountability Problem

The problem of accountability becomes quite complex once artificial intelligence is introduced as an autonomous decision maker into a system. However, in the case of this project, the accountability for

failure cases at different points along the data pipeline can be clearly sketched via flowchart analysis as introduced by Fleddermann [15].

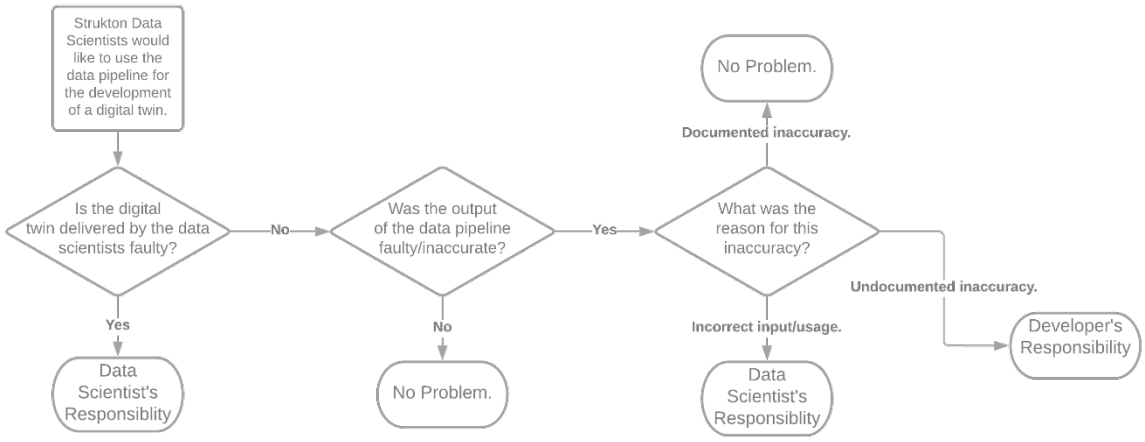

Figure A1: Flowchart Analysis of Failure Accountability

In the figure above (*Figure 1*), the responsibilities for failure cases between the developers of the data pipeline (aka me) and the users of the data pipeline, the data scientists at Strukton rail, are sketched out. The first decision clarifies that the developer is not responsible for faults in the product of the end result for which the data scientists have been using the tool. The only path to a developer's responsibility in this project is if the data pipeline produces inaccurate results due to undocumented inaccuracies/quirks within the data pipeline. In the case of this project, there is no case in which responsibility can be assigned to the AI/deep-learning model, as its accuracies are either undocumented, in which case it is the developer's fault, or documented, in which case there is no failure reportable beyond the expected margin of error.

3.3.2 Influence on Local Economies

Previously, the potential influence of this project's improvement to transport infrastructure's efficiency and safety on local economies has been discussed and determined to be negligible. Yet, it may be worth to consider the potential scenarios and their ethical assessment. To do so, Fleddermann's line drawing analysis [15] can be employed. Essentially, a scale between a positive and a negative paradigm is drawn, on which the expected problem and scenarios can be placed in relation to each other.

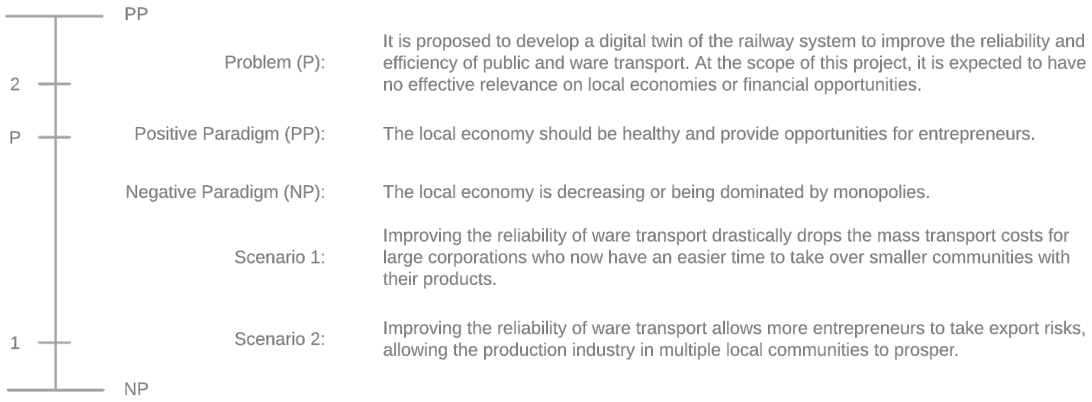

Figure A2: Line Drawing Analysis of Economic Impact

3.4 UN Sustainable Development Goals

In a globalized world, in which the environmental and social impact of individual companies outdoes that of entire countries [16], global guidelines for ethical and sustainable development are becoming increasingly relevant. To provide such guidelines, the UN has published a list of 17 goals as a call to action for companies and governments to ensure the preservation of our home and the equal treatment and availability of opportunity to all humans [17]. Aligning the development of new technologies with these goals should therefore be a universal concern. This project more or less directly relates to three of the 17 sustainable development goals (SDGs): “Industry, Innovation, and Infrastructure”, “Sustainable Cities and Communities”, and “Responsible Consumption and Production”.

Goal 9 of the SDGs states to “build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation” [17]. This project aims directly at improving the reliability and efficiency of transport infrastructures. Within this goal, it will especially aid with target 9.1, which states to “develop quality, reliable, sustainable, and resilient infrastructure, including regional and transborder infrastructure, to support economic development (…)” [17]. More reliable transport is likely to open up a variety of economic opportunities, while increasing the efficiency of maintenance and monitoring of the railway system will make it more resilient and sustainable.

Goal 11 of the SDGs states to “make cities and human settlements inclusive, safe, resilient and sustainable” [17]. Reliable local train transport is an important factor for the quality of life and sustainability of cities. Target 11.2 of this goal states to “provide access to safe, affordable, accessible and sustainable transport systems for all” [17]. The improvement of monitoring and maintenance systems for

the railway network will inevitably make local transport more affordable and sustainable by reducing system failures and delays through early damage detection.

Goal 12 of the SDGs states to “ensure sustainable consumption and production patterns” [17]. In the railway network, nearly-expired or damaged equipment will commonly only be detected by the time it has caused a system failure, potentially damaging other components in the process. By monitoring the entire network via LiDAR scanners and deep-learning technologies, any potentially expiring equipment can be identified early and fixed, which will reduce the number of new components that need to be manufactured and therefore also reduce the resource consumption of the railway network as a whole.

4. Conclusions

4.1 Impact Statement

This project stands as a prototype for the process of developing a digital twin of the catenary system of the Dutch railway, a key process in the digitalization efforts of Strukton Rail, the client of this project. Ethically, no major concerns have been raised about this specific project nor the overarching digitalization undertaking, beyond the potential loss of manual labour jobs in the monitoring tasks.

As a driver for change, the outcome of this project will primarily function as a skeleton for the future development of a digital twin, based on this or modified implementations of the designed data pipeline. Its recommendation for technologies and their initial implementation should allow for rapid process in the development of a digital twin, once other hurdles, such as the low accuracies of deep learning models for the segmentation of catenary arches, have been overcome.

4.2 Limitations

On a technical level, the prototype is considerably limited in comparison to the overall vision of a pipeline for the development of a digital twin. For one, computing efficiency is not fully optimized, which may make large scale employment near impossible. Secondly, the deep-learning model implemented into the data pipeline is not trained for all possible components and their various versions and is significantly inconsistent in terms of the accuracy with which it identifies different components. For example, it performs significantly better on larger components such as poles than on smaller components such as insulators. Lastly, the dataset used for training and development of this pipeline was simply a small sample which underwent various preprocessing steps. Future versions of the pipeline would have to be adapted

to the datasets used at production-level, of which the structure and fidelity is not known at this point in time.

The client is aware of those limitations and the project was planned with them in mind, which leaves no ethical uncertainties in the client-student relationship of this project. The project is meant to be of developmental and experimental nature, of which the results should be provided in a shape usable to the researchers at AMI, which makes documentation and code clarity the number one ethical responsibility of this project.

4.3 Implications for the Future

Digitalization is a nearly unstoppable process that will slowly encompass all aspects of our lives over the coming years. In most cases, this could be considered for the better, for example in the infrastructure sector. Monitoring via digital twins will make transport for example significantly more reliable in the long term, and deep-learning and computer vision technologies employed in this project will play a major role in perfecting that. At the same time, the digitalization of cities and human life through AI will bring with a wave of ethical dilemmas, as well as privacy and security uproars. Just considering the access question brought up by the fourth ethical dilemma discussed earlier. In a digitalized world, how will access privileges be handled, and how will they influence the power dynamics of the world. In comparison to an analogue world, in which access to any kind of data or decision making generally comes with physical and time-consuming boundaries, these limitations may essentially be eliminated and raise the potential power individuals hold over information and decisions exponentially. In the context of railway's catenary systems, this may not be a significant concern though, so the implication of this digitalization project in particular is primarily positive because of it boosting the reliability and efficiency of the railway infrastructure in the Netherlands, which will improve the quality of life for residents, open up cheaper economic opportunities, and reduce the waste of resources used in the maintenance of the system.

References (Ethical Assessment)

[1] D. Hofland, "125 jaar Amsterdam Centraal," Sanoma Media Netherlands, Amsterdam, 2014.

[2] ProRail, "ProRail," [Online]. Available: https://www.prorail.nl/. [Accessed September 2021].

[3] Strukton, "Strukton Rail," [Online]. Available: https://strukton.com/en/rail. [Accessed September 2021].

[4] Saxion, "Ambient Intelligence," Saxion University of Applied Sciences, [Online]. Available: https://www.saxion.edu/business-and-research/research/smart-industry/ambient-intelligence. [Accessed September 2021].

[5] J. Manyika, S. Lund, M. Chui, J. Bughin, J. Woetzel, P. Batra, R. Ko and S. Sanghvi, "Automation Could Eliminate 73 Millio U.S. Jobs By 2030," McKinsey Global Institute, 2017.

[6] ThinkAutomation, "The AI black box problem," ThinkAutomation, 2019.

[7] D. Gaus and H. Link, "Economic Effects of Transportation Infrastructure Quantity and Quality: A Study of german Counties," DIW Berlin, Berlin, 2020.

[8] S. Polyzos and D. Tsiotas, "The Contribution of Transport Infrastructure to the Economic and Regional Development: A Review of the Conceptual Framework," *Theoretical and Empirical Researches in Urban Management,* vol. 15, no. 1, pp. 5-23, 2020.

[9] P. Prus and M. Sikora, "The Impact of Transport Infrastructure on the Sustainable Development of the Region - Case Study," *MDPI Agriculture,* vol. 11, no. 4, p. 279, 2021.

[10] IEEE, "IEEE Code of Ethics," IEEE, 2020.

[11] D. Gotterbarn, K. Miller and S. Rogerson, "Software engineering code of ethics," *Commun. ACM,* vol. 40, no. 11, pp. 110-118, 1997.

[12] Data Science Association, "Data Science Code of Professional Conduct," Data Science Association, 2022.

[13] K. S. Kitchener, "Intuition, Critical Evaluation and Ethical Principles: The Foundation for ethical decisions in counseling psychology," *The Counseling Psychologist,* vol. 12, no. 3, pp. 43-55, 1984.

[14] N. M. Meara, L. D. Schmidt and J. D. Day, "Principles and Virtues: A Foundation for Ethical Decisions, Policies, and Character," *The Counseling Psychologist,* vol. 24, no. 1, pp. 4-77, 1996.

[15] C. B. Fleddermann, Engineering Ethics, Upper Saddle River, NJ: Prentice Hall, 1999.

[16] A. Sguazzin, "The World's Biggest Emitter of Greenhouse Gases," Bloomberg Green, 2020.

[17] United Nations, "The 2030 Agenda and the Sustainable Development Goals," *United Nations publication,* 2018.

Appendix B: Flowchart Legend

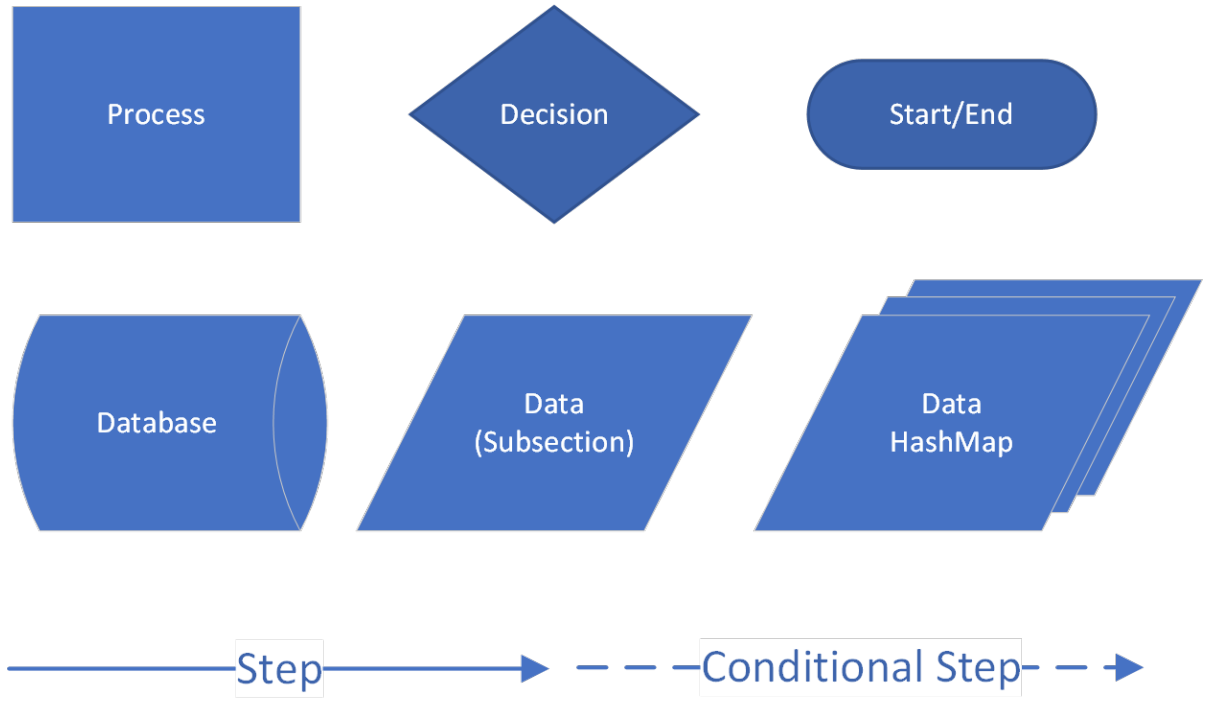

Figure B1: Flowchart Legend

Appendix C: All Catenary Arch Point Clouds

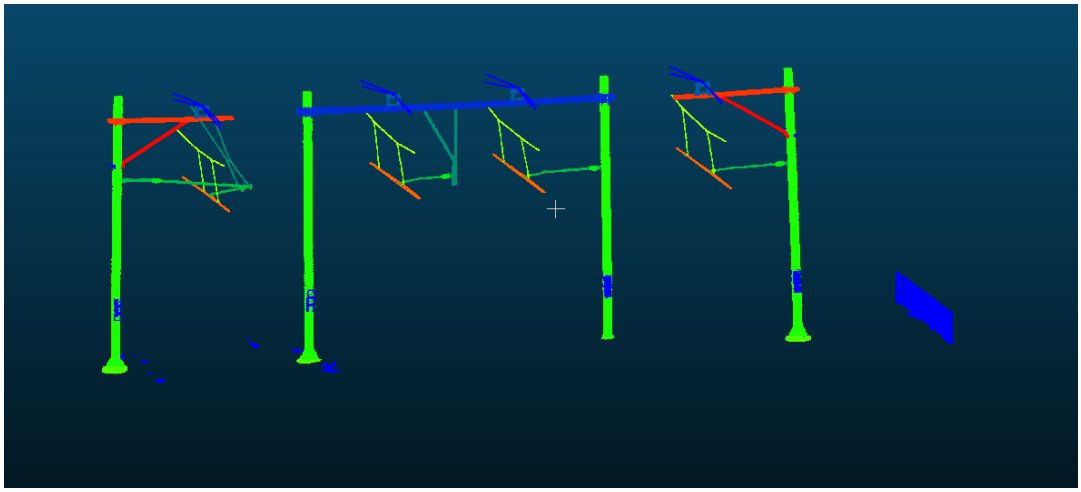

Figure C1: 01_01.laz

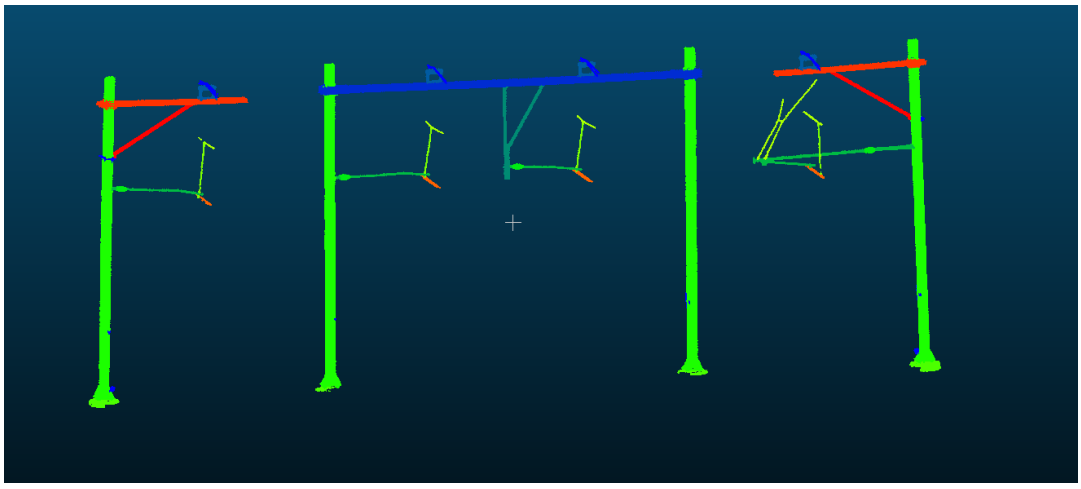

Figure C2: 01_02.laz

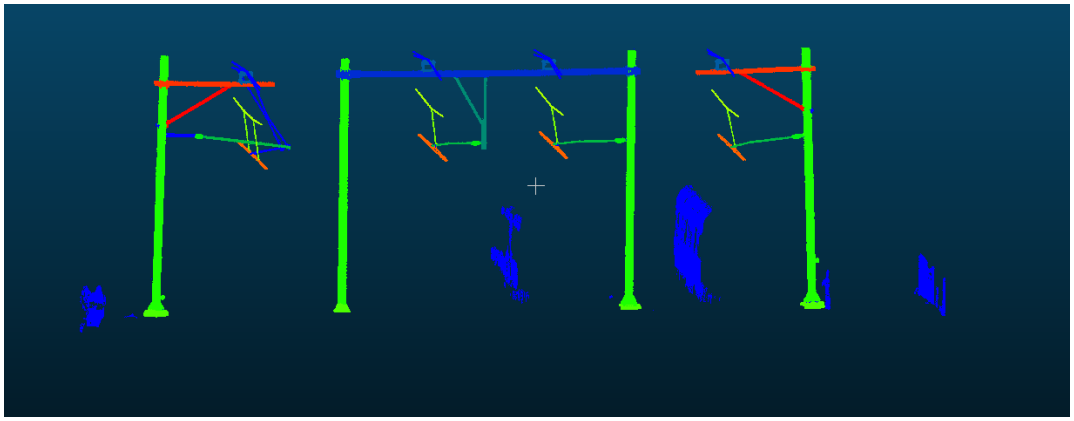

Figure C3: 01_03.laz

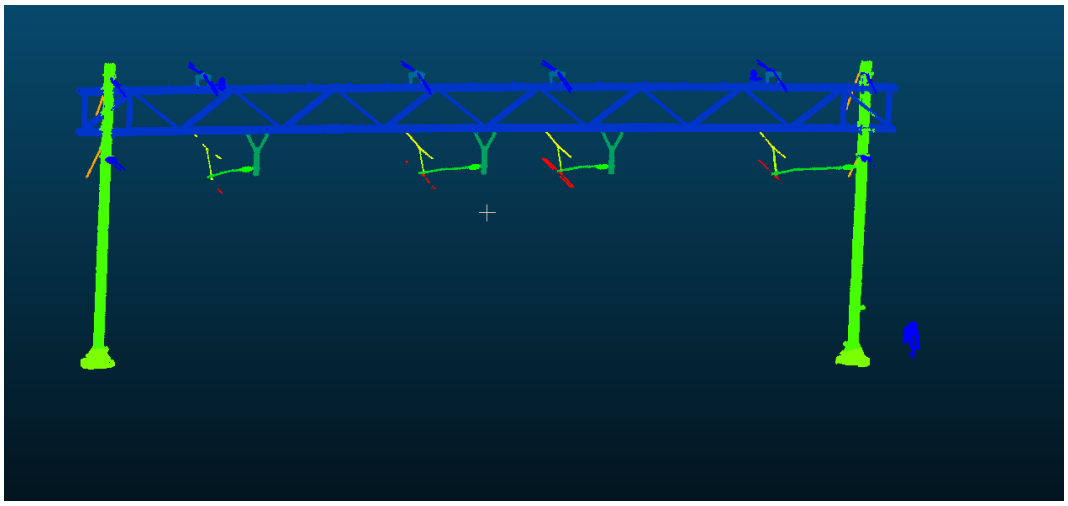

Figure C4: 02_02_bram.laz

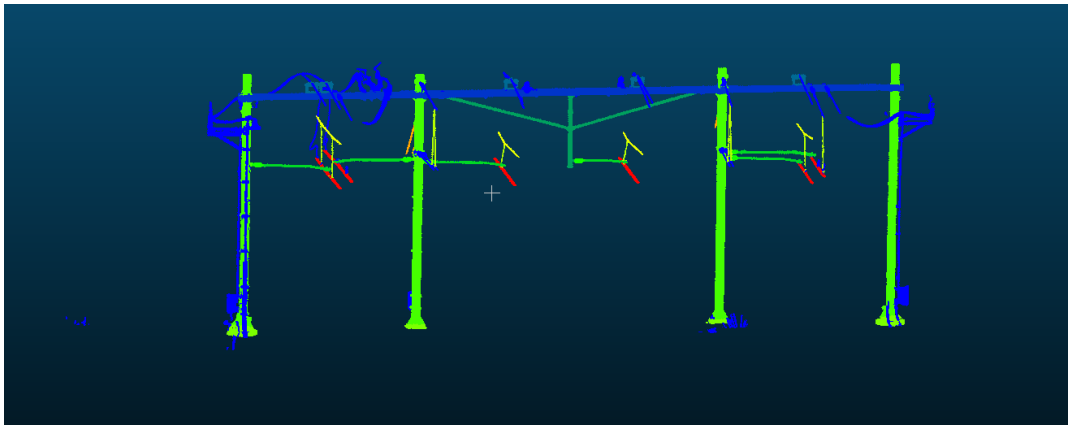

Figure C5: 02_03.laz

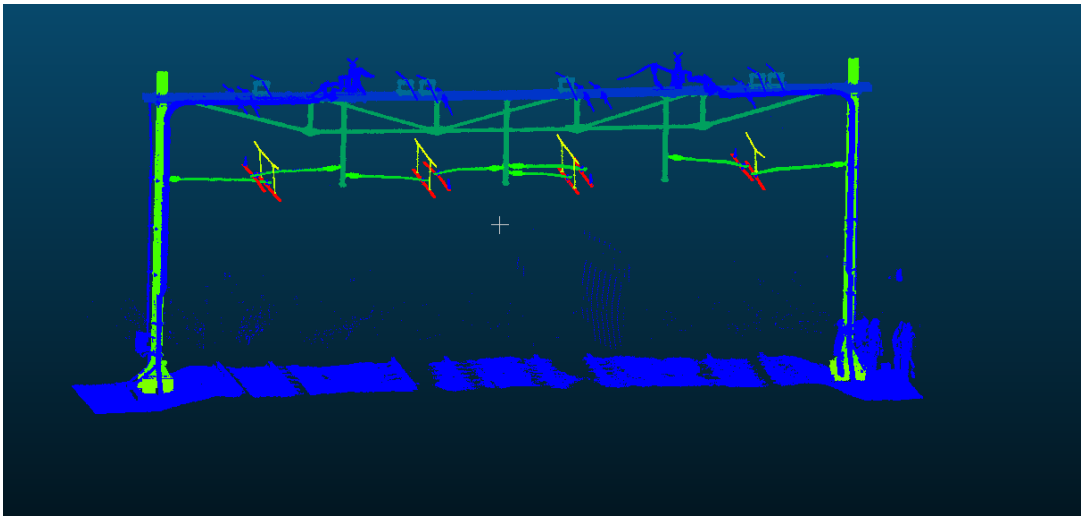

Figure C6: 02_04_bram.laz

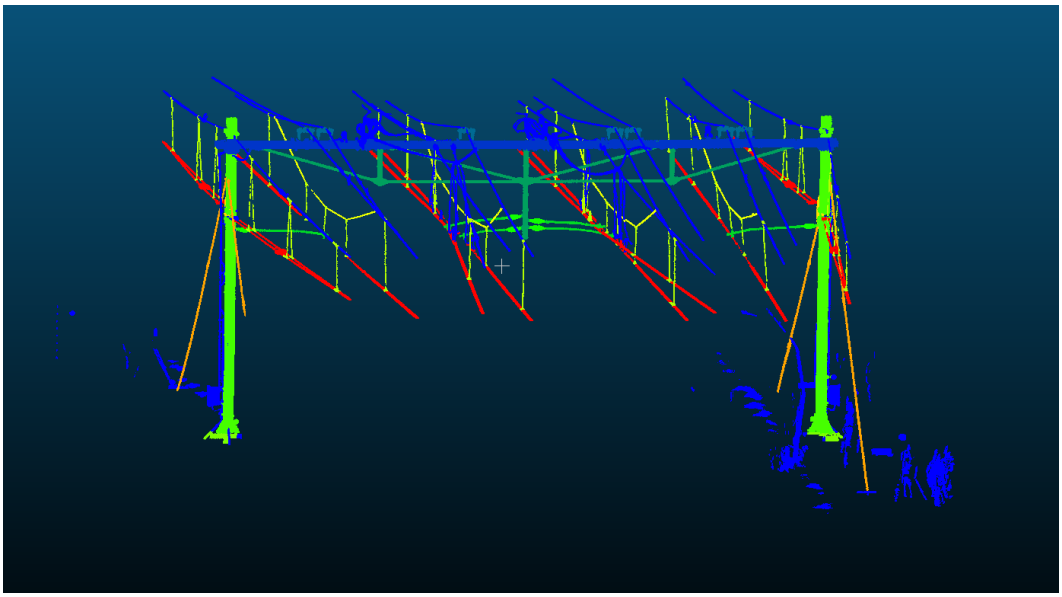

Figure C7: 03_01.laz

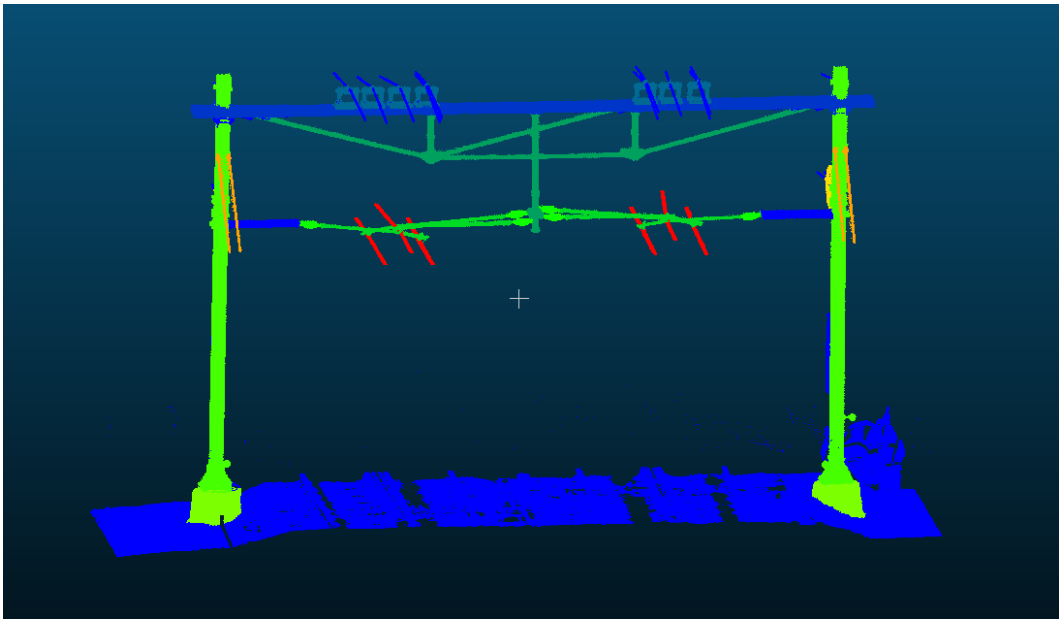

Figure C8: 03_03_bram.laz

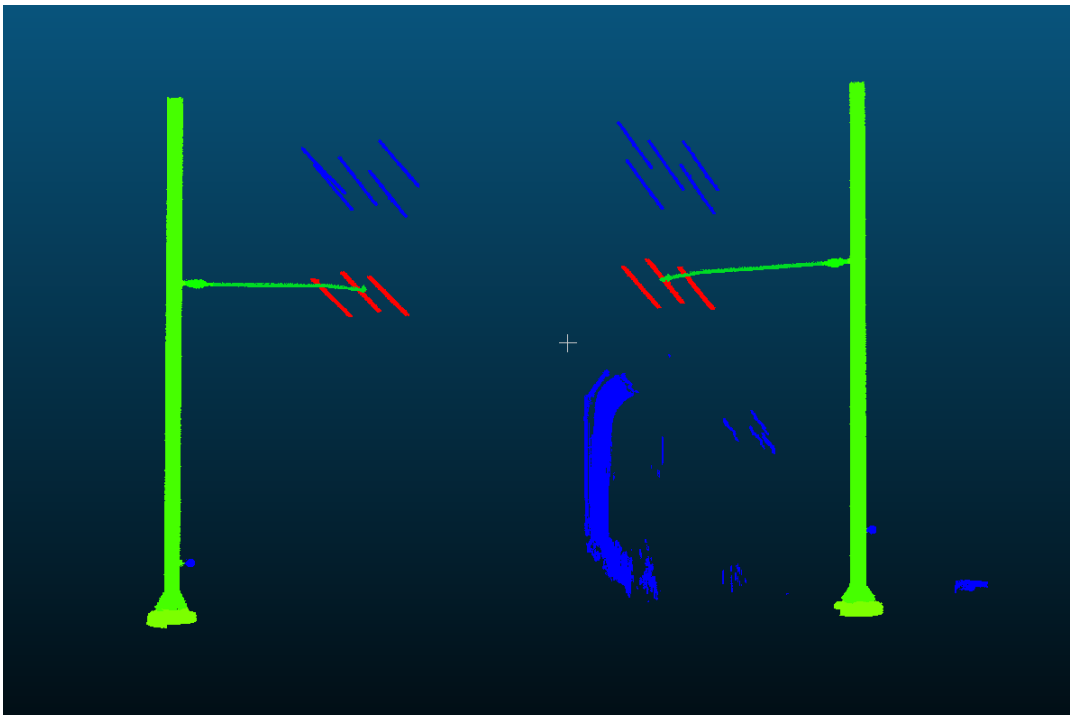

Figure C9: 03_03.laz

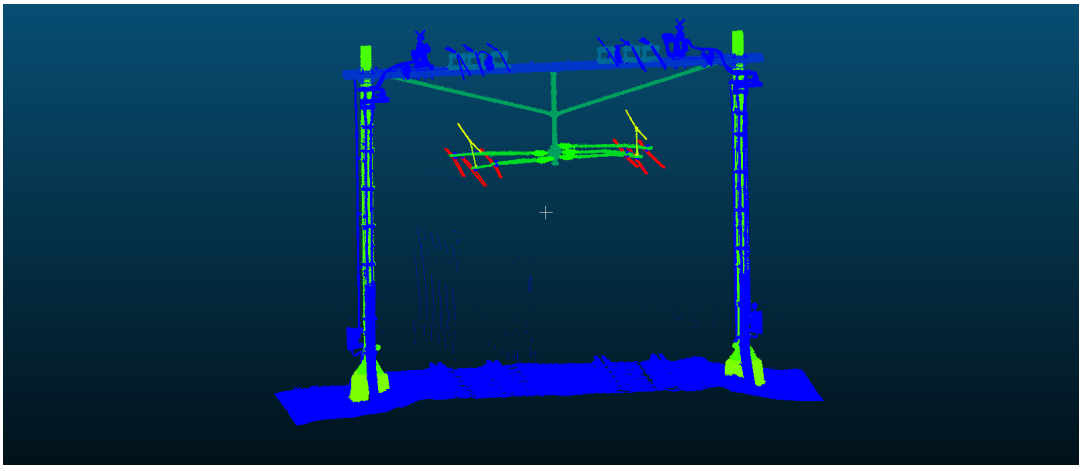

Figure C10: 03_04_bram.laz

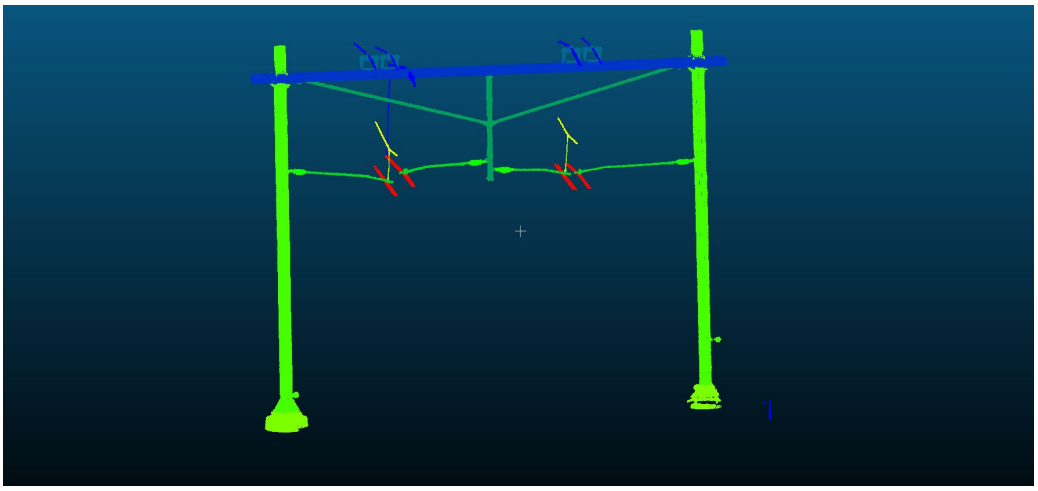

Figure C11: 04_01.laz

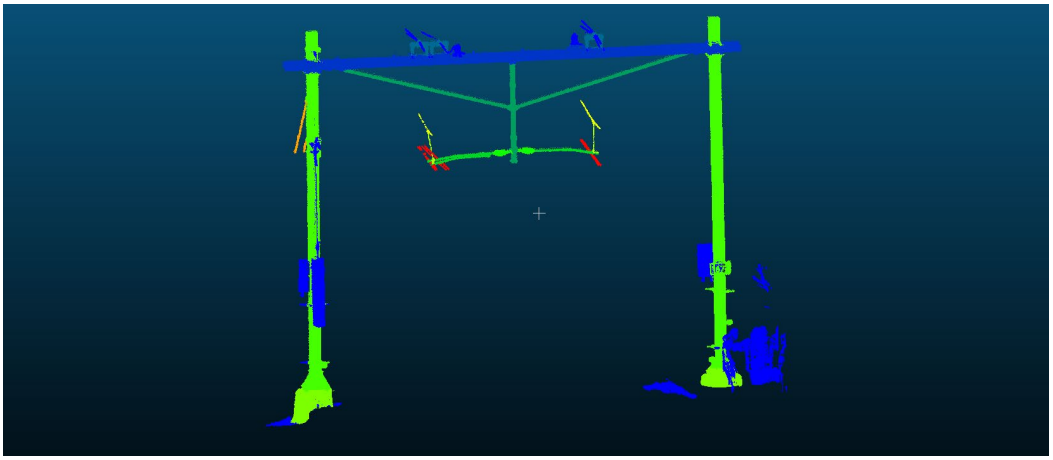

Figure C12: 04_02_bram.laz

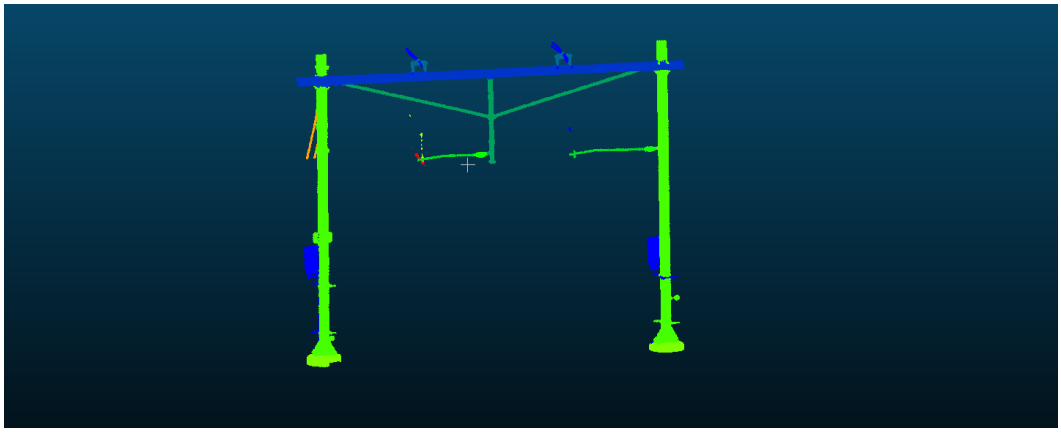

Figure C13: 04_03_bram.laz

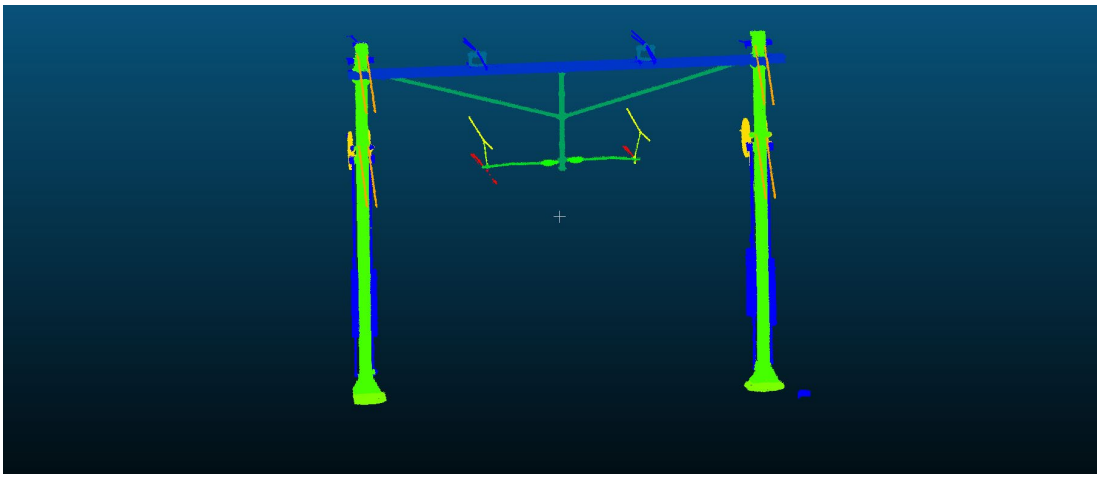

Figure C14: 04_04_bram.laz

Appendix D: Used CAD Models

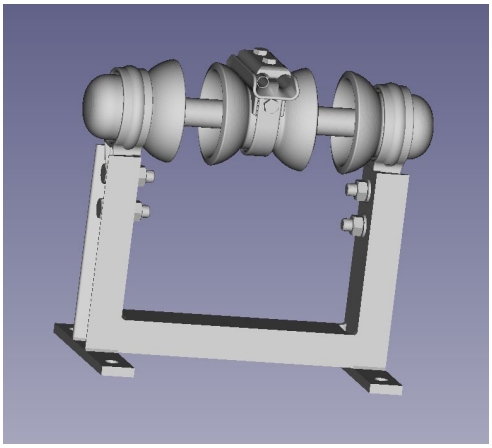

Figure D1: CAD Model of Messenger Wire Support

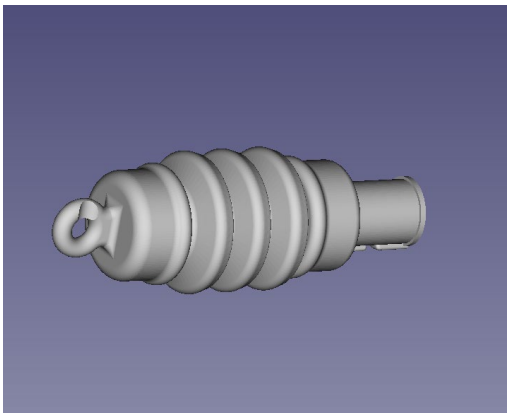

Figure D2: CAD Model of Insulator

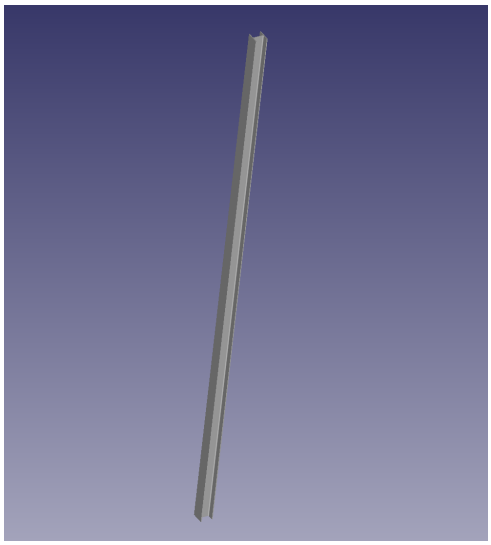

Figure D3: CAD Model of Pole

Appendix E: Template Matching Notebook

```
%reload_ext autoreload
```

```
%autoreload

import pickle
import time
import numpy as np
from ppf import Labels, load_scene, load_instances, load_templates, normalize, save_pc,
    generate_normals, voxel_down_sample, pc_resolution, bounding_box_diameter,
    segment_instances, retrieve_ppf, generate_hypothesis, point_region_overlap,
    apply_transformation, fine_align, add_translation

# Imports and settings
INFILE = '03_02_bram_pred.laz'
INTYPE = Labels.PREDICTED
DATA_DIR = '/home/jupyter-zino/Data/data/data_train/'
TEMPLATE_DIR = '/home/jupyter-zino/Data/data/templates/'
INSTANCE_DIR = '/home/jupyter-zino/Data/data/instances/'

MAX_PAIRS = 2000
MAX_RANSAC_ITERATIONS = 10

INSTANCE_INPUT = { # if this is none, instances will be automatically segmented
    2: '03_02_bram_pred_2.laz',
    5: '03_02_bram_pred_5.laz',
    6: None
}

SAVE_SCALE = {
    2: 0.000000001,
    5: 0.0000001,
    6: 0.000000001
}

TARGET_OVERLAP = {
    2: 0.01,
    5: 0.1,
    6: 0.5
}
FINE_TARGET_OVERLAP = {
    2: 0.25,
    5: 0.25,
    6: 0.975
}

COLOURS = [
    [255, 0, 0], # red
    [0, 255, 0], # green
    [255, 255, 0], # yellow
    [0, 255, 255], # acqua
    [255, 95, 0], # orange
    [255, 175, 255], # pink
    [135, 175, 255], # dark green
    [221, 0, 210] # purple
]

VOXEL_SIZE = {
    2: 0.005,
    5: 0.001,
    6: 0.01
}
```

```
INSTANCE_DISTANCE = {
    2: 0.0365, # not used
    5: 0.05,  # not used
    6: 0.5
}


def print_transform(matrix):
    for row in matrix:
        print(f"{row[0]} {row[1]} {row[2]} {row[3]}")
```

```
# Main
%autoreload

if __name__ == "__main__":

    # load template point clouds
    labels_template, template_points = load_templates(TEMPLATE_DIR)
    print("-> Loaded templates for labels", labels_template)

    # load segmented scene point clouds
    labels, seg_points = load_scene(DATA_DIR, INFILE, INTYPE)

    for l in labels:
        print("------------------------------------------------------------------------")
        print("--Running template matching and pose estimation for object with label", l)

        if l not in labels_template:
            print("-> No template was found for label", l)
            continue

        tp = template_points[l]
        tp = voxel_down_sample(tp, voxel_size=VOXEL_SIZE[l]) # for manual
        save_pc(np.asarray(tp), COLOURS[3], f"{l}_template_sampled", SAVE_SCALE[l])
        tp_wn = generate_normals(tp)
        tp_resolution = pc_resolution(tp)

        sp = seg_points[l]
        sp = voxel_down_sample(sp, voxel_size=VOXEL_SIZE[l]) # for predicted
        print("Sampled", len(sp), "points for this object.")

        sp_resolution = pc_resolution(sp)
        sp_diameter = bounding_box_diameter(sp)
        print("Segmented object has a point cloud resolution of", sp_resolution, 'and
    bounding box diameter of', sp_diameter)

        ins_sp = None
        if INSTANCE_INPUT[l] is not None:
            ins_sp = load_instances(INSTANCE_DIR, INSTANCE_INPUT[l])
        else:
            ins_sp = segment_instances(sp, sp_diameter, INSTANCE_DISTANCE[l])

        print(f"Segmented {len(ins_sp)} instances for component {l}.")


        for i, ins in enumerate(ins_sp):

            print("------------------------------------------------")
            print(f"Matching for instance {i+1}/{len(ins_sp)} of object {l}.")

            if len(ins) < 5:
                print(f"- !Not enough points found for this instance ({len(ins)}). Continue
    with next.")
                continue
```

```
52.
53.
54.             if INSTANCE_INPUT[l] is not None:
55.                 ins = voxel_down_sample(ins, voxel_size=VOXEL_SIZE[l])
56.
57.             save_pc(ins, COLOURS[i], f"{l}_{i}")
58.
59.             ins_diameter = bounding_box_diameter(ins)
60.             sp_wn = generate_normals(ins)
61.
62.             # RANSAC Loop
63.             best_hypo = None
64.             pre_score = 0
65.             best_score = 0
66.             total_applied = []
67.             ins_done = False
68.             for ransac_i in range(MAX_RANSAC_ITERATIONS):
69.                 if ins_done:
70.                     print(f"Completed alignment of instance {i} of object {l}.")
71.                     break
72.
73.                 ransac_start = time.perf_counter()*1000
74.
75.                 # randomly sample point pairs and their features
76.                 pairs = retrieve_ppf(sp_wn, ins_diameter, MAX_PAIRS)
77.                 template_pairs = retrieve_ppf(tp_wn, ins_diameter, MAX_PAIRS)
78.
79.                 # generate hypothesis by comparing with template ppfs
80.                 i_matched = []
81.                 n = 0
82.                 for key in pairs:
83.                     if key in list(template_pairs.keys()):
84.                         n += 1
85.                         i_matched.append(key)
86.
87.                 local_best = 0
88.                 for j, index in enumerate(i_matched):
89.
90.                     # create hypothesis
91.                     hypothesis = generate_hypothesis(pairs[index], template_pairs[index])
92.                     if hypothesis is None:
93.                         continue
94.
95.                     # apply hypothesis
96.                     tp_applied = apply_transformation(tp, hypothesis) # not yet implemented
97.                     total_applied = [y for x in [total_applied, tp_applied] for y in x]
98.
99.                     # score hypothesis
100.                      score = point_region_overlap(tp_applied, ins, pre_score,
    th=3*sp_resolution/2) # too slow atm, th:experimental threshold value
101.
102.                      local_best = max(local_best, score)
103.                      best_score = max(best_score, score)
104.                      pre_score = max(pre_score, score)
105.
106.                      if score >= TARGET_OVERLAP[int(l)]:
107.                          tp_applied, score, fine_matrix = fine_align(tp_applied, ins,
    best_score, th=3*sp_resolution/2)
108.                          hypothesis = add_translation(hypothesis, fine_matrix)
109.
110.                          if best_hypo is None or score > best_score:
111.                              best_hypo = hypothesis
112.                              best_score = score
113.
114.  #                       if score >= FINE_TARGET_OVERLAP[int(l)]:
```

```
115.  #                             save_pc(tp_applied, [255, 255, 255], f"{l}_{i}_applied")
116.  #                             ins_done = True
117.  #                             break
118.
119.                  ransac_time = int(time.perf_counter()*1000 - ransac_start)
120.  #                 print(f"-- Best hypothesis' point region overlap: {local_best}, in
    {ransac_time}ms.")
121.
122.              print(f"- RANSAC complete with best score {best_score} and pre_score
    {pre_score}.")
123.
124.              if best_hypo is None:
125.                  continue
126.
127.              print_transform(best_hypo)
128.
129.              # Fine align and save transformed point cloud
130.              tp_applied = apply_transformation(tp, best_hypo)
131.              save_pc(tp_applied, [255, 255, 255], f"{l}_{i}_applied", SAVE_SCALE[l])
132.
133.  #             if len(total_applied) > 0:
134.  #                 save_pc(total_applied, [0, 0, 255], f"{l}_{i}_total_applied",
    SAVE_SCALE[l])
135.
136.  print("Finished")
```

Appendix F: Auxiliary Code Display (ppf.py)

```
1.   from enum import Enum
2.  import laspy
3.  import numpy as np
4.  import open3d as o3d
5.  import random
6.  import time
7.  import math
8.  from scipy.spatial import distance
9.  from scipy.spatial.transform import Rotation
10. from pathlib import Path
11. import mmh3
12.
13. class Labels(Enum):
14.     MANUAL = 0
15.     PREDICTED = 1
16.
17.
18. # LAZ Loading and Saving ----------------------------------------------------------------------
    ---------------------------------
19.
20. ''' Load Segmented Input Point Cloud
21. '''
22. def load_scene(directory, file, label_mode: Labels):
23.
24.     file = directory + file
25.     print("----------------------------------------")
26.     print("--Attempting to load scene from:", file)
27.
28.     with laspy.open(file) as fh:
29.         las = fh.read()
30.
31.         lbl = None
32.         try:
33.             if label_mode == Labels.MANUAL:
34.                 lbl = las.label
35.             else:
36.                 lbl = las['class']
37.         except ValueError as e:
38.             raise e
39.
40.         pc = np.stack([las.x, las.y, las.z], axis=1)
41.
42.         if label_mode == Labels.MANUAL:
43.             pc = normalize(pc)
44.
45.         save_pc(pc, [0, 255, 0], "03_02_post")
46.
47.         labels = np.unique(lbl)
48.
49.         print("--Total points:", len(pc))
50.         print("--Found labels:", labels)
51.         print("--Sorting objects:")
52.
53.         # For each label
54.         objects = {}
55.         for i in labels:
56.             objects[i] = np.delete(pc, np.where(lbl != i), axis=0)
57.             print("Sorted ", i, "with length", len(objects[i]))
58.
59.         print("----------------------------------------")
```

```
60.         return labels, objects
61.
62. def load_instances(directory, file):
63.
64.     file = directory + file
65.     with laspy.open(file) as fh:
66.         las = fh.read()
67.
68.         pc = np.stack([las.x, las.y, las.z], axis=1)
69.
70.         lbl = las.label
71.
72.     labels = np.unique(lbl)
73.
74.     instances = [0]*len(labels)
75.     for i in labels:
76.         instances[int(i)] = np.delete(pc, np.where(lbl != i), axis=0)
77.
78.     return instances
79.
80.
81. ''' Load Template Point Clouds
82. '''
83. def load_templates(directory):
84.
85.     print("----------------------------------------")
86.     print("--Preparing template features from folder:", directory)
87.
88.     files = list(Path(directory).glob('*.laz'))
89.     if len(files)==0:
90.         print("No laz files found")
91.
92.     objects = {}
93.     labels = []
94.     for f in files:
95.
96.         l = int(f.name.split('_')[0])
97.         if l in labels:
98.             continue
99.
100.          labels.append(l)
101.
102.          with laspy.open(str(f)) as fh:
103.              las = fh.read()
104.              objects[l] = np.stack([las.x, las.y, las.z], axis=1)
105.
106.     return labels, objects
107.
108.
109. ''' Save Point Cloud
110. '   Method that stores a point list with a given colour as a laz file in a designated
    directory.
111. '''
112. def save_pc(points, colour, file, scale=0.000000001):
113.
114.     las = laspy.create()
115.     las.header.scale = [scale, scale, scale]
116.     pc = np.asarray(points)
117.
118.     las.x = pc[:,0]
119.     las.y = pc[:,1]
120.     las.z = pc[:,2]
121.     colours = np.array([colour]*len(points))
122.     las.red = colours[:,0]
123.     las.green = colours[:,1]
```

```
124.      las.blue = colours[:,2]
125.
126.      las.write("/home/jupyter-zino/Data/data/save_las/{:s}.laz".format(file))
127.
128.
129.  # Point Cloud Modifications ----------------------------------------------------------------
    ---------------------------------------
130.
131.  ''' Generate Normals
132.  '   Generate Normals for a list of points. Returns pointlist with a point [x, y, z, nx,
    ny, nz]
133.  '   Uses open3D.
134.  '''
135.  def generate_normals(points):
136.
137.      pcd = o3d.geometry.PointCloud()
138.      pcd.points = o3d.utility.Vector3dVector(points)
139.
140.      pcd.estimate_normals(search_param=o3d.geometry.KDTreeSearchParamHybrid(radius=0.1,
    max_nn=30))
141.
142.      pts = np.asarray(pcd.points)
143.      nms = np.asarray(pcd.normals)
144.
145.      return np.stack([pts[:,0], pts[:,1], pts[:,2], nms[:,0], nms[:,1], nms[:,2]], axis=1)
146.
147.
148.  ''' Voxel-based Downsampling
149.  '   Samples points from input point list based on voxel grid.
150.  '   Similar method used in prediction, so recommended to use.
151.  '''
152.  def voxel_down_sample(points, voxel_size=0.025):
153.      pcd = o3d.geometry.PointCloud()
154.      pcd.points = o3d.utility.Vector3dVector(points)
155.
156.      return np.asarray(pcd.voxel_down_sample(voxel_size).points)
157.
158.
159.  # Point Cloud Properties ----------------------------------------------------------------
    ------------------------------------------
160.
161.
162.  ''' Point Cloud Resolution
163.  '   Calculates the resolution of an input point point cloud, approximated as the average
    nearest neighbour
164.  '   distance.
165.  '''
166.  def pc_resolution(points):
167.      start = time.perf_counter()*1000
168.
169.      distances = []
170.      for (point, i) in zip(points, knn(points, 1)):
171.          distances.append(dist(point, points[i][0]))
172.
173.      output = sum(distances) / len(distances)
174.
175.      runtime = time.perf_counter()*1000 - start
176.      print("Found resolution of", output, "in", int(runtime), "ms.")
177.
178.      return output
179.
180.
181.  ''' Bounding Box Diamater
182.  '   Returns the diameter of the oriented bounding box of a point cloud.
183.  '''
```

```
184.  def bounding_box_diameter(points):
185.      pcd = o3d.geometry.PointCloud()
186.      pcd.points = o3d.utility.Vector3dVector(points)
187.
188.      bb = o3d.geometry.OrientedBoundingBox.create_from_points(pcd.points)
189.
190.      a, b, c = bb.get_max_bound()
191.      diagonal = math.sqrt(a**2 + b**2 + c**2)
192.
193.      return diagonal
194.
195.
196.  ''' K-Nearest Neighbours
197.  '   Returns a list of the k-nearest neighbours of each point in an input point list.
198.  '''
199.  def knn(points, k):
200.      D = distance.squareform(distance.pdist(points))
201.      closest = np.argsort(D, axis=1)
202.      return closest[:, 1:k+1]
203.
204.
205.  # Main Functional Methods ----------------------------------------------------------------
    ---------------------------
206.
207.
208.  ''' Segment Instances
209.  '   Method that separates same object points from one point list into a list of point
    lists
210.  '   for each instance of that object. Uses spherical clusters to determine instances.
211.  '''
212.  def segment_instances(points, diameter, th, axis=0):
213.      start = time.perf_counter()*1000
214.
215.      centers = []
216.      instances = []
217.
218.      for p in points:
219.
220.          # check if the current point belongs to an existing instance
221.          added = False
222.          for i, center in enumerate(centers):
223.              if dist(p, center) < th:
224.                  instances[i].append(p)
225.                  centers[i] = np.average(instances[i], axis=0)
226.                  added = True
227.                  break
228.
229.          if added:
230.              continue
231.
232.          # otherwise add a new instance
233.          instances.append([p])
234.          centers.append(p)
235.
236.      runtime = time.perf_counter()*1000 - start
237.  #     print("Segmented", len(instances), "instances in", int(runtime),"ms.")
238.      return instances
239.
240.
241.  ''' Point Pair Feature Retrieval
242.  '   Method that samples point pairs from a point list and hashes their features.
243.  '   Returns a dict with hashed features as indices and point pairs as values.
244.  '''
245.  def retrieve_ppf(points, pc_diameter, sample_count):
246.      pairs = {}
```

```
247.
248.      for i in range(sample_count):
249.
250.          # sample a random p1
251.          p1 = points[random.randrange(len(points))]
252.
253.          # sample a random p2 that is within the neighbourhood of p1
254.          p2 = None
255.          while p2 is None:
256.              p2 = points[random.randrange(len(points))]
257.
258.              if (p1 == p2).all() or dist(p1, p2) > pc_diameter:
259.                  p2 = None
260.
261.          # Get an index from the pairs features
262.          hash_val = mmh3.hash_from_buffer(features(p1, p2))
263.
264.          # Add it to the hashmap
265.          pairs[hash_val] = (p1, p2)
266.
267.      return pairs
268.
269.
270.  ''' Calculate Features
271.  '   Method that calculates the actual point pair features for two input points and
272.  '   returns a tuple containing the features.
273.  '''
274.  def features(p1, p2):
275.
276.      m1, n1 = pan(p1)
277.      m2, n2 = pan(p2)
278.
279.      # Feature 1: distance between point 1 and point 2
280.      f1 = round(dist(p1, p2))
281.
282.      vec_dis = m2 - m1
283.
284.      # Feature 2: angle between distance vector and p1's directional vector (normal)
285.      f2 = round(angle_between(vec_dis, n1))
286.
287.      # Feature 3: angle between distance vector and p2's directional vector
288.      f3 = round(angle_between(vec_dis, n2))
289.
290.      # Feature 4: Kmin / Kmax, where Kmin and Kmax are the principal curvatures at p1
291.      f4 = 0
292.
293.  #     print([f1, f2, f3, f4])
294.      return np.array([f1, f2, f3, f4])
295.
296.
297.  ''' Calculate Hypothesis
298.  '   Generates a transformation hypothesis based on matched scene and template pairs.
299.  '   INCOMPLETE
300.  '''
301.  def generate_hypothesis(scene_pair, template_pair):
302.      pi, pj = scene_pair
303.      pk, pl = template_pair
304.
305.      mi, ni = pan(pi)
306.      mj, nj = pan(pj)
307.      mk, nk = pan(pk)
308.      ml, nl = pan(pl)
309.
310.      uij = (mj - mi) / np.linalg.norm(mj - mi, 2)
311.      vpij = np.matmul((np.identity(3) - np.matmul(uij, uij.T)), ni)
```

```
312.      vij = vpij / np.linalg.norm(vpij, 2)
313.
314.      ukl = (ml - mk) / np.linalg.norm(ml - mk, 2)
315.      vpkl = np.matmul((np.identity(3) - np.matmul(ukl, ukl.T)), ni)
316.      vkl = vpkl / np.linalg.norm(vpkl, 2)
317.
318.      rij = np.asarray([uij, vij, np.cross(uij, vij)])
319.      rkl = np.asarray([ukl, vkl, np.cross(ukl, vkl)])
320.
321.      rijkl = np.matmul(rkl, rij.T)
322.
323.      cvec = mk + np.matmul(rijkl, mi)
324.
325.      dir_angle = np_angle([1, 0, 0], (mi - mk))
326.      hypo_angle = np_angle([1, 0, 0], cvec)
327.      if hypo_angle < dir_angle-np.pi/8 or hypo_angle > dir_angle+np.pi/8:
328.          return None
329.
330.      tijkl = np.asarray([
331.       np.asarray([rijkl[0, 0], rijkl[0, 1], rijkl[0, 2], cvec[0]]),
332.       np.asarray([rijkl[1, 0], rijkl[1, 1], rijkl[1, 2], cvec[1]]),
333.       np.asarray([rijkl[2, 0], rijkl[2, 1], rijkl[2, 2], cvec[2]]),
334.       np.asarray([0, 0, 0, 1])
335.      ])
336.
337.  #     tijkl = np.asarray([
338.  #      np.asarray([1.0, 0.0, 0.0, cvec[0]]),
339.  #      np.asarray([0.0, 1.0, 0.0, cvec[1]]),
340.  #      np.asarray([0.0, 0.0, 1.0, cvec[2]]),
341.  #      np.asarray([0.0, 0.0, 0.0, 1.0])
342.  #     ])
343.
344.  #     print(tijkl)
345.      return tijkl
346.
347.
348.  def fine_align(input, control, init_score, th=1):
349.      start = time.perf_counter()*1000
350.
351.      output = input
352.      score = init_score
353.
354.      step_th = 0.001
355.
356.      fine_align_matrix = x_matrix_trans(0)
357.
358.      # x translation
359.      for i in [-1, 1]:
360.          step = i
361.          while abs(step) > step_th:
362.
363.              change = apply_transformation(output, x_matrix_trans(step))
364.              new_score = point_region_overlap(change, control, 0.0, th)
365.
366.              if new_score > score:
367.                  output = change
368.                  score = new_score
369.                  fine_align_matrix = add_translation(fine_align_matrix,
    x_matrix_trans(step))
370.                  continue
371.
372.              step /= 1.1
373.
374.      # y translation
375.      for i in [-1, 1]:
```

```
376.          step = i
377.          while abs(step) > step_th:
378.
379.              change = apply_transformation(output, y_matrix_trans(step))
380.              new_score = point_region_overlap(change, control, 0.0, th)
381.
382.              if new_score > score:
383.                  output = change
384.                  score = new_score
385.                  fine_align_matrix = add_translation(fine_align_matrix,
    y_matrix_trans(step))
386.                  continue
387.
388.              step /= 1.1
389.
390.      # z translation
391.      for i in [-1, 1]:
392.          step = i
393.          while abs(step) > step_th:
394.
395.              change = apply_transformation(output, z_matrix_trans(step))
396.              new_score = point_region_overlap(change, control, 0.0, th)
397.
398.              if new_score > score:
399.                  output = change
400.                  score = new_score
401.                  fine_align_matrix = add_translation(fine_align_matrix,
    z_matrix_trans(step))
402.                  continue
403.
404.              step /= 1.1
405.
406.
407.      runtime = int(time.perf_counter()*1000 - start)
408.  #     print(f"Fine Alignment produced a score of {score} in {runtime}ms.")
409.
410.      return output, score, fine_align_matrix
411.
412.  def add_translation(m1, m2):
413.      return [
414.          [m1[0][0], m1[0][1], m1[0][2], m1[0][3]+m2[0][3]],
415.          [m1[1][0], m1[1][1], m1[1][2], m1[1][3]+m2[1][3]],
416.          [m1[2][0], m1[2][1], m1[2][2], m1[2][3]+m2[2][3]],
417.          [m1[3][0], m1[3][1], m1[3][2], m1[3][3]]
418.      ]
419.
420.  def x_matrix_trans(step):
421.      return [
422.          [1, 0, 0, step],
423.          [0, 1, 0, 0],
424.          [0, 0, 1, 0],
425.          [0, 0, 0, 1]
426.      ]
427.  def y_matrix_trans(step):
428.      return [
429.          [1, 0, 0, 0],
430.          [0, 1, 0, step],
431.          [0, 0, 1, 0],
432.          [0, 0, 0, 1]
433.      ]
434.  def z_matrix_trans(step):
435.      return [
436.          [1, 0, 0, 0],
437.          [0, 1, 0, 0],
438.          [0, 0, 1, step],
```

```
439.          [0, 0, 0, 1]
440.      ]
441.
442.
443.  ''' Apply Transformation
444.  '   Method that applies a transformation to a point list. Returns a point list.
445.  '''
446.  def apply_transformation(points, trans_matrix):
447.      data = np.hstack((points, np.asarray([[1]]*len(points))))
448.      return np.matmul(trans_matrix, data.T).T[:,:3]
449.
450.
451.  ''' Point Region Overlap
452.  '   Tests point of a test point cloud against proximity to points in a control point cloud
    and returns the
453.  '   percentage of test points that are in proximity with control points.
454.  '''
455.  def point_region_overlap(test, control, best_score, th=1):
456.      start = time.perf_counter()*1000
457.
458.      miss_limit = int((1-best_score)*100)
459.      n = 0
460.
461.      indices = np.random.randint(len(test), size=100)
462.      for i, tp in enumerate(np.array(test)[indices]):
463.          for cp in control:
464.              if dist(tp, cp) < th:
465.                  n += 1
466.                  break
467.
468.          if i - n > miss_limit:
469.              break
470.
471.      runtime = time.perf_counter()*1000 - start
472.  #     print(f"Performed region overlap in {int(runtime)} ms with result {n/100}.")
473.
474.      return n/100
475.
476.
477.  # Utility Functions ----------------------------------------------------------------------
    ----------------------------------------------
478.
479.
480.  ''' Point and Normal
481.  '   Breaks 6-field array into two 3-field arrays to separate point coordinates and their
    normals.
482.  '''
483.  def pan(point):
484.      return point[:3], point[3:]
485.
486.
487.  ''' Point Distance
488.  '   Returns the distance between two points.
489.  '''
490.  def dist(a, b):
491.      ax, ay, az = a[:3]
492.      bx, by, bz = b[:3]
493.      return math.sqrt((ax-bx)**2 + (ay-by)**2 + (az-bz)**2)
494.
495.
496.  def round(input):
497.      return format(input, '.2f')
498.
499.
500.  def angle_between(vec1, vec2):
```

```
501.      high = np.linalg.norm(np.cross(vec1, vec2))
502.      low = np.linalg.norm(np.dot(vec1.T, vec2))
503.      return np.arctan(high/low)
504.
505.
506.  def np_angle(vec1, vec2):
507.      unit_vector_1 = vec1 / np.linalg.norm(vec1)
508.      unit_vector_2 = vec2 / np.linalg.norm(vec2)
509.      dot_product = np.dot(unit_vector_1, unit_vector_2)
510.      return np.arccos(dot_product)
511.      return np.arctan(dot_product)
512.
513.
514.  def vec_mag(vec):
515.      x, y, z = vec
516.      return math.sqrt(x**2+y**2+z**2)
517.
518.  def normalize(pc):
519.  #     centroid = np.mean(pc, axis=0)
520.      vmin = np.min(pc, axis=0)
521.      vmax = np.max(pc, axis=0)
522.      span = vmax-vmin
523.      centroid = vmin + span/2
524.      #centroid[2] = 0 # Leave z-axis alone
525.
526.      pc = pc - centroid
527.  #     m = np.max(np.linalg.norm(pc, axis=1))
528.      m = 13.5 # 27m diagonal range will be from -1 to 1
529.      pc = pc / m
530.      return pc
```