

# Automated Parametric Modeling of Bridge Components Using Point Clouds

Scientific work to obtain the degree

**Bachelor of Science (B.Sc.)**

at the TUM School of Engineering and Design  
of the Technical University of Munich.

**Supervised by** Prof. Dr.-Ing. André Borrmann  
M. Saeed Mafipour (M.Sc.)  
Chair of Computational Modeling and Simulation  
Markus Hochmuth (Dipl.-Ing.)  
Obermeyer Digital Solution GmbH

**Submitted by** Irena Samu [REDACTED]  
[REDACTED]  
[REDACTED]  
[REDACTED]

**Submitted on** 31. Mai 2023

## Abstract

In this thesis, a novel automatic parametric model-to-cloud fitting algorithm, proposed by MAFIPOUR et al. (2021), is discussed and tested. This method automates the process of generating a 3D geometric model of a bridge component based on its point cloud data (PCD).

To facilitate the maintenance and operation process of existing bridges, there is an urgent need to digitalize the inspection, condition assessment, and evaluation of bridges. To this end, the idea of implementing a bridge of digital twins has recently gained ground in the Architecture, Engineering, and Construction (AEC) industry. A digital twin is a virtual replica of a physical asset that enables visualization, analysis, and simulation of the physical counterpart in a virtual environment. By continuously receiving real-time data from sensors and scanners embedded in the physical counterpart, digital twins enable remote monitoring and condition-based maintenance.

However, automating the process of extracting useful information from scans and monitoring data is a complicated task that needs to be solved. Part of this task is to automate the generation of a 3D model from PCD, which is currently mainly done manually. However, this is very time-consuming and costly, and the existing automated methods can only fit primitive models to point cloud subparts. Therefore, MAFIPOUR et al. (2021) proposed a new parametric model fitting method that can automatically generate arbitrary 3D models based on PCD to advance the development of digital twinning for bridge maintenance.

In this thesis, a segmented bridge point cloud is a prerequisite, and the focus is on the reconstruction of the bridge deck. A parametric model is fitted to the point cloud using a metaheuristic algorithm to automatically reconstruct the bridge deck based on the PCD. The novelty of this method is to employ reverse engineering with parametric modeling to extract the value of parameters from point clouds. For this reason, the point cloud of the deck must first go through some preparation steps to extract the cross-section of the deck. Once the cross-section of the deck is extracted, a 2D parametric prototype model, or parametric prototype model (PPM), is created that approximates the shape of the point cloud cross-section. This PPM is then fitted to the corresponding point cloud section using a metaheuristic optimization algorithm that minimizes the distance between the parametric model and the point cloud section. Once the PPM is correctly fitted to the point cloud, its parameters provide information about the dimensions of the deck, a surface can be created based on the parametric model, and this surface is then extruded along an axis to generate a 3D volumetric body of the deck.

The described method was tested on six bridges with arbitrary deck shapes. The results show that an average mean absolute error (MAE) of 4.21 cm and an average processing time of 90.79 sec per bridge is achieved. Thus, the parametric model fitting to the point cloud algorithm provides a close estimation in a short time, plus important information about the dimension of a component is automatically extracted and used to generate a complex shaped bridge deck, which in turn can be further used for simulation, analysis, and visualization.

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Background</b>	<b>4</b>
2.1	3D Geometric Representation . . . . .	4
2.1.1	Boundary Representation . . . . .	4
2.1.2	Implicit Representation . . . . .	6
2.1.3	Procedural Modeling . . . . .	7
2.1.4	Parametric Modeling . . . . .	9
2.2	Machine learning and Clustering algorithms . . . . .	10
2.2.1	DBSCAN . . . . .	11
2.2.2	PCA . . . . .	12
2.2.3	RANSAC . . . . .	15
2.3	Reverse Engineering . . . . .	16
2.4	Optimization . . . . .	17
2.5	Related research . . . . .	20
2.6	Research gap . . . . .	22
<b>3</b>	<b>Methodology</b>	<b>23</b>
3.1	Capturing process and data set . . . . .	24
3.2	Preparation of Point Clouds . . . . .	26
3.2.1	Sub sampling . . . . .	27
3.2.2	Geometric transformation . . . . .	28
3.2.3	Preparation abutment . . . . .	29
3.2.4	Preparation bridge deck . . . . .	29
3.3	Parametric modeling . . . . .	30
3.3.1	Generating a parametric prototype model . . . . .	31
3.3.2	Metaheuristic optimization algorithms . . . . .	32
3.3.3	Fitness function . . . . .	36
<b>4</b>	<b>Results/ Discussion</b>	<b>38</b>
4.1	Evaluation metrics . . . . .	38
4.2	Real world applications . . . . .	39
4.3	Discussion . . . . .	44
<b>5</b>	<b>Conclusion</b>	<b>46</b>
	<b>Bibliography</b>	<b>48</b>

# List of Figures

2.1	An example of a B-Rep data structure and the corresponding vertex-edge-face graph, from BORRMANN and BERKHAHN (2018)	4
2.2	Half-Edge in ACIS	5
2.3	Co-Edge in ACIS	5
2.4	An example of a triangulated face, from BORRMANN and BERKHAHN (2018)	6
2.5	An example of several implicit functions representing a surface, or region, from LU (2019)	6
2.6	An example of a CSG-Tree, from BORRMANN and BERKHAHN (2018)	7
2.7	An example of extrusion and rotation methods, from BORRMANN and BERKHAHN (2018)	8
2.8	An example of a parametric sketch in a CAD environment, from BORRMANN and BERKHAHN (2018)	9
2.9	An illustration of the key concept of DBSCAN, from WALKER (2022)	11
2.10	Projection of a data point onto a line	12
2.11	The distance of one point to an imaginary line, that is rotated until the square sum of all points distances to the line are at the minimum	12
2.12	Principle components	12
2.13	Illustration of the covariance	13
2.14	An example of conventional reverse engineering, from SAIGA et al. (2021)	16
2.15	A structure of optimization algorithms	18
2.16	First iteration	19
2.17	Second iteration	19
2.18	Third iteration	19
3.1	Preparation steps for bridge deck and abutments	26
3.2	The cross-section of a bridge deck point cloud before and after sub-sampling	27
3.3	Simple sketch illustrating the concept of sub-sampling	27
3.4	Translating center into origin	28
3.5	Detect deck alignment and calculate angle	28
3.6	Rotated point cloud	28
3.7	Abutment point cloud	29
3.8	Deck point cloud cross section and PPM in random position with random dimensions	30
3.9	A parametric model corresponding to a deck cross-section, or PPM	32
3.10	An illustration of the bounding box of a point cloud cross section	33
3.11	Fitting process with small and big dimensional constraints	34
3.12	An example of the flowchart for Teaching-learning-based optimization (TLBO), from RAO et al. (2011)	34
3.13	Euclidean distances from one $p_i$ to all edges of the PPM	36

4.1	Test bridges . . . . .	39
4.2	Fitted parametric models: (a) bridge deck 1; (b) bridge deck 2; (c) bridge deck 3; (d) bridge deck 4; (e) bridge deck 5; (f) bridge deck 6 . . . . .	42
4.3	Illustration of a deck point cloud compared to the corresponding automatically generated 3D geometric model . . . . .	43
4.4	Comparison between the parametric model fitting (a) and the convex hull (b)	44
4.5	Comparison between the parametric model fitting (a) and the convex hull (b)	45

# List of Tables

4.1	MAE [cm] of fitted bridge decks . . . . .	41
4.2	Time [sec] elapsed during fitting process . . . . .	41
4.3	Number of points in the subsampled deck points cloud cross-section . . . . .	41

# Acronyms

<b>AI</b>	artificial intelligence
<b>ALS</b>	airborne laser scanner
<b>B-Rep</b>	Boundary Representation
<b>BIM</b>	Building Information Modeling (STEIN, 2014)
<b>BMS</b>	bridge management system
<b>CAD</b>	computer-aided design
<b>CSG</b>	Constructive Solid Geometry
<b>DBSCAN</b>	Density-Based Spatial Clustering of Applications with Noise
<b>DT</b>	digital twin
<b>FCM</b>	Fuzzy-C-Means
<b>LIDAR</b>	Light Detection And Ranging
<b>MAE</b>	mean absolute error
<b>MLS</b>	mobile laser scanner
<b>PCA</b>	principal component analysis
<b>PCD</b>	point cloud data
<b>PPM</b>	parametric prototype model
<b>RANSAC</b>	Random Sample Consensus
<b>RMSE</b>	root-mean-square error
<b>TLBO</b>	Teaching-learning-based optimization
<b>TLS</b>	terrestrial laser scanner

# Chapter 1

## Introduction

On August 1, 2007, the Interstate 35W bridge over the Mississippi River collapsed in Minneapolis, Minnesota, United States. Thirteen people were killed and 145 injured. The National Transportation Safety Board investigation identified several factors that contributed to the collapse. These included inadequate inspections, a lack of redundancy in the bridge's design, and a failure to recognize the importance of warning signs (BOARD, 2008).

Bridges are an essential part of modern infrastructure, opening up new opportunities for economic growth, social inclusion, and cultural exchange. However, despite their importance, bridges are often subjected to harsh environmental conditions and wear and tear, as illustrated in the collapse of the Minneapolis Bridge. With that said, bridges require constant maintenance and renovation.

Nevertheless, the challenges to the preservation of their structural integrity are significant. The maintenance of bridges is a complex and ongoing process that requires careful planning, regular inspections, and timely repairs. However, current bridge maintenance practices are primarily inefficient and outdated. Visual inspection is still the most common method of assessing bridge conditions, and the data obtained is then entered into a bridge management system (BMS) where the data is analyzed by analytical tools and algorithms (LU and BRILAKIS, 2019).

A BMS is a software or database designed to support the management and maintenance of bridges. It provides a comprehensive framework for collecting, storing, analyzing, and reporting data related to bridge inventory, condition assessment, inspection, maintenance, and rehabilitation. The analytical tools in a BMS include evaluation of collected data followed by assignment of a condition rating, analysis of historical data to identify deterioration patterns and predict future condition changes, and combining condition information with other factors such as traffic volumes or environmental conditions. Even though BMSs are a step towards digitalization, they lack some essential capabilities. For example, the ability to decide whether or not to repair a damage. To do this, the damage must be located in a 3D model and on the exact component before deciding whether a supporting component is damaged and whether the bridge structure is at risk (ISAILOVIĆ et al., 2019).

As HOSAMO and HOSAMO (2022) explains, new technologies such as the Internet of Things, artificial intelligence (AI), and cloud computing enable the digitalization of various assets. In addition, sensors and intelligent data collection, such as drones and vehicles equipped with cameras and sensors, help to improve the monitoring of any lifecycle that requires maintenance. The digital twin (DT) idea uses the technologies mentioned above to integrate a virtual replica of a physical object, system, or process. In other words, the DT



represents a digital counterpart that mirrors the real-time and real-world status. The DT is usually created and updated using data collected from sensors, drone scans, and other sources that capture real-time condition information about the physical entity. These new technologies provide the necessary foundation for new developments in the digitalization of bridge maintenance, for example, defect prognostics and production efficiency. Recently, the idea of implementing a bridge DT into a BMS has appeared in the literature, which could lead to the development of a more intelligent BMS.

As stated by HOSAMO and HOSAMO (2022), this new BMS would include an automated process for obtaining real-time condition information from instrumentation and sensors installed on bridges. In addition, the BMS would consist of a semantically rich model that is automatically built and contains the data of the original and current geometry. Finally, the BMS should include the DT derived from the BIM model. The DT should be able to be automatically updated with site monitoring data, and it could be linked to additional layers, such as finite element models, to simulate predicted future behavior.

The main challenge in implementing the DT is processing the data received from the sensors and scanners into useful information. A significant challenge is automatically generating a 3D geometric model based on scan data and photogrammetry. Generating solid models from point clouds is often done manually, which is extremely time- and cost-consuming. The ability to automatically generate 3D models is necessary to advance the digitalization of bridge maintenance.

Therefore, this thesis aims to explore a method for extracting dimensional parameters of a segmented bridge deck point cloud by using a novel parametric model fitting method proposed by MAFIPOUR et al. (2021) and to test the method on six bridges. The fitting method consists of three main steps. First, a cross-section of the segmented bridge deck point cloud needs to be extracted, so preparation steps for the bridge deck PCD are introduced, using several machine learning techniques. Then, based on the point cloud section, a parametric model is generated following the logic of reverse engineering, and finally, the parametric model is fitted to the point cloud section using an optimization algorithm.

The thesis is divided into five chapters. Chapter 2 provides background information for the techniques used in chapter 3, starting with a description of existing computational approaches for 3D model representation, then defining machine learning and explaining some algorithms used in this thesis, continuing with a brief explanation of reverse engineering, followed by an introduction of optimization algorithms, and finalized with a related research and research gap section. In chapter 3, the proposed method is explained, while chapter 4 is an overview of the results.

# Chapter 2

## Background

### 2.1 3D Geometric Representation

Chapter 1 shows that digital twinning is of great importance for the future development of bridge maintenance. The 'as-built' 3D geometry model is fundamental for digitalizing building information, because the model can be manipulated and visualized in various ways, and semantic information can be added. The digital representation of 3D geometric models can be solved using different methods describing volumetric bodies. There are four primary methods for implementing solid modeling representation in computer-aided design (CAD) software: Boundary Representation (B-Rep), implicit representation, procedural modeling, and parametric modeling. In the following paragraphs, these methods will be explained.

#### 2.1.1 Boundary Representation

This section explains B-Rep following BORRMANN and BERKHAHN (2018). B-Rep describes bodies using their boundary surfaces. The surfaces consist of faces, which are comprised of edges. These edges have a start vertex and an end vertex. The arrangement of these boundary elements builds the topology of the model, and the coordinates of the vertices determine the geometry. In other words, the topological representation defines the connectivity and relationships among the different elements of the solid model, and the geometric representation of B-Rep describes the shape and size of the solid model. The topology of the 3D model can also be characterized by a vertex-edge-face graph, as shown in figure 2.1.

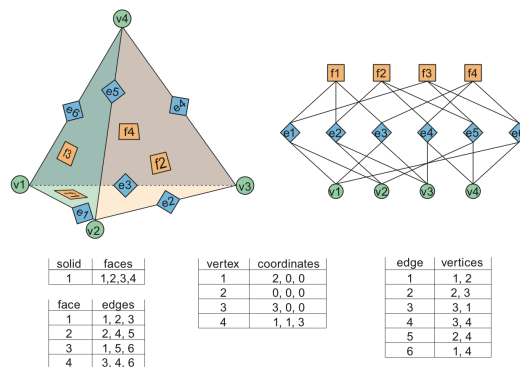


Figure 2.1: An example of a B-Rep data structure and the corresponding vertex-edge-face graph, from BORRMANN and BERKHAHN (2018)

As the vertex-edge-face-graph only describes simple solids without openings, an extension of this topological description is needed to properly represent more complex bodies. An object-oriented data model based on the geometry kernel ACIS was proposed to create bodies with several openings. These openings are defined using so-called CoEdges. As shown in figure 2.2, a CoEdge is a pair of half-edges that always are arranged in opposite directions. Each half-edge has a pointer to the next half-edge around the same face and a pointer to the opposite half-edge bounding the neighboring face. This connectivity information is essential for performing geometric operations, such as Boolean operations, as it allows the kernel to determine which edges and faces are connected. The half-edge data structure is typically used to generate the representation of openings in the kernel. Each edge is represented as a pair of half-edges, and each face is described as a collection of half-edges that define its boundary, called a loop, as shown in figure 2.3. Eventually, The openings are bounded by a collection of faces and volumes connected and oriented by these edges.

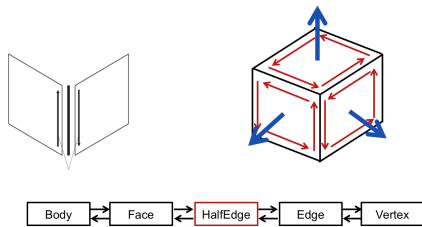


Figure 2.2: Half-Edge in ACIS

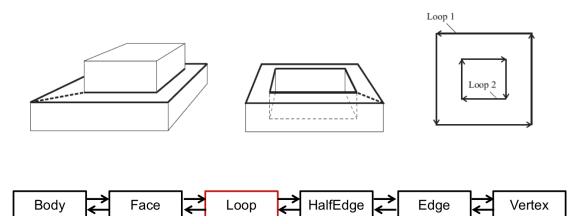


Figure 2.3: Co-Edge in ACIS

### Triangulated Face Set

The triangulated face set, also known as a triangle mesh or simply a mesh, is a simplified [B-Rep](#), which consists of a collection of triangles that connect vertices in 3D space, forming a network of interconnected triangles, as shown in figure 2.4. Each triangle in the mesh is defined by three vertices and their corresponding positions in 3D space, where the vertices are typically represented by their 3D coordinates  $x$ ,  $y$ , and  $z$ . Triangulated meshes are widely used because triangles are simple and planar geometric primitives that can tessellate any surface. For example, a curved surface is approximated by choosing a finer mesh, which requires more storage capacity. BORRMANN and BERKHAHN (2018) explains the Indexed Face Set, as a data structure of the triangulated surface model. First, all the coordinates of the vertices are stored as a list, where each vertex is assigned an index number. These indexes then define the triangulated surfaces to avoid saving points repeatedly. This data structure is simple and has the capability to represent complex shapes. However, it still requires a high storage capacity for precise curved surface representation. When it comes to automatically generating 3D solid models from point clouds, this representation is mostly useless because the point clouds are often missing some data points due to the limited line of sight during the scanning process. Instead, a planar surface would represent this region, which is likely to be a poor approximation of the real model.

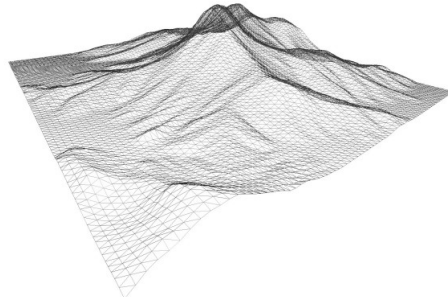


Figure 2.4: An example of a triangulated face, from BORRMANN and BERKHAHN (2018)

## 2.1.2 Implicit Representation

Unlike the Boundary representation, where a collection of vertices and polygons defines surfaces, implicit modeling represents surfaces as the zero level-set of a continuous function called an implicit function. In implicit modeling, the shape of an object is defined by an equation or function rather than explicitly specifying its surface geometry. The implicit function results in a positive or negative value at any point in space, indicating whether the point is inside or outside the surface. The surface itself is defined as the set of points where the implicit function evaluates to zero. However, the implicit functions are only capable of describing simple shapes, such as the plane, sphere, and torus shown in the figure. To create complex and idealized shapes, the primitives must be logically combined, for example, using Boolean operations. Therefore, implicit functions themselves are not useful for 3D modeling based on point cloud data because they cannot form complex shapes (SHAPIRO, 2002).

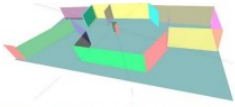
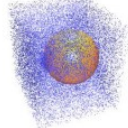
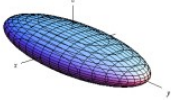

Shape	Equation	Example
Plane	$ax + by + cz = d$ (Eq. 2.5)	 (Limberger & Oliveira, 2015)
Sphere	$x^2 + y^2 + z^2 = r^2$ (Eq. 2.6)	 (Schnabel et al., 2007)
Ellipsoid	$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$ (Eq. 2.7)	 (Weisstein, 2018a)
Torus	$(\sqrt{x^2 + y^2} - R)^2 + z^2 = r^2$ (Eq. 2.8)	 (Weisstein, 2018d)

Figure 2.5: An example of several implicit functions representing a surface, or region, from LU (2019)

### 2.1.3 Procedural Modeling

In contrast to the above-discussed modeling, procedural modeling saves not only the final result but also the steps that are required to create a 3D model. Therefore, this type of representation is known as a procedural method. The discussion on two specific procedural techniques is following.

#### Constructive Solid Geometry

Constructive Solid Geometry (CSG) is a fundamental procedural description of 3D geometries. Primitive objects, like cubes, cylinders, and spheres, are combined using Boolean operators such as AND, OR, and NOT to generate more complex models. This results in a construction tree, such as in figure 2.6. The primitive bodies are simple shapes that can be described by mathematical equations, for example, a cylinder equation in standard position is  $x^2 + y^2 - r^2 = 0$ . To make it easy to adapt to the application, the dimensions of the basic bodies are usually parameterized in CAD implementations (BORRMANN and BERKHAHN, 2018).

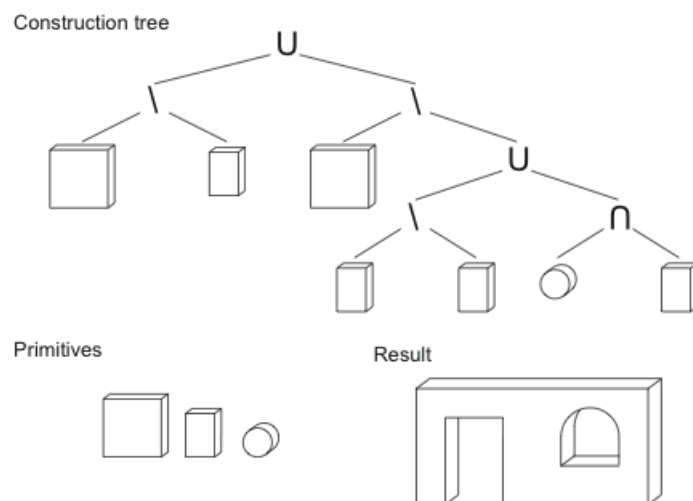


Figure 2.6: An example of a CSG-Tree, from BORRMANN and BERKHAHN (2018)

## CAD Functionalities

Extrusion and rotation methods are exactly what their names state: a 2D sketch that is extruded along a path in the 3D space. There are several ways to draw a 2D sketch, implicit functions can be used to describe simple shapes like rectangles, triangles, and circles, but very common in CAD modeling software is to draw a 2D parametric sketch that gets extruded or rotated. The four ways of extrusion, rotation, sweep, and lofting are visually described in figure 2.7, BORRMANN and BERKHAHN, 2018. Furthermore, these extruded volumetric bodies can be combined using **CSG**. Most **CAD** software have implemented these extrusion methods and **CSG**, and are therefore capable of generating idealized shapes.

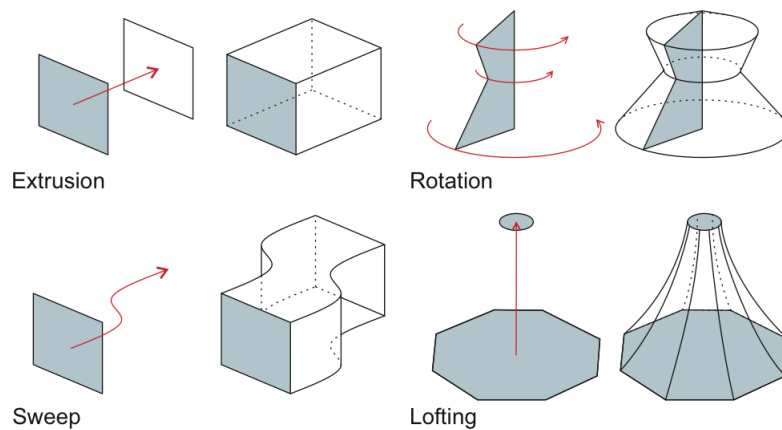


Figure 2.7: An example of extrusion and rotation methods, from BORRMANN and BERKHAHN (2018)

## 2.1.4 Parametric Modeling

Parametric modeling allows users to define flexible geometries by creating a system of elements, that are linked together by a series of mathematical equations, and the element's dimensions are defined by parameters. For example, to create a parametric 2D sketch of a rectangle, four parameters need to be created first: start vertex x coordinate, start vertex y coordinate, length, and height. The first vertex of the rectangle is defined by an equation equal to a vector comprising the x- and y-coordinates of the start vertex, then the next point is defined by an equation taking the resulting vector of the first vertex and adding the length parameter to its x-value. The third point is similarly generated by taking the resultant vector of the second point and adding the height parameter to the y-value. The final point takes the third vertex and subtracts the length parameter from the x-value. In this way, the edges of the rectangle, or the elements of the system, are created one after the other, and each element refers to the coordinates of the previous element. Similarly, more complex shapes can be defined in this way, but require the definition of more geometric and dimensional constraints. In order to create a 3D representation, the parametric modeling is implemented in combination with **CSG** and **CAD** functionalities. One such software is Siemens NX, where solid models can be represented by first generating a 2D parametric sketch and then extruding this sketch along a path. Further, these extruded models can be combined by **CSG** and represent extremely complex shapes, as shown in figure 2.8 (BORRMANN and BERKHAHN, 2018). An example of a more complex parametric model is shown in section 3.3.1, as this is the method used in this thesis for the 3D modeling from **PCD**.

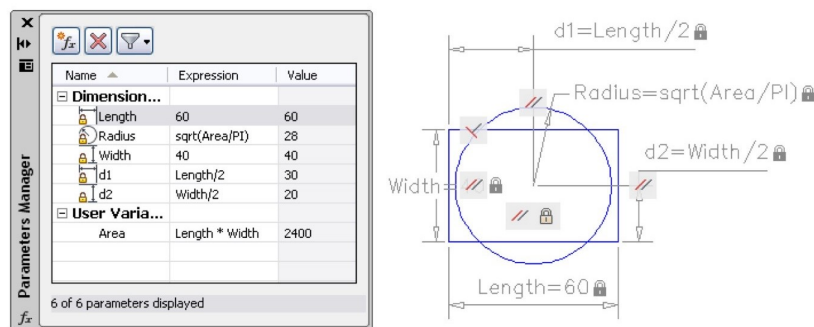


Figure 2.8: An example of a parametric sketch in a CAD environment, from BORRMANN and BERKHAHN (2018)

## 2.2 Machine learning and Clustering algorithms

In most studies on 3D modeling from [PCD](#), [AI](#) is used. [AI](#) is a broad field that generally develops multiple approaches to making machines intelligent. Machine learning, a subfield of this, is about making machines that can learn to perform specific tasks. Further below, a quick definition of machine learning and clustering algorithms, following [REBALA et al. \(2019\)](#) is given. Finally, multiple algorithms used for the preparation of the point cloud are introduced, including the Density-Based Spatial Clustering of Applications with Noise ([DBSCAN](#)) clustering algorithm, the principal component analysis ([PCA](#)), and the Random Sample Consensus ([RANSAC](#)) algorithm.

Machine learning algorithms aim to solve complex problems that cannot be solved by conventional algorithms. In conventional algorithms, as [REBALA et al. \(2019\)](#) states, a "detailed design" of the problem is implemented. This means that the algorithm includes a specific problem-oriented process, rule, or logic to solve the problem. Hence, the conventional algorithm receives some input and this input is processed through the "detailed design", in order to generate a correct solution. In contrast to this approach, machine learning algorithms receive input with the correct output, or labeled data set. Machine learning algorithms then learn the logic behind solving the problem by itself. Further, based on the learned "detail design", the algorithm is able to predict correct solutions to other input data concerning this problem; this is called supervised learning. This is why machine learning can solve problems at a higher level, meaning that an algorithm is able to solve several non-specific problems by solving the problem in an indirect way. In addition, machine learning algorithms can solve problems that are hard or even impossible to formulate in a "detailed design". However, this results in the problem of not clearly knowing how exactly a problem is being solved by the algorithm. Obviously, the main characteristic of machine learning is the learning technique. There are a number of different learning techniques, but the two main techniques will be presented below - the supervised learning model and the unsupervised learning model ([REBALA et al., 2019](#)).

In supervised learning, the machine is given a data set along with the right answers to a question, therefore the algorithm is able to predict a correct outcome for other inputs without previous knowledge about the labeled output. It has elements of trying different 'detailed designs', and if the currently tested 'detailed design' does not lead to the labeled output data, the optimization algorithm gets taught by the failure and internally adjusts its understanding. This type of learning technique is particularly useful for solving classification and regression problems. In unsupervised learning, the algorithm is given only the input data without the labeled output data, and so the algorithm tries to identify clusters or groups of similar items. In order to solve clustering problems, all clustering algorithms are machine learning algorithms that use the unsupervised learning technique. Further below, two clustering algorithms used in the preprocessing of the bridge deck point cloud are discussed ([REBALA et al., 2019](#)).



## 2.2.1 DBSCAN

**DBSCAN** is a clustering algorithm used in various applications, including image segmentation. WALKER (2022) explains the key concept of **DBSCAN** as follows. First, two parameters must be defined, the distance  $\epsilon$  and the minimum samples. The distance  $\epsilon$  is used to stretch an imaginary circle around an instance to specify the currently observed region in which all other instances are summed up and considered as the instance's neighborhood. If the number of neighborhood instances is equal to or greater than the minimum sample value, the instance is considered to be the core instance and its neighborhood is considered to be the cluster. This cluster is then extended by iteratively visiting the neighbors of each instance, and if a point has more neighborhood instances than the minimum sample value, they are all added to the cluster. An instance that is more than  $\epsilon$  away from all other instances is considered noise, as shown in figure 2.9. This process is repeated until all instances have been visited.

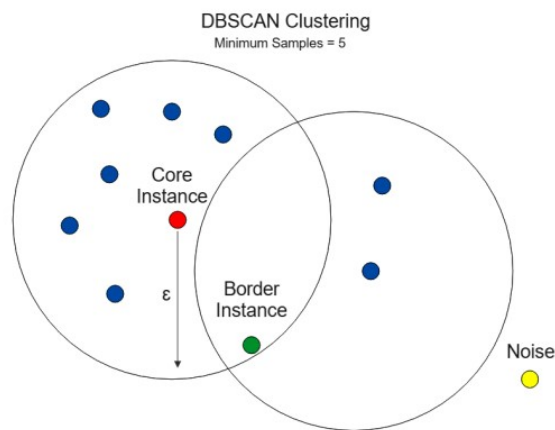


Figure 2.9: An illustration of the key concept of DBSCAN, from WALKER (2022)

The **DBSCAN** clustering algorithm has many advantages that make it ideal for clustering abutments, as they typically consist of at least two retaining walls, and for further steps in point cloud preparation, these retaining walls need to be saved individually in a point cloud segment.

- 1) **DBSCAN** can detect arbitrary shapes, so all types of retaining walls can be detected as clusters, regardless of shape complexity.
- 2) Resistance to noise is a plus, as reducing noise around the segmented point cloud clusters improves the efficiency of the fitting algorithm.
- 3) The number of clusters does not need to be predefined, which helps to automate the preparation algorithms.

## 2.2.2 PCA

PCA is a statistical technique used for dimensionality reduction and data exploration. It aims to transform a high-dimensional data set into a lower-dimensional representation while preserving the most important information contained in the original data (RABBANI, 2006).

At its core, PCA identifies the linear combinations of random variables, known as principal components, that capture the maximum amount of variance in the data set. These linear combinations are made in such a way that the new variables, or principal components, are uncorrelated. The first principal component accounts for the largest possible variance, and each subsequent component explains the maximum remaining variance orthogonal to the previous components. Thus, there are as many principal components as there are random variables in the data.

In Figure 2.12, the principal components are shown for a data set of 2D points previously centered on the coordinate origin. Geometrically, the first principal component represents the direction of the data that contains a maximum amount of variance. To find this principal component, an imaginary line, the dashed line in figure 2.11, is rotated around the origin until the sum of the squared distances  $d_i$  is minimal, resulting in the principal component shown in figure 2.12. According to Pythagoras' theorem, as the distance  $d_i$  becomes smaller, the distance  $a_i$  between the projected red point and the coordinate origin becomes larger. In fact, the distance  $a_i$  is maximized in the computational implementation of the PCA.

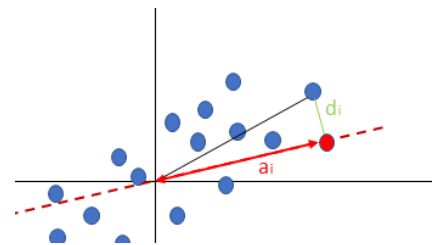


Figure 2.10: Projection of a data point onto a line

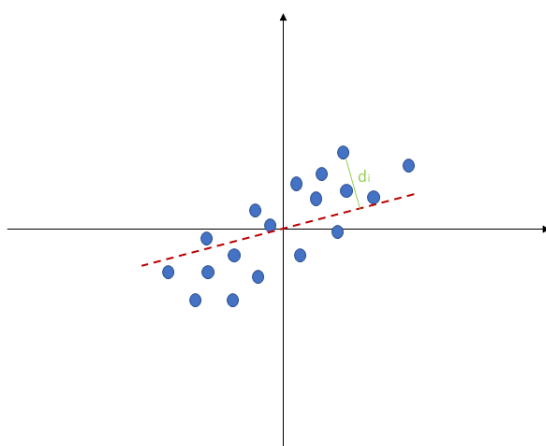


Figure 2.11: The distance of one point to an imaginary line, that is rotated until the square sum of all points distances to the line are at the minimum

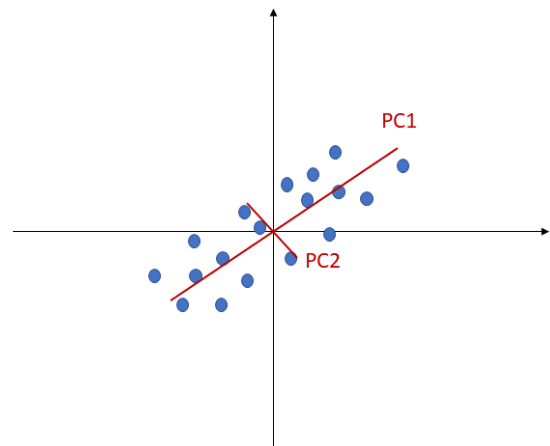


Figure 2.12: Principle components

In order to explain the computation of the principle components of a data set, the mathematical background is further explained, following REBALA et al. (2019). To simplify the explanation, the observed data will be further considered as a 2D point cloud, represented in  $X$  with two random variables, the x and y coordinates.

$$X = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_i & y_i \end{bmatrix}$$

## Covariance

In the first calculation step, the covariance matrix of the data set  $X$  is calculated, so a brief definition of covariance is given below, following MURPHY (2022). To understand the covariance matrix, the difference between covariance and variance will be discussed.

- 1) The variance measures the variation of a single random variable. This means that for observed data  $X$  with two random variables, the variance for the x- or y-coordinates describes the distribution of the points in the x- or y-direction, respectively. The following equation 2.1 defines the variance in the x-direction

$$var(x_i) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \quad (2.1)$$

where  $n$  is the number of point in the observed data and  $\bar{x}$  the mean value of the x-values(BRÉMAUD, 2017).

- 2) Covariance measures the relative relationship between variables. If there is no relationship between the variables, the covariance value will be close to zero, but if the correlation between the points is high, the value will be close to 1 or -1, as shown in the figure ???. The following equation 2.2 defines covariance as

$$cov = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) \quad (2.2)$$

where  $n$  is the number of points in the observed data and  $\bar{x}, \bar{y}$  the mean value of the x- and y-values, respectively.

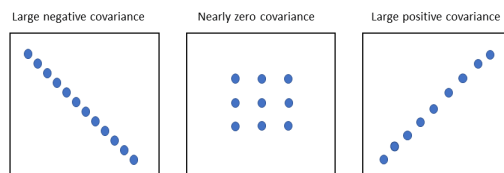


Figure 2.13: Illustration of the covariance

The covariance calculates the entries of the covariance matrix. The diagonal entries of the covariance matrix are the variances and the other entries are the covariance, so the covariance matrix is symmetric, as  $\sigma(x_i, x_j) = \sigma(x_j, x_i)$ . The calculation of the covariance matrix is expressed as follows.

$$COV = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})(X_i - \bar{X})^T = \begin{pmatrix} cov(x, x) & cov(x, y) \\ cov(y, x) & cov(y, y) \end{pmatrix} = \begin{pmatrix} var(x) & cov(x, y) \\ cov(y, x) & var(y) \end{pmatrix} \quad (2.3)$$

If the data is centered on the origin  $\bar{X} = 0$  follows and the equation 2.3 is reduced to 2.4.

$$COV = \frac{1}{n-1} X X^T \quad (2.4)$$

### Eigenvector and Eigenvalue

After calculating the covariance matrix of  $X$ , its eigenvector is calculated. In the rows of this eigenvector, the principal components of the observed data set  $X$  are determined.

Eigenvalues and eigenvectors are concepts from linear algebra that are used in various mathematical and computational applications, including [PCA](#).

An eigenvector of a square matrix is a non-zero vector whose multiplication by the matrix results in a scaled version of the original vector. In other words, the direction of the eigenvector remains unchanged when the matrix is applied because it is only rotated about its own axes by multiplication with the matrix. The following equation defines the square matrix  $A$  and an eigenvector  $v$

$$A * v = \lambda * v$$

where  $\lambda$  represents the eigenvalue associated with the eigenvector  $v$ . The eigenvalue  $\lambda$  determines the scaling factor by which the eigenvector  $v$  is stretched or shrunk when multiplied by the matrix  $A$  (BRONSTEIN et al., 1989).

The [PCA](#) is used in the preparation of the bridge deck [PCD](#) in order to detect the alignment of the bridge. First, the points of the bridge deck point cloud are projected on the x-z-plane. For calculating the alignment of the bridge, the first principle component of the projected 2D cloud points is calculated. As the bridge already is centered, the covariance of the projected points is calculated, and then its Eigenvector. The Eigenvector is defined as a two by two matrix, as the observed data has two dimensions, and the first row presents a vector that defines the direction of the first principle component. Finally, the information about the direction of the first principle component is used to define the alignment of the bridge.

### 2.2.3 RANSAC

In the thesis, the Random Sample Consensus ([RANSAC](#)) algorithm is used to separate the wing walls from the retaining wall. Each wall is a planar surface and lies in an individual plane. To identify a wall, a planar object in the point cloud can be detected using the [RANSAC](#) algorithm. [RANSAC](#) is an iterative algorithm used to fit models when outliers are present. By iteratively selecting subsets of data points and estimating model parameters based on these subsets, the algorithm aims to find the best model that fits the majority of the outlier data points. The key concept behind RANSAC is the assumption that the data can be divided into inliers and outliers. Inliers are the data points that can be accurately described by the underlying model, while outliers are the data points that deviate significantly from the model. The implementation of the [RANSAC](#) algorithm for the detection of planar surfaces in point clouds is explained below (DERPANIS, 2010).

The first step is to select a random subset of the data points, in this case, three random points, and to create a plane from these three points. The goal is to find a plane that fits into a flat wall in the point cloud, so the accuracy of how well the plane fits into the point cloud needs to be calculated. This is done by calculating the distance of all points to the plane and counting the number of points that are within a boundary, for example -10cm to 10cm from the plane. Data points within this boundary are considered inliers. This process is repeated and many planes of three points are randomly generated. The plane with the highest number of inliers is the best-fitted plane. Finally, the points belonging to this plane are extracted and the process is repeated for a fixed number of iterations or until all planes are detected.

The advantage of the [RANSAC](#) algorithm is that it can deal with a large percentage of outliers in the data and still provide a reliable estimate of the model. By iterative sampling and fitting models, [RANSAC](#) reduces the influence of outliers and focuses on finding the model that best represents most of the data. However, to balance the trade-off between accuracy and computational efficiency, it is important to choose appropriate thresholds and stopping conditions.

## 2.3 Reverse Engineering

Reverse engineering is the process of analyzing and understanding the design, structure, or functionality of a product, system, or component through deconstruction or examination of how the observed subject behaves. This method involves working backward from the final product or system to uncover its underlying principles, technologies, or algorithms, and then copying these critical features of an existing object in order to create an exact or enhanced virtual model of it. There is a difference between conventional engineering and reserved engineering. Conventional methods first create a concept, logic, or abstract idea of a model based on customer needs and other relevant information. Then, virtual models are generated based on the concept, and the best of these virtual models is transformed into a physical model or prototype. In summary, conventional engineering follows the concept-model-object workflow. In some cases, a reverse process is required, such as the modeling of existing bridges. For this purpose, the reverse engineering process is proposed, where a digital CAD model is derived from the final physical object, thus representing an object-model-concept workflow (SAIGA et al., 2021).

Reverse engineering approaches are used in several CAD and graphics applications, and there are many studies on reverse engineering algorithms for solid model reconstruction from PCD because it gives a workflow in situations where a copy of a physical object is needed. Especially in the infrastructure and building sector, the digitalization of objects manufactured in the pre-digital era is of great interest, since CAD models are needed for future modifications of the construction and for the digitalization of maintenance systems (HELLE and LEMU, 2021). There are many algorithms for conventional reverse engineering of primitives from PCD with the

following procedure, also shown in figure 2.14. First, a scan of the physical object is made, then the scan data is processed by e.g. noise and point reduction and other filter algorithms. Finally, the most complicated step is to generate a solid model from the observed data, which allows manipulation of the 3D data (UY et al., 2022). The automated method for parametric modeling of bridge components using PCD presented in this thesis also follows the reverse engineering approach and follows the above-mentioned procedure steps, scanning to processed point cloud to CAD solid model.

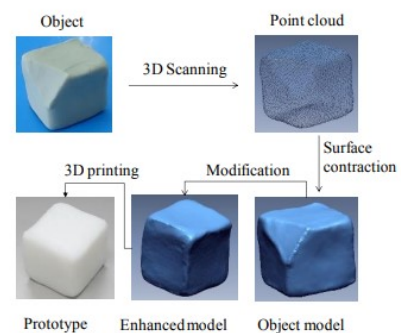


Figure 2.14: An example of conventional reverse engineering, from SAIGA et al. (2021)

## 2.4 Optimization

This section deals with mathematical optimization problems and computational methods for solving them. First, a brief definition of optimization problems in mathematics is given, and then methods for solving these optimization problems using optimization algorithms are discussed. Finally, metaheuristic optimization algorithms are explained in more detail because of their importance in the method described in this thesis.

The purpose of an optimization algorithm is to solve an optimization problem, which is a mathematical problem that is solved by finding the solution that best fits given criteria. As GANDOMI et al. (2013) determines, an optimization problem can be expressed as follows, where function  $f_i(x)$  is the objective function or cost function.

$$\begin{aligned} \min_x \quad & f_i(x), & (i = 1, 2, \dots, M) \\ \text{subject to} \quad & h_j(x) = 0, & (j = 1, 2, \dots, j), \\ & g_k(x) \leq 0, & (k = 1, 2, \dots, k) \end{aligned} \tag{2.5}$$

However, the objective function also could be maximized, if applicable. The components of  $x$  are called design or decision variables, and they can be real, continuous, discrete, or a mix of both. The space spanned by the decision variables is called the design space or search space, and the equality  $h_j(x)$  and inequality  $g_k(x)$  are called constraints. The objective function represents the optimization problem, such as minimizing or maximizing cost, distance or performance. This problem is solved by finding the right set of design variables in the search space that results in a maximum or minimum of the objective function value, while the required conditions must be met by the solution, such as resource limitation, position, dimensional parameters, or quality standards.

Solving these optimization problems requires the use of mathematical methods and algorithms, ranging from classical analytical techniques to modern computational approaches. Optimization algorithms, which is an overarching term for computational methods solving these problems, include a wide range of different mathematical methods, from classical analysis to numerical approaches. Structuring optimization algorithms can be challenging since there are many different ways depending on their characteristics and objectives. However, in this thesis, optimization algorithms are mainly grouped into classic algorithms and stochastic algorithms, as shown in figure 2.15. Classic algorithms have objective functions that can be differentiated at a point, thus the algorithm can make use of the calculated gradient information. Stochastic optimization refers to a field of optimization algorithms that explicitly uses randomness to find the optima of an objective function. The main difference is that the fitness function of a stochastic algorithm cannot be derived. Further, stochastic optimization algorithms can be structured based on whether they operate on a population of candidate solutions or a single solution. Population-based algorithms maintain a set of candidate solutions and update them based on a fitness

function. Single-solution algorithms include simulated annealing and tabu search, which update a single solution based on a random or probabilistic rule.

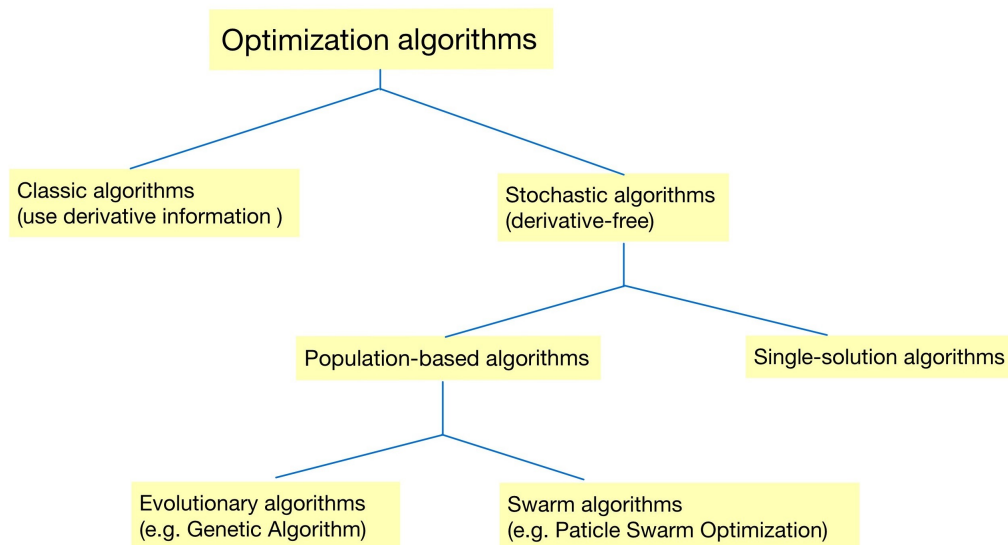


Figure 2.15: A structure of optimization algorithms

### Metaheuristic algorithms

Heuristic and metaheuristic are the main types of stochastic algorithms. As DOKEROGLU et al. (2019) states, a heuristic is a problem-solving strategy or guideline that aims to find good or satisfactory solutions efficiently, even if they are not guaranteed to be optimal. Heuristics are based on intuition, past experience, or domain knowledge and are used to simplify and guide the decision-making process in problem-solving. In computation, a heuristic algorithm determines an optimal solution by iteratively attempting to improve a candidate solution with respect to a given quality measure. Meta means 'higher level'. Thus, metaheuristics perform better than just simple heuristics, but they are pretty similar. Metaheuristic algorithms are often inspired by nature, specifically by swarm behavior and evolution. Therefore, they can be classified into swarm optimization and evolutionary algorithms, as shown in figure 2.15. The following paragraph describes the basic idea behind metaheuristic algorithms based on the explanation of GANDOMI et al. (2013).

Metaheuristic algorithms are population-based, stochastic optimization algorithms that use randomization and global exploration and usually address complex or non-linear optimization problems that cannot be expressed by a mathematical function. Solving these complex problems by exploring every possible solution in the search space is impossible. Metaheuristic algorithms are used to find the best solution in the search space in an acceptable time. GANDOMI et al. (2013) describes the two main strategies to achieve this goal, using the terms: "intensification" and "diversification". Diversification randomly generates a large number of solutions to explore the search space on a global scale. Intensification focuses on the current best solution and explores the surrounding region.



To illustrate these strategies, their implementation in particle swarm optimization is described. Considering the figures 2.16, 2.17 and 2.18, the goal of the particle swarm optimization is to minimize the distance between the parametric red rectangle and the stationary blue rectangle. In the beginning, a population of parametric rectangles is randomly initiated, represented by the five red rectangles. The best of these rectangles is calculated based on its closeness to the blue rectangle, and all the other population candidates are moved toward the best rectangle by adding a step. This step is calculated based on the distance between the candidate's position and the current best candidate. The movement toward the best solution is an example of intensification. Moreover, slight randomization is added to this step value. This randomization is an example of diversification because it avoids all candidates overlapping over the current best solution. So the key idea of the metaheuristic algorithm is the good combination of intensification, which ensures the convergence to the optimum, and diversification, which keeps the solution from being trapped. These steps are iteratively repeated until one of the population candidates fits in the blue rectangle.

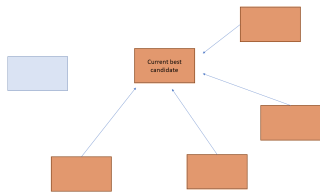


Figure 2.16: First iteration

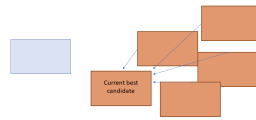


Figure 2.17: Second iteration

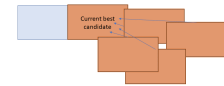


Figure 2.18: Third iteration

## 2.5 Related research

In general, researchers have put a lot of effort into creating useful models from **PCD** for decades. In particular, a lot of work has been done on building construction because the main components of buildings, such as walls and beams, all have very simple and standardized shapes. Researchers have mainly focused on object detection in raw **PCD**. Existing methods usually first cluster the **PCD** and automatically fit surface primitives to the clustered point cloud subsets. Using this surface reconstruction, the point cloud cluster is classified according to its shape, e.g. in indoor scenes, mostly rectangular shapes are detected, which can be classified as walls or ceilings.

What almost all surface fitting algorithms have in common is that they calculate some distance between the point cloud segment and the fitting model, which is then minimized. This distance calculation is often the most complicated part, and since primitive objects are being fitted, the distance can be described by an analytical equation. The optimization problem is then solved using the derivative information of the distance equation.

For surface reconstruction, there are many different research papers trying to implement different 3D representations of the point cloud cluster. As RABBANI (2006) points out: "the choice of representation is important as it influences the complexity of the resulting method and can limit the types of models that can be faithfully fitted". Each of these approaches has its own advantages and limitations.

VALERO et al. (2012) proposed an automatic method for generating indoor boundary models from **PCD**. After segmenting the set of points belonging to each wall, a plane is fitted to the segment. The sum of the distances between each point of the point cloud segment and the plane is minimized. The plane is defined by the following equation.

$$\Pi = Ax + By + Cz + D = 0$$

The following equation defines the optimization problem, where  $n$  is the number of points in the point cloud segment and  $x_i, y_i, z_i$  are the coordinates of each point.

$$\sum_{i=1}^n \frac{|Ax_i + By_i + Cz_i + D|^2}{A^2 + B^2 + C^2}$$

Therefore, this fitting problem is solved with a classical algorithm, since the optimization problem could be formulated in a derivative form. Once the planes have been fitted to the surfaces, the intersections between the connected planes are calculated. Then, after finding the edges and corners of the space, the vertices can be extracted. Based on these vertices and edges, a vertex-edge-surface graph is implemented and the **B-Rep** of the walls and ceiling is created. WALSH et al. (2013) describes another method for extracting important information from **PCD**. For the clustering part, existing sharp feature detection and segmentation methods are extended. Once the PC is segmented, surface fitting is

used for classification. Primitive objects are fitted to the segmented PC using a least squares approach. These primitive objects, such as cylinders, spheres, and cones, can be represented by a quadratic equation. The coefficients of the equation determine the surface type, and the correct coefficients are found by solving the least square solution. This is followed by a review of research that moves away from primitive object fitting and attempts to fit more complex 3D models to the point cloud.

RABBANI (2006) proposed an automatic method to fit a complex model, represented as a **CSG** tree, to the selected subset of points. The problem of calculating the orthogonal distance from a point to the complex model surface is solved by three different approaches. The first approach is to convert the **CSG** model into a **B-Rep** and calculate the distance from a given point to the nearest surface in the **B-Rep**. The second is to convert the **CSG** model to a triangle mesh and calculate the distance to a point from the nearest triangle. The last approach is to convert the **CSG** model to a point cloud and calculate the distance to the nearest point. LU and BRILAKIS (2019) proposed a novel automatic fitting method to generate a 3D shape using a slicing strategy. QIN et al. (2021) proposed a new boundary fitting algorithm to obtain the feature data of a segmented bridge beam with a cuboid structure. First, the cuboid structure is sliced to reduce the problem of extracting features from a 3D cuboid to extracting features from a 2D rectangular shape. The corner coordinates are then extracted using this new boundary-fitting algorithm. CROCE et al. (2023) CROCE et al. (2023) proposed a semi-automated 3D model reconstruction approach for architectural heritage. After semantic segmentation of an architectural heritage using machine learning algorithms, the segmented and classified point cloud is imported into a 3D modeling environment where each class of architectural components can be isolated from the initial point cloud. A library of template geometries is then generated for each component, following the logic of parametric **BIM** families. Each class can be customized over the point cloud by manipulating sliders and parameters. These template geometries are then propagated to generate repetitive elements.

## 2.6 Research gap

In the last three decades, there has been a growing interest in the automatic reconstruction of 3D models of existing buildings from [PCD](#), also known as scan-to-BIM or digital twinning, due to the advantages of [BIM](#) applications for the maintenance of buildings. The main efforts are made in the field of building and industrial construction, since they usually consist of standardized elements, such as walls, ceilings, and pipes. Usually, these elements can be represented by simple shapes, so-called primitives, such as cuboids or cylinders. Therefore, most existing methods focus on extracting these elements and then fitting surfaces or geometric primitives to the segmented point cloud subparts. However, this approach has its limitations, which is why some research gaps are listed below:

- 1) There is no standard metric for the validation of the accuracy and reliability of automatically generated 3D bridge models. Techniques must be developed to evaluate the quality, accuracy, and completeness of the models and to compare them to ground truth data or reference models.
- 2) An unsolved problem is the extraction of semantic features from point clouds and the classification of bridge components based on these features. To generate a semantically rich 3D model, techniques for classifying the type of bridge deck or identifying specific bridge elements such as expansion joints, railings, and cables need to be further explored.
- 3) Most methods focus on building and industrial components, which typically have simple shapes. Therefore, the extraction of primitive objects and subsequent fitting of surfaces or geometric primitives to the segmented point cloud subparts is required. However, bridges often have complex geometries and structural elements, requiring the fitting of complex shapes to point clouds. Research is needed to address the challenges associated with modeling and accurately representing these complex structures from [PCD](#).

## Chapter 3

# Methodology

The process of generating a bridge 3D model from bridge **PCD** involves two main steps. The first step is the segmentation of the **PCD** into individual bridge components, such as the bridge deck, railing, and abutment, which then are saved as individual, segmented point clouds, each representing one component. In the second step, the parametric modeling, a **PPM** is created corresponding to a segmented point cloud, and the **PPM** is then fitted into this cloud by applying a metaheuristic optimization algorithm, which minimizes the distance between the **PPM** and the segmented point cloud. Using the parametric model, the measurements of this bridge component are extracted to create its 3D volumetric model. Finally, all components are assembled to complete the entire bridge model.

The essential part of the thesis deals with the parametric modeling of one bridge component. Hence, a prerequisite is a segmented point cloud with an individual deck point cloud, abutment point cloud, and railing point cloud. The described process and all further explanations in chapter 2 are based on the research proposed by MAFIPOUR et al. (2021).

Generating a 3D model of one segmented point cloud requires knowledge of the shape and dimension of the bridge component. Commonly, geometric primitives are used for fitting point cloud subparts with standardized shapes, such as rectangular walls, pipes, flanges, and steel beams. This involves finding the best-fitting mathematical representation of a geometric shape to a set of points and using algorithms that minimize the distance between the points and the surface of the fitted primitive. Bridge point clouds have a very complex geometry, which is why it is difficult to fit primitive 3D bodies to complex, non-standardized shapes like bridge decks.

The novelty of the approach used in this thesis is the utilization of a 2D parametric model, or **PPM**, to fit a point cloud cross-section. Therefore, the introduction of one more step before starting the parametric modeling is necessary; the preparation of the point cloud. Which extracts a cross-section of one bridge point cloud component. After the preparation, a **PPM** is created, which is a close replica of the observed component cross-section, but has the ability to take different dimensions and positions by steady form. Finally, the distance between these randomly generated **PPM** edges and the cross-section of the point cloud is iteratively calculated and minimized using a metaheuristic optimization algorithm.

The scope of this thesis covers the extraction of measurements from concrete bridge decks since the majority of bridges are made of reinforced or prestressed concrete. In Germany, prestressed and reinforces bridges account for 70 % and 17 % of the existing stock of federal motorways, respectively (für DIGITALES UND VERKEHR, 2019). Additionally, the proposed method was only tested on the bridge deck, since the deck usually has the

most complicated profile, compared to the other component's cross sections, and therefore provides the most efficient way to test the parametric model fitting algorithm. Specifically, bridges with straight alignments are exclusively used for the case study. Therefore, the following preparation steps are only valid for the straight alignment. These steps would need to be adapted for a curved alignment.

In this chapter, an overview of the data acquisition process and the resulting data quality and data set preprocessing is given first. Then the preparation steps for the extraction of the cross-section are given, followed by an explanation of the parametric model fitting process.

## 3.1 Capturing process and data set

### Capturing process

Common methods for 3D **PCD** acquisition are 3D laser scanning, photogrammetry, videogrammetry, RGB-D camera, and stereo camera. 3D laser scanning or Light Detection And Ranging (**LiDAR**) measures the distance to a target object by emitting a laser beam and detecting its reflection. There are two different techniques, the time-of-flight technique and the phase-shift technique. The time-of-flight technique measures the time between emitting the beam and detecting the reflection. From this time, the distance can be calculated since the speed of the laser is known. In the phase shift technique, a wave is emitted and when its reflection is detected, the phase shift of the wave is registered, so the distance can be calculated based on the wavelength. Additionally, there are three types of scanners: the terrestrial laser scanner (**TLS**), airborne laser scanner (**ALS**) and mobile laser scanner (**MLS**). The **TLS** is placed on a tripod on the ground and is stationary during the scanning process. For this reason, the **TLS** has the highest accuracy, at least at the millimeter level, and is therefore used to monitor buildings and infrastructure. For example, to scan a bridge, several **TLS**s are positioned around the bridge to capture the entire object. The **ALS** is typically conducted from an aircraft or helicopter equipped with a specialized **LiDAR** sensor and therefore is often used to measure distances and create highly accurate and detailed 3D representations of the Earth's surface. In **MLS**, multiple laser scanners are typically installed on the vehicle, facing different directions (WANG et al., 2020).

When acquiring the point cloud of a bridge, after taking several scans from positions around the bridge, the generated scans need to be processed. First, the noise of a point cloud must be removed, which is called data cleansing. Second, the collected point clouds from each scanner must be aligned in a common coordinate system; this step is known as data registration. Third, the point cloud is segmented into meaningful subsets, such as individual bridge components like deck, abutment, and railing. Fourth, the final step is data recognition, which aims to classify certain objects in the point cloud (WANG et al., 2020).

## Resulting quality of points

However, the methods described above produce point clouds with several imperfections. These imperfection properties of point clouds are listed below. (HELLE and LEMU, 2021)

- 1) Due to the geometric features of the shape or the position of the scanner, point clouds have variations in sampling density, meaning that some areas have a higher density of points while others have a lower density. This can cause different regions of the point cloud to have different levels of detail and accuracy.
- 2) Point clouds can contain noise, which refers to inaccurate measurements, thus sensor noise. Noise can arise from sensor limitations, environmental conditions, or reflective surfaces that cause reflections and multiple returns. Noise can distort the shape and details of the captured objects.
- 3) Outliers are individual points that deviate significantly from the expected distribution in the point cloud, which should be the true surface of the scanned object. Outliers can be caused by sensor errors, occlusions, or objects that are not part of the scene of interest.
- 4) When multiple scans are used to create a point cloud of the entire object, errors can occur in the alignment and registration of the individual scans. Misalignment or registration errors can cause distortions and misrepresentations in the final merged point cloud. The larger the scans, the more difficult their correct alignment becomes.
- 5) A common problem is missing data and gaps in the point cloud due to occlusion during the scanning process, limited viewpoints, or high light absorption.

## 3.2 Preparation of Point Clouds

In this section, the steps for extracting the cross-section of the deck point cloud are explained, as shown in figure 3.1. First, the entire point cloud is geometrically transformed, which means the point cloud is translated to the coordinate origin, and then the point cloud is rotated about the z-axis to align it with the x-axis. Once the point cloud is transformed, the preparation steps are different for the abutment and the deck. First, however, is an explanation of subsampling, which is used in the preparation process. Sub-sampling is enormously important for further processing using clustering algorithms and [PCA](#), as they require an evenly distributed point set for accurate results.

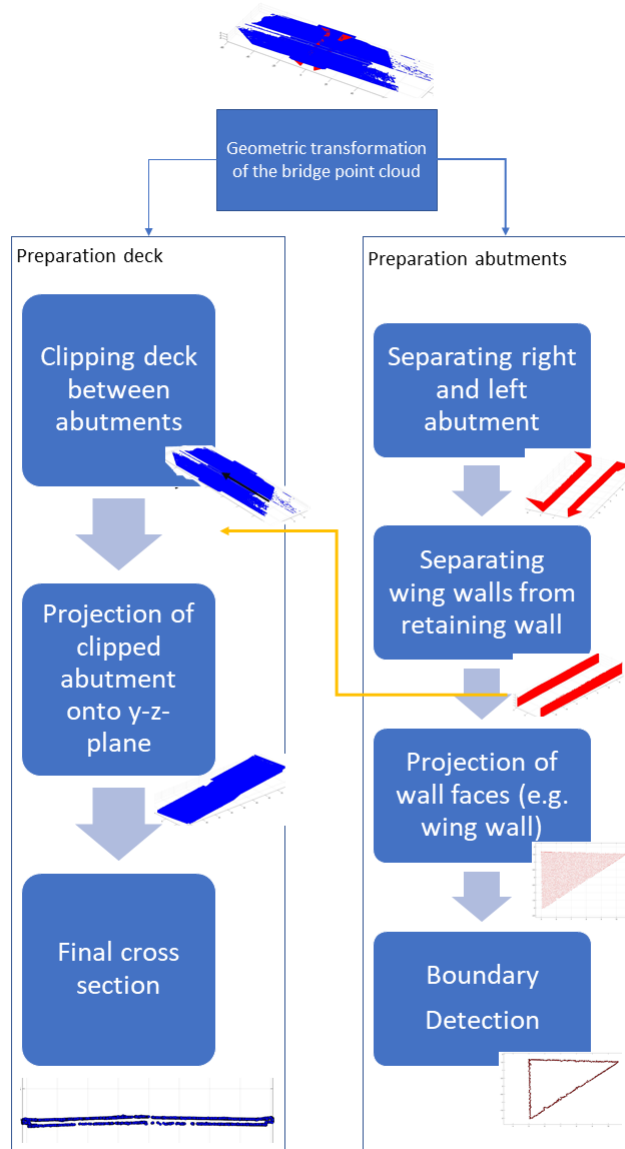


Figure 3.1: Preparation steps for bridge deck and abutments



### 3.2.1 Sub sampling

In order to avoid overlapping points and uneven point distribution, there are several subsampling methods, and they all reduce the size of a data set by selecting a subset of the original data, as shown in figure 3.2. The following is the implementation of this process.

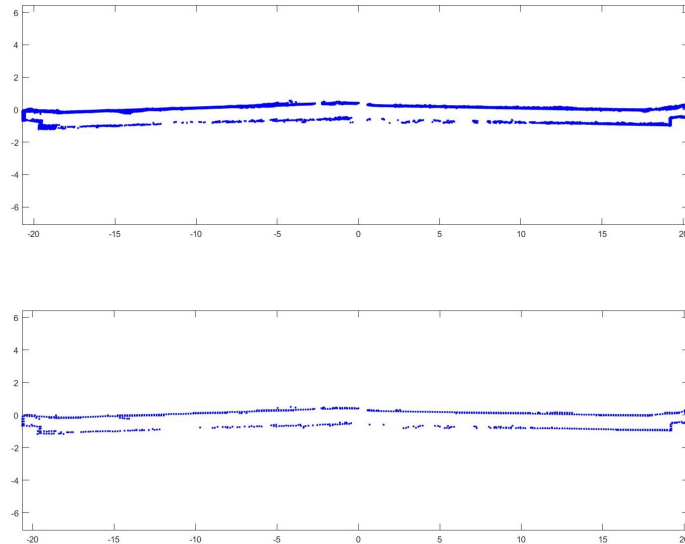


Figure 3.2: The cross-section of a bridge deck point cloud before and after sub-sampling

A grid of predefined size is laid over the points. In the scope of this thesis, the bridge decks have a width between 10 m and 40 m, hence 15 cm wide by 5 cm high grid cell is an appropriate approximation of the point cloud cross-section. In other words, the cross-section of the bridge deck point cloud consists of a point every 15 cm in the x direction and every 5 cm in the y direction after sub-sampling.

The next step is to run a loop over each grid cell, checking to see if there are any points in a grid cell. If there are one or more points, they are replaced by a point in the center of the grid cell. A simple sketch in figure 3.3 illustrates the concept of the sub-sampling method. The original data set is represented by the blue points. After sub-sampling, the blue points are reduced to the red points. The size of the sub-sampled red points does not actually change. The sketch shows them larger to make them more visible.

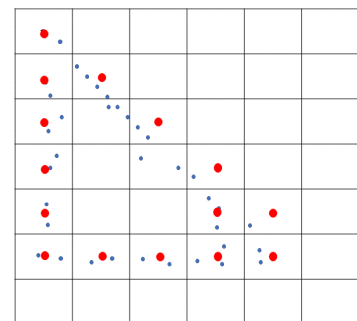


Figure 3.3: Simple sketch illustrating the concept of sub-sampling

### 3.2.2 Geometric transformation

To simplify the preparation of the point cloud, the bridge point cloud must be translated to the coordinate origin and rotated so that it is aligned with the x-axis. In order to translate the point cloud, the coordinates of the deck's center are calculated, using the following equation

$$cent = \frac{\sum_{i=1}^i \vec{v}_i}{i} \quad \text{for } \vec{v}_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$$

where  $\vec{v}_i$  is the vector of one point in the point cloud. To translate the center of the whole point cloud,  $cent$  is subtracted from every point  $\vec{v}_i$  of the point cloud, as shown in figure 3.4.

The second step is to rotate the abutment point cloud around the z-axis and align it with the x-axis. This is more complicated because the angle between the x-axis and the alignment of the bridge must first be calculated. The calculation of the bridge orientation requires the use of PCA, which is explained in section 2.2.2. Before applying PCA, the deck point cloud is projected onto the x-z plane and this 2D point set is subsampled to obtain a uniform point distribution. The first principle component of the sub-sampled point cloud is then computed. Once the alignment of the bridge is determined, the angle between the alignment and the x-axis is calculated using trigonometry, as shown in figure 3.5.

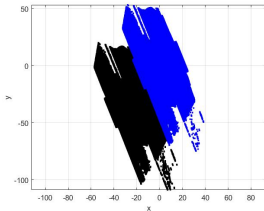


Figure 3.4: Translating center into origin

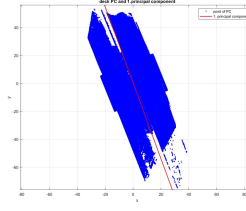


Figure 3.5: Detect deck alignment and calculate angle

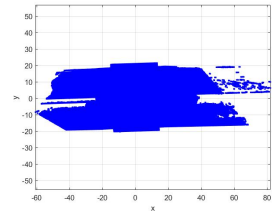


Figure 3.6: Rotated point cloud

Further, the point cloud needs to be rotated by multiplying every point  $\vec{v}_i$  from the point cloud with the rotation matrix, which is expressed by

$$T = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

where  $\alpha$  is the previously calculated angle between the bridge alignment and the x-axis, as shown in figure 3.6.

### 3.2.3 Preparation abutment

The abutment consists of right and left abutment walls. A wall is composed of a retaining wall (highlighted in green figure 3.7) and two wing walls (highlighted in red-figure 3.7). The following steps describe the derivation of the 2D wall profile from the 3D point cloud cluster.

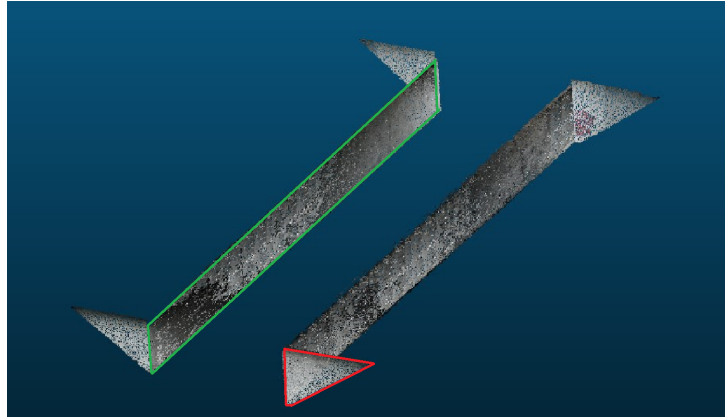


Figure 3.7: Abutment point cloud

- 1) Based on the density of the points, the abutment is separated into individual right and left abutment walls using **DBSCAN** clustering algorithm, which is described in section 2.2.1.
- 2) Once the individual abutment wall is provided, the retaining wall and wing wall are separated from each other based on the **RANSAC** algorithm. See section 2.2.3 for more details about the algorithm.
- 3) The wing and retaining walls are projected onto the y-z-plane and x-y-plane, respectively, to obtain the faces of the walls.
- 4) Due to the fitting algorithm, the boundaries of the wall faces must be detected with a clustering algorithm such as the Fuzzy-C-Means (**FCM**) algorithm. However, this step will be omitted, since in the scope of this work, only the cross-section of the deck is needed.

### 3.2.4 Preparation bridge deck

The following is the description of the preparation steps for the bridge deck. To reduce noise in the projected deck cross-section, the bridge deck is clipped between the abutments. The abutment retaining walls enclose the deck subpart, which is cut out of the whole deck. First, the retaining wall needs to be projected on an x-z-plane and the projected points are sub-sampled, in order to detect their alignment by applying the **PCA**. Based on this alignment, a function for the alignment's graph is generated, and then a query is implemented to check whether the deck points lie between these boundaries. Finally, the clipped bridge deck is projected onto a y-z-plane to obtain the deck cross-section.

### 3.3 Parametric modeling

Once the deck cross-section is extracted from the deck point cloud, the cross-section points are sub-sampled to straighten the cloud points of an edge and avoid overlapping points, making further calculations of the distance between the PPM and cloud points faster and more accurate. In addition, a PPM is generated in a random position and with random dimensions, as shown in figure 3.8. The distance between the sub-sampled point cloud cross section and the PPM needs to be minimized, which therefore presents an optimization problem, which is either solved analytically or numerically. A numerical method is necessary, since describing the shape of a bridge deck using a differential form is, mathematically, highly complex. Particularly, a population-based metaheuristic optimization algorithm is used. A function that generates the PPM is implemented and embedded in the optimization algorithm. Using this function, a population of PPMs is generated in the metaheuristic algorithm. Furthermore, a fitness function is nested within the function that generates the PPM. Therefore, each time a new PPM is generated, the cost value is calculated immediately after, based on the distance between the models. The cost indicates how well the PPM fits the point cloud cross-section. The following paragraphs describe the creation of a PPM, the use of the metaheuristic algorithm, and the fitness function.

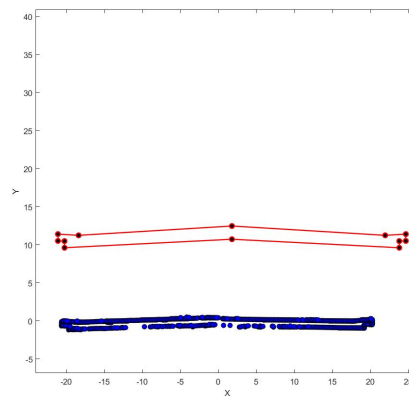


Figure 3.8: Deck point cloud cross section and PPM in random position with random dimensions

### 3.3.1 Generating a parametric prototype model

As described in section 2.1.4, the parametric model is a model defined by parameters and dependencies. The parameters usually represent geometric dimensions, i.e, the position, length, and height of the model. The dependencies form relationships between the parameters and can be defined using custom equations. The parameters are variable, but the dependencies, which define the geometrical element, are constant. Thus, the parametric model can change position and dimension by keeping the same geometrical representation.

This thesis exclusively discusses 2D parametric models. 2D geometric shapes can be defined by a set of points and by lines that connect the points in a closed chain. Similarly, simple shapes like polygons, including triangles, squares, and pentagons, are already classified. A polygon's geometric information remains the same even after relocation, rescaling, and reorientation. Correspondingly, in this thesis, a 2D parametric model, or **PPM**, is created for one specific bridge deck cross-section, as a pre-classified shape.

The required **PPM** is formed based on the observed point cloud cross-section. However, an extracted point cloud cross-section is very noisy and has a distorted shape, thus an approximation of its form is made from engineering knowledge. In addition, appropriate geometric constraints such as symmetry, connectivity, and parallelism are used to build the **PPM**.

After a geometric construct is selected, the shape is generated by starting to define the parameters: x-coordinate of the origin  $x_0$ , y-coordinate of the origin  $y_0$ , length  $l_i$ , height  $h_i$  and, if applicable, angles  $\alpha_i$  of required edges.

Next, the vertices are formulated by an equation for each vertex. First, vertex  $v_1$  is defined by the origin coordinates, as shown in equation 3.1, then recursively parameters like length and height are added to the previous vertex to generate the following vertex, as shown in equations 3.2 and 3.3.

$$v_1 = \begin{pmatrix} x_0 & y_0 \end{pmatrix} \quad (3.1)$$

$$v_2 = \begin{pmatrix} v_1(1) + l_1 & v_1(2) - \tan \vec{\alpha}_1 \cdot l_1 \end{pmatrix} \quad (3.2)$$

$$v_3 = \begin{pmatrix} v_2(1) + l_2 & v_2(2) + \tan \vec{\alpha}_1 \cdot l_2 \end{pmatrix} \quad (3.3)$$

...

Following figure 3.9 represents the resulting **PPM** formed by those equations.

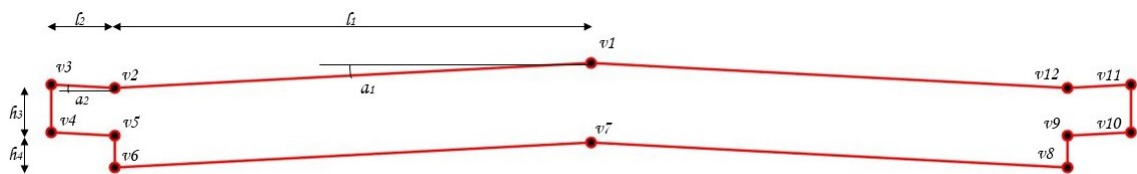


Figure 3.9: A parametric model corresponding to a deck cross-section, or **PPM**

### 3.3.2 Metaheuristic optimization algorithms

As mentioned in section 2.4, a metaheuristic algorithm is a type of optimization algorithm that can generally be used to solve many different and complex optimization problems. Metaheuristic algorithms use stochastic methods to explore the feasible region and find fitting solutions, unlike classic optimization algorithms, which often rely on mathematical models to find an exact solution. The key component of a metaheuristic algorithm is the fitness function, which is used to evaluate the quality of candidate solutions generated by the algorithm. The fitness function gives each solution a numeric rating, or cost, based on how well the solution meets the objectives and constraints of the problem.

The fitness function is always problem-specific, and in this application case, the function defines how well the **PPM** fits the point cloud cross-section. In this particular fitting problem, the fitness function considers the distance between the **PPM** and the point cloud cross-section. This function plays an important role in guiding the search for better solutions. The metaheuristic algorithm uses the fitness function to evaluate the quality of each candidate solution and decide which **PPM** is the closest to the point cloud. The best solution is used for further exploration, while solutions with high costs are discarded. Based on the best candidate, the search operators of the metaheuristic algorithm modify the current solution. Thus, the parameters of the **PPM** are minimized or maximized toward the parameters of the best **PPM** by adding or subtracting an individually calculated value from each parameter.

Embedded in the metaheuristic algorithm, a population of **PPMs** is initialized, by creating a list, in which every item consists of a set of parameters. Thus, every set of parameters represents a **PPM** with different dimensions. These parameters are dimensionally constrained in the optimization algorithm. Dimensional constraints are very important. On the one hand, they control the geometric shape and add engineering knowledge to the geometric shape, and on the other hand, they improve the performance of the optimization algorithm.

In terms of adding engineering knowledge, for example, the German standard for bridge construction, Richtlinien für die Anlage von Stadtstraßen (RASt), states that a road slope should be greater than 2.5 % for drainage reasons. Therefore, the parameter defining the angle can be limited to greater than 1.4 °. In addition, a **PPM** of a deck cross-section with no dimensional constraints can lead to the generation of non-sense geometric shapes. For example, a pavement section could be constructed where the pavement is wider than

the roadway. The implementation of dimensional constraints prevents such nonsensical profiles.

Regarding the optimization algorithm, the efficiency of the optimization algorithm depends directly on the dimensionality of the search space. As dimensionality increases, the proportion of good solutions in the search space decreases, so the algorithm spends more time exploring search spaces that do not contain optimal solutions. This makes it more difficult for the algorithm to efficiently identify and explore promising regions, instead quickly leading to suboptimal solutions. The dimensional constraints of the PPM limit the search space.

Hence, the ability of the PPM to fit the point cloud correctly depends mainly on the adjustment of the dimensional constraints of its parameters for the initial PPMs in the optimization algorithm. In general, the stricter the constraints are, the faster and better the model is fitted. To fit a PPM in the optimization algorithm, the dimensional constraints of the parameters are optimally constrained in a part of the bounding box parameters of the point cloud cross-section. Therefore, before generating the initial population, the bounding box is calculated and the maximum and minimum values for all parameters of the PPM are defined based on the bounding box parameters and engineering knowledge.

The bounding box with length  $l_{bb}$  and height  $h_{bb}$  is a rectangular box enclosing the point cloud cross-section. The lower left and upper right corners are determined by calculating the smallest and largest x and y values of the point cloud, as shown in figure 3.10.



Figure 3.10: An illustration of the bounding box of a point cloud cross section

Figure 3.11 shows the fitting process of the same bridge deck in two scenarios, the upper one with small dimensional constraints and the lower one with larger dimensional constraints. In the upper illustration, the dimensional constraints for all parameters representing a length  $l_i$  or height  $h_i$  in the PPM were limited to the length  $l_{bb}$  and height  $h_{bb}$  of the bounding box, resulting in a suboptimal solution. In contrast, in the lower illustration, the dimensional constraints of the parameters are each determined as an individual portion of  $l_{bb}$  and  $h_{bb}$ , leading directly to the optimal solutions. The portion of the  $l_{bb}$  or  $h_{bb}$  that defines the dimensional constraint should be chosen as accurately as possible, so engineering knowledge is taken into account.

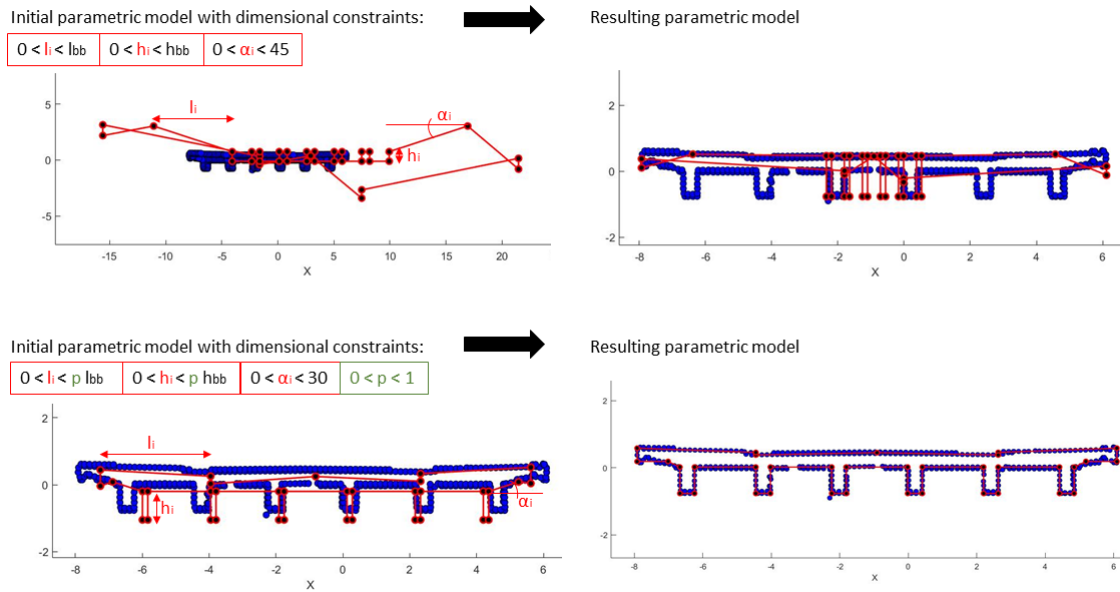


Figure 3.11: Fitting process with small and big dimensional constraints

### Teaching-learning-based optimization

Teaching-learning-based optimization (TLBO) was proposed by RAO et al. (2011). The TLBO algorithm is a metaheuristic optimization algorithm inspired by the teaching and learning processes observed in a classroom.

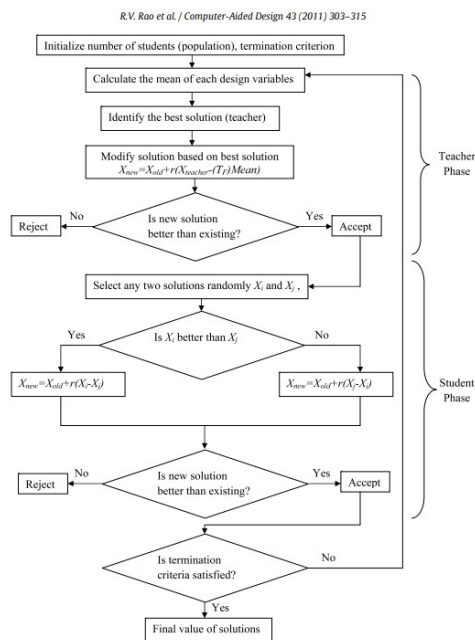


Figure 3.12: An example of the flowchart for TLBO, from RAO et al. (2011)



The following assumptions are made for the algorithm. The population is a group of students. The results of the learners are considered as the values of the fitness function. The smartest of the population, or the one with the best fitness function value, is the teacher. The different design variables that define the optimization problem, here the parameters, are considered the different subjects. The teacher aims to increase the mean value of the student's result by trying to move it towards his own state of knowledge, that is, towards his value of the fitness function. In this process, the student's mean state of knowledge is increased by the teacher and its quality, but also depending on their own capacity. The algorithm is divided in two main phases: the teaching and the learning phase.

In the teacher phase, the teacher wants to bring the students up to his knowledge, but the teacher can only increase the average knowledge of the students based on the capacity of the students. Thus,  $M_i$  is considered as the mean of the population or students and  $M_{new}$  is the value of the teacher state of knowledge and the mean of the population  $X_i$  needs to be moved towards  $M_{new}$ . Therefore, a step is calculated based on the difference between the mean of the population  $M_i$  and the value of the teacher  $M_{new}$ , with some randomization. Next, this step is added to the population  $X_i$ . The step  $dmaen_i$  is computed as follows

$$dmaen_i = r_i(M_{new} - T_f M_i)$$

where  $i$  is the iteration step,  $r_i$  is a random number from 0 to 1,  $T_f = round[1 + rand(0, 1)2 - 1]$

Then it is added to the population  $X_i$ .

$$X_{new,i} = X_{old,i} + dmaen_i$$

In the learning phase, students learn either through the teacher's input or through interaction with other students. If there is one student who has more knowledge, the others can benefit and learn from him. For the implementation of this phase, two random students are selected and moved toward each other (RAO et al., 2011).

In contrast to the other metaheuristic algorithms, the TLBO is simple and easy to implement. It has fewer parameters and high accuracy, good convergence performance, and good robustness on optimization problems, that's why the TLBO algorithm was chosen for testing the proposed method in this thesis (XUE and WU, 2019).

### 3.3.3 Fitness function

The following steps for programming the fitness function were proposed by MAFIPOUR et al. (2021). First, the Euclidean distance from every point in the point cloud cross section  $p_i$  to the PPM is calculated. If  $p_i$  is in the line segment between the vertices from the PPM  $v_i$  and  $v_{i+1}$ , the shortest distance between the point and line segment is calculated. If the point  $p_i$  is not within the line segment, the distance between the point and nearest vertex from the PPM is calculated. Figure 3.13 shows how to calculate the required distances simplified on a rectangle.

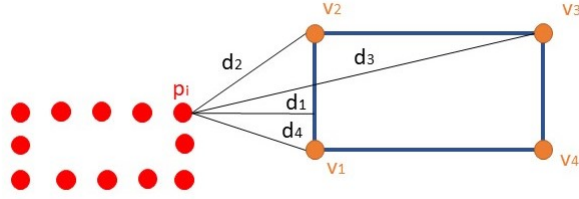


Figure 3.13: Euclidean distances from one  $p_i$  to all edges of the PPM

Following, the smallest of these distances is chosen and saved in the variable  $e_i$ .

$$e_i = \min(d_1, \dots, d_i) \quad (3.4)$$

Two nested loops are implemented to program the fitness function. The outer loop iterates over all points of the point cloud cross section  $p_i$  and the inner loop iterates over all edges of the PPM. In the inner loop, the minimum Euclidean distance from each point  $p_i$  to the PPM is calculated and the index of the PPM edge to which the distance  $e_i$  is calculated is stored in a variable. An average of these distances is needed for the fitness function. Therefore the root-mean-square error (RMSE) of the distances  $e_i$  can be determined by the following equation 3.5

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n e_i^2} \quad (3.5)$$

where  $n$  is the number of points in the point cloud section. However, an inverse expression of this average distance, expressed by the RMSE, is needed for further processing. A variable  $e_j$  is determined, which stores the distances from an edge to all nearest cloud points. The RMSE can now be expressed by summing the  $e_j$  values instead of the  $e_i$  values, as the following equation 3.6 shows

$$RMSE = \sqrt{\frac{1}{k} \sum_{j=1}^k e_j^2} \quad (3.6)$$

where  $k$  is the number of edges of the PPM.

The goal is to define the cost value mentioned in section 3.3.2 by the fitness function. If the cost value is high, the PPM will not be fitted into the point cloud at all, but as the cost value becomes smaller, the PPM will fit better into the point cloud section. The equation 3.6 comes close to this goal by describing an average of the distances between the PPM and each cloud point. However, one part is missing. PPMs that are not close to the cloud section usually have a smaller cost than it should be, leading to false results. This problem is due to the fact that the inner loop always chooses the smallest distance. Thus, when looking at the figure 3.13, the distances calculated from the points  $p_i$  refer only to the first PPM edge between  $v_1$  and  $v_2$ . To avoid this problem, in the inner loop where the minimum distance  $e_i$  is stored, the edge assigned to  $p_i$  is stored in a separate vector, and for each PPM edge, a vector is defined where the points are stored that calculate the minimum distance to that edge. MAFIPOUR et al. (2021) introduced the concepts of active and passive edges. Active edges are assigned to more than one point  $p_i$ , and passive edges are not assigned to any point  $p_i$  during the computation of the minimum distances. In the second part of the cost function, a loop is implemented that iterates over each edge of the PPM and checks whether the edge is active or passive. If a passive edge is detected, the cost value of the PPM is smaller than it should be. Therefore, an artificial, correcting value ( $\lambda_i e'_i$ ) is added to the cost. This discredits the PPMs that mislead the metaheuristic algorithm.

$$cost = \sqrt{\frac{1}{k} \sum_{j=1}^k \omega_j (e_j + \lambda_j e'_j)^2} \quad (3.7)$$

The equation 3.7 determines the final mathematical formulation of the cost value calculated in the fitness function with the weighting factor  $\omega_j$ . The weighting factor  $\omega_j$  is larger for edges in the cloud section that contains fewer points and smaller for edges that contain more points, so the distribution of points in the cloud section is also taken into account.

## Chapter 4

# Results/ Discussion

The parametric model fitting algorithm, described in chapter 3 was tested on six bridge decks with different cross sections to evaluate its accomplishment. The computational method for parametric model fitting was implemented in Matlab on a laptop computer (CPU: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99GHz, Memory: 16.0 GB, SSD: 512 GB). After the cross-section is extracted from the deck point cloud, the corresponding PPM is fitted to the deck cross-section using the TLBO optimization algorithm. Furthermore, the evaluation metrics and test results are presented and discussed.

### 4.1 Evaluation metrics

As described in section 3.3.1, the PPM is an approximation and does not have exactly the same form as the point cloud cross section because the point cloud has a large amount of noise and redundancy. Therefore, the fitted model does not go through every point and there is a small distance between some cloud points and the PPM. The smaller the sum of these distances, the better the fitting of the PPM. This quality of the generated model is gauged by the MAE of the distances of all cloud points to the PPM.

The MAE is computed by a function in Matlab that implements two loops. The Euclidean distances  $d_j$  from a cloud point to each edge of the PPM is calculated in the inner loop. The smallest of these distances  $d_j$  is then defined as the minimum Euclidean distance  $dmin_i$ . The outer loop iterates over each point of the point cloud, calculating the minimal Euclidean distance  $dmin_i$  from each cloud point to the PPM. All of these minimum distances  $dmin_i$  are summed and divided by the number of cloud points  $n$  to calculate the MAE, as defined by the following equation.

$$MAE = \frac{\sum_{i=1}^n dmin_i}{n}$$

## 4.2 Real world applications

Figure 4.1 shows the six point clouds of the test bridges with different shapes and point densities, excluding their environment. The PCD of the Bridge 1 consists of 20 LiDAR scans from different positions around the bridge. These scans were first registered in Recap, then the aligned and complete bridge point cloud was manually segmented into the subparts environment, deck, railings, and piers. After downsampling, the bridge deck still has a high point density of 4,617,080 points. Some points are missing, especially on the upper side in the right and left corners, where the railings are usually located, and the lower side also has some missing points. In general, the bridge deck has relatively little noise and outliers after processing, and almost all points are in the right place, only the left and right edge caps have some redundancies and misalignment.

The next three Bridges, 2, 3, and 4, are from LU and BRILAKIS (2019), which already were segmented. Bridge 2 has 425,892 points, Bridge 3 has 498,584 points, and Bridge 4 has 506,081 points along the entire bridge deck. All three bridges only have missing points at the upper left and right edges of the deck capes. However, they all have more outliers and noise compared to the other datasets. Also noteworthy is that Bridge 4 actually has three additional stiffeners that have been omitted from parametric modeling.

Bridge 5 has 829,698 points and bridge 6 has 1,436,546 points. Both bridges have several places where points are missing, the whole bottom surface of the deck has a much lower point density than the upper surface, and also the upper surface, since it consists of two separate roads, has hardly any points in the space between the roads. In particular, Bridge 6 has no points at all in the upper surface between the roads, which makes it more difficult to fit a model.

All point clouds are saved as a text file in a list with six columns and one row for each point, so in each row in the first three columns, the x, y, and z coordinates are saved and in the last three columns, the RGB color code is saved.

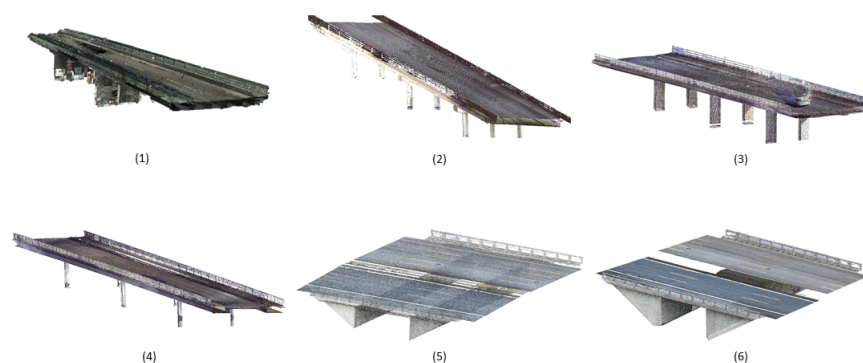


Figure 4.1: Test bridges

The preparation steps for extracting a point cloud deck cross-section and the parametric model fitting algorithm described in chapter 3 were implemented in Matlab. After importing the segmented bridge PCD into Matlab, the last three columns of the imported text file containing the RGB color code information are deleted. Thus, the PCD is stored in a list with three columns for the x, y, and z coordinates of the points.

The first step is to extract the cross-section of the deck point cloud. For some bridges, the segmented deck point cloud needs to be clipped between the abutments in advance. Therefore, the alignments of the retaining walls must be detected. For this reason, the abutments must first be separated using DBSCAN. When implementing DBSCAN, the threshold for a neighborhood search radius epsilon is defined by  $\epsilon = 1$ . The higher the threshold, the faster the abutments will be clustered, but at the same time, the threshold cannot include both abutments in the same cluster. After segmenting the abutments, the two wing walls and the retaining wall have to be separated. To do this, the RANSAC plane detection algorithm is implemented. The RANSAC algorithm for plane detection has an inner loop. With each new iteration, a new plane is generated and the inliers of this plane are counted. At the end, the plane with the highest number of inliers is saved as the best plane. In this loop, two input parameters must be defined. First, the maximum distance from the randomly generated plane in which a point is counted as an inlier; this distance is set to 10 cm. Then, a maximum number of iterations as a suitable termination criterion. This maximum number of iterations is set to two times the number of abutment points. An additional termination criterion is set by defining a percentage of maximum covered points by a plane, so if a plane is found before the last iteration is finished that covers more than 60 percent of all points, this plane can be determined to be the retaining wall, because looking at the whole point cloud, the retaining wall covers about 70 percent of all points, this additional criterion can save computation time. Additionally, an outer loop repeats the search for the best plane in the point cloud as many times as the number of planes searched, i.e. in the observed case, three planes are searched. Thus, the three best-fitting planes are found and each plane clusters its inliers to an entity or wall. After clipping the deck point cloud between the retaining walls, the clipped deck point cloud is projected onto the y-z-plane, and the cross-section is obtained.

The second step is to fit a parametric model to the point cloud cross-section. To do this, the TLBO optimization algorithm is implemented in Matlab. In the TLBO, a population of 50 PPM is generated and 100 iterations are performed. Additionally, within the optimization algorithm, the bounding box of the point cloud cross section is determined, which is used to define some dimensional constraints for the parametric models.

Figure 4.2 shows the fitted PPM in the point cloud cross sections. Tables 4.2 and 4.1 summarize the runtime of the algorithm and the MAEs of the fitted bridges and the average values. The method achieves an average modeling distance of 3.675 cm and an average modeling time of 114.775 s.

Table 4.1: MAE [cm] of fitted bridge decks

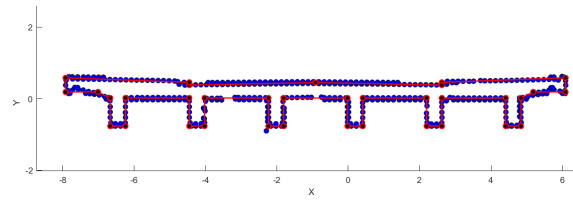
Bridge deck 1	Bridge deck 2	Bridge deck 3	Bridge deck 4	Bridge deck 5	Bridge deck 6	average MAE
2.6	3.2	5.6	3.74	4.6	5.5	4.2

Table 4.2: Time [sec] elapsed during fitting process

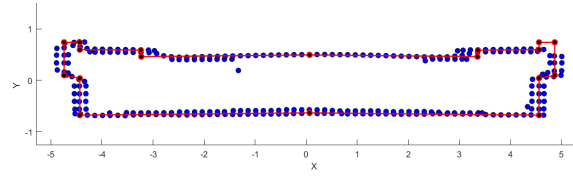
Bridge deck 1	Bridge deck 2	Bridge deck 3	Bridge deck 4	Bridge deck 5	Bridge deck 6	average time
117.2	52.3	80.2	60.2	84.2	150.7	90.8

Table 4.3: Number of points in the subsampled deck points cloud cross-section

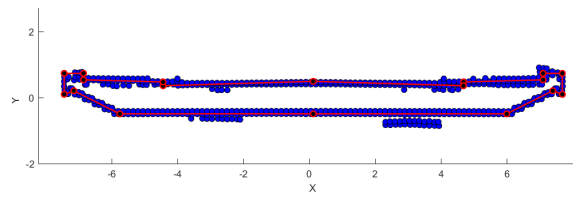
Bridge deck 1	Bridge deck 2	Bridge deck 3	Bridge deck 4	Bridge deck 5	Bridge deck 6
411	273	504	333	745	1287



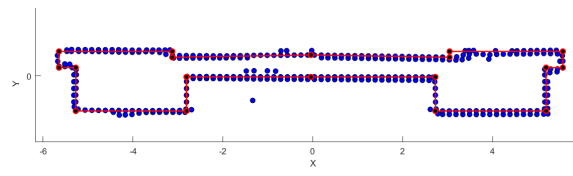
(a)



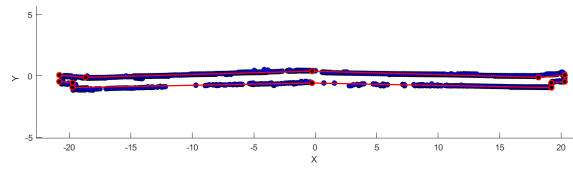
(b)



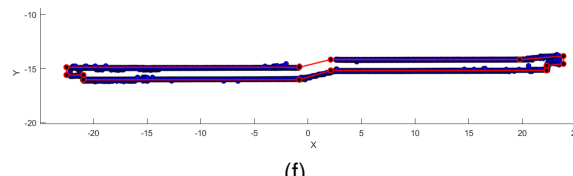
(c)



(d)



(e)



(f)

Figure 4.2: Fitted parametric models: (a) bridge deck 1; (b) bridge deck 2; (c) bridge deck 3; (d) bridge deck 4; (e) bridge deck 5; (f) bridge deck 6



After fitting the PPMs, a volumetric body can be created by extruding the surface of the fitted parametric model, as shown in the figure 4.3. In Matlab, the parametric model is first triangulated and the surface is then saved in STL format. This surface is then extruded along the axis using the Matlab extrude function.

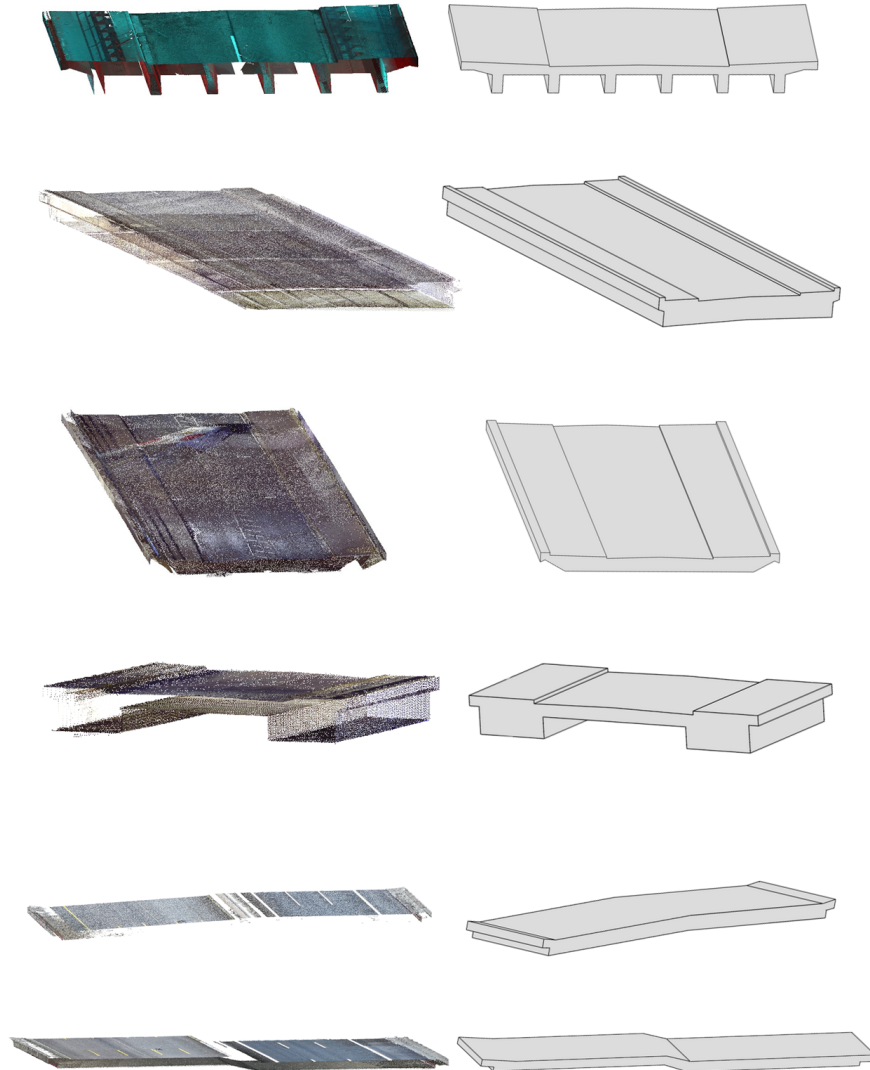


Figure 4.3: Illustration of a deck point cloud compared to the corresponding automatically generated 3D geometric model

### 4.3 Discussion

Observations of the results in Tables 4.1 and 4.2 indicate that the MAE and parametric model fitting time vary from bridge to bridge. Bridge decks 3 and 6 have the highest MAE values, while bridge decks 1 and 6 have the longest fitting time.

There are several factors that affect the height of the MAE. In particular, the noise and outliers of the point cloud have a large influence on the MAE. As described in section 3.1, outliers are points that are far from the true shape and their density is usually less than the density of the points that mark the true surface. When the bridge deck is fitted in the correct position, the PPM is aligned with the points that mark the true shape, so the outliers have a relatively large distance from the PPM. Consequently, the more outliers a point cloud cross section has, the higher its MAE for the fitted parametric model, as observed in the tests of bridge deck 3 and bridge deck 6. In this particular case, the higher MAE does not imply that the PPM is incorrectly fitted, on the contrary, the PPM should preferably not pass through the outliers, hence a distance is expected.

The time it takes the objective function to compute the distances from the cloud points to the PPM edges is the main determinant of the fitting time. The speed of this calculation depends on the number of points in the point cloud and the number of edges in the PPM or its complexity. For example, bridge deck 1 has a complex geometry with 35 edges, and bridge deck 6 has the largest number of points in the point cloud cross-section, so these two decks require more time to be fitted.

The following section will focus on comparing the novel parametric model fitting algorithm presented in this thesis with a more common method of shape analysis, the convex hull. A convex hull is a fundamental concept in computational geometry, which refers to the smallest convex polygon or polyhedron that encloses a given set of points in Euclidean space. More simply, this hull is the smallest convex shape that covers all the points of a set. There are several geometric algorithms, such as the Graham algorithm, the gift wrapping algorithm, and the Quickhull algorithm, that can determine the convex hull of a point set. For the following comparison, the cloud points of the convex hull of a deck point cloud cross section are detected by the Matlab function boundary. These points are then connected by piecewise linear simple curves in the Euclidean plane to represent the shape of the bridge deck, as shown in the figures.

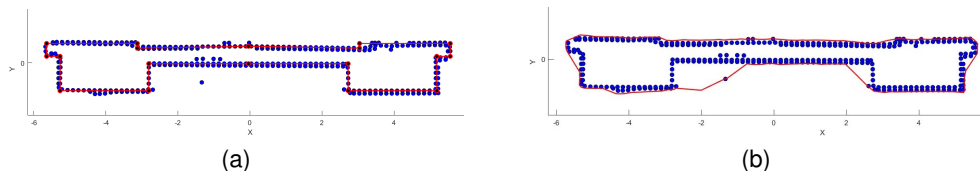


Figure 4.4: Comparison between the parametric model fitting (a) and the convex hull (b)

A fundamental difference between the convex hull and parametric model fitting is that the parametric model is fitted to a point cloud by minimizing the average distance to all cloud

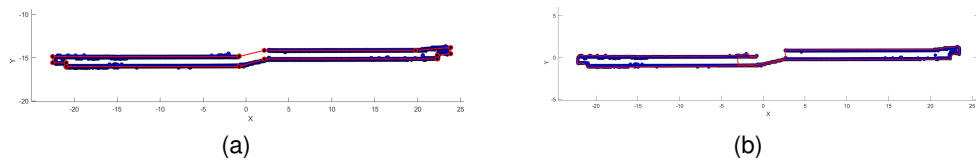


Figure 4.5: Comparison between the parametric model fitting (a) and the convex hull (b)

points, and the convex hull searches for a hull that includes the extreme boundaries of the point cloud. Therefore, the convex hull is very sensitive to imperfections in the point cloud, such as noise, outliers, redundancy, and missing points.

Figure 4.4 show the effect of outliers on the convex hull shape representation, as one outlier point makes the shape representation unusable. In the parametric model fitting, noise has a relative effect. That is, if the density of outliers is less than the density of points at the correct location, the parametric model will fit where the density of points is higher. This is the reason why the parametric model fitting is a better representation of the true shape in the figure 4.4 in illustration (a) because the point cloud usually has the highest point density where the true shape is. In illustration (b) of figure 4.5, the missing points result in an incorrect representation of the cross-section. Parametric fitting is very tolerant of missing points because the shape of the cross-section is predefined, and all import geometric dependencies between edges, such as connectivity, symmetry, and closure, are stored.

A very useful feature is the extraction of dimensional parameters with parametric model fitting. In contrast, the convex hull doesn't provide any information about the dimensions of the component. The parametric model provides all relevant dimensions of the component that are important for further applications.

## Chapter 5

# Conclusion

This thesis presents a novel algorithm for automatically reconstructing bridge components from **PCD**. An important step towards more efficient and accurate digitized bridge maintenance is the digital twinning of bridges. However, there are still not many methods capable of generating an arbitrarily shaped bridge model from **PCD**. To advance this process, MAFIPOUR et al. (2021) have proposed a novel parametric model fitting algorithm capable of fitting complex shapes to point clouds. This method extends the conventional methods widely used in the construction industry, which can only fit primitives to point clusters.

The novelty of this approach is that the fitting problem is first reduced to a 2D fitting problem. Then a **PPM** is fitted to the 2D point cloud section by applying a metaheuristic optimization algorithm. Essentially, the method consists of two steps, first extracting the point cloud cross section and second generating a parametric model corresponding to the observed section, which is fitted to the point cloud section by a metaheuristic optimization algorithm. In this thesis, a segmented bridge point cloud is a prerequisite, and the proposed fitting method was tested on six different bridge decks with arbitrary shapes. All six bridge decks were successfully fitted with high accuracy.

The results show that the proposed fitting algorithm is capable of fitting arbitrarily shaped models to point clouds, thereby extracting important dimensional information about the component. In addition, the effects of **PCD** imperfections, such as noise, missing points, and outliers, are handled very well compared to other methods. The accuracy of the method can compete with the accuracy of manual model fitting, while the processing time is much faster than the manual method, as the comparisons with the results from MAFIPOUR et al. (2021) can show. However, the most important achievement is that important information about dimensional parameters can be extracted from the bridge component, and a useful 3D model is automatically generated for further applications.

The metaheuristic optimization algorithm used in the fitting process has its limitations, and those limitations are listed.

- 1) The more parameters a **PPM** has, the harder the optimization algorithm will try to make the **PPM** fit the cloud correctly. However, if the correct dimensional constraints are added to the parametric model, fitting of very complex shapes, such as the bridge decks shown here, can be achieved.
- 2) Imperfections of the point cloud, such as noise, outliers, and missing point data, have a distorting effect on the fitting process, but the effect is relatively small compared to other common methods, such as convex hull detection.

For future work, following some ideas of extensions of the proposed method are listed.

- 1) Extending the preparation steps for curved alignments; the article from LU and BRILAKIS (2019) describes a way to get the correct cross-section of the curved bridge deck by slicing the bridge in pieces, which could be applied to the preparation steps of a bridge point cloud with curved alignment. Once we have the cross-section even for curved bridges, the parametric modeling, and model fitting would remain the same.
- 2) A library could be created containing several standard PPMs so that the parametric models would be automatically taken from this library. To achieve this, for example, an algorithm could be implemented that classifies the observed point cloud cross section by comparing it to all the PPMs in the library. In addition to geometric information such as volumetric dimensions, such classification would provide important semantic information about the type of bridge.

As the results show, the proposed method can fit arbitrary models to point clouds, extract important dimensional parameters, and enable automatic modeling of a 3D model from PCD. With some future extensions, such as the implementation of a PPM library and slicing methods in the preparation steps, this method could be able to generate geometrically and semantically rich models for further simulations, failure prediction, and optimization of bridge maintenance.

# Bibliography

- BOARD, U. S. N. T. S. (2008). *Collapse of i-35w highway bridge, minneapolis, minnesota, august 1, 2007*. Createspace Independent Publishing Platform.
- BORRMANN, A., & BERKHAHN, V. (2018). Principles of geometric modeling. *Building Information Modeling: Technology Foundations and Industry Practice*, 27–41.
- BRÉMAUD, P. (2017). Discrete probability models and methods. *Springer*. [https://doi.org/10.1007/978-3-319-43476-6\\_10](https://doi.org/10.1007/978-3-319-43476-6_10), 978–3.
- BRONSTEIN, I. N., SEMENDJAJEW, K. A., MUSIOL, G., & MÜHLIG, H. (1989). Taschenbuch der mathematik, 24. Auflage, Thun u. Frankfurt/Main: Harri Deutsch.
- CROCE, V., CAROTI, G., PIEMONTE, A., DE LUCA, L., & VÉRON, P. (2023). H-bim and artificial intelligence: Classification of architectural heritage for semi-automatic scan-to-bim reconstruction. *Sensors*, 23(5), 2497.
- DERPANIS, K. G. (2010). Overview of the ransac algorithm. *Image Rochester NY*, 4(1), 2–3.
- DOKEROGLU, T., SEVINC, E., KUCUKYILMAZ, T., & COSAR, A. (2019). A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering*, 137, 106040.
- für DIGITALES UND VERKEHR, B. (2019). Brücken: Zahlen, daten, fakten [Online; accessed 30-Mai-2023]. <https://bmdv.bund.de/SharedDocs/DE/Artikel/StB/bruecken-zahlen-daten-fakten.html>
- GANDOMI, A. H., YANG, X.-S., TALATAHARI, S., & ALAVI, A. H. (2013). Metaheuristic algorithms in modeling and optimization. *Metaheuristic applications in structures and infrastructures*, 1.
- HELLE, R. H., & LEMU, H. G. (2021). A case study on use of 3d scanning for reverse engineering and quality control. *Materials Today: Proceedings*, 45, 5255–5262.
- HOSAMO, H. H., & HOSAMO, M. H. (2022). Digital twin technology for bridge maintenance using 3d laser scanning: A review. *Advances in Civil Engineering*, 2022.
- ISAILOVIĆ, D., PETRONIJEVIĆ, M., & HAJDIN, R. (2019). The future of bim and bridge management systems. *Proceedings of the IABSE Symposium*, 3.
- LU, R. (2019). *Automated generation of geometric digital twins of existing reinforced concrete bridges* (Doctoral dissertation). University of Cambridge.
- LU, R., & BRILAKIS, I. (2019). Digital twinning of existing reinforced concrete bridges from labelled point clusters. *Automation in construction*, 105, 102837.
- MAFIPOUR, M. S., VILGERTSHOFER, S., & BORRMANN, A. (2021). Deriving digital twin models of existing bridges from point cloud data using parametric models and metaheuristic algorithms. *EG-ICE 2021 Workshop on Intelligent Computing in Engineering*, 464.
- MURPHY, K. P. (2022). *Probabilistic machine learning: An introduction*. MIT Press. [probml.ai](https://probml.ai)

- QIN, G., ZHOU, Y., HU, K., HAN, D., & YING, C. (2021). Automated reconstruction of parametric bim for bridge based on terrestrial laser scanning data. *Advances in Civil Engineering*, 2021, 1–17.
- RABBANI, T. (2006). Automatic reconstruction of industrial installations using point clouds and images. *Publications on Geodesy*, 62.
- RAO, R. V., SAVSANI, V. J., & VAKHARIA, D. (2011). Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems. *Computer-aided design*, 43(3), 303–315.
- REBALA, G., RAVI, A., CHURIWALA, S., REBALA, G., RAVI, A., & CHURIWALA, S. (2019). Machine learning definition and basics. *An introduction to machine learning*, 1–17.
- SAIGA, K., ULLAH, A. S., KUBO, A., et al. (2021). A sustainable reverse engineering process. *Procedia CIRP*, 98, 517–522.
- SHAPIRO, V. (2002). Solid modeling. *Handbook of computer aided geometric design*, 20, 473–518.
- STEIN, S. (2014). Eine kleine latex (tex) einföhrung [Stand: 2014-10-21].
- UY, M. A., CHANG, Y.-Y., SUNG, M., GOEL, P., LAMBOURNE, J. G., BIRDAL, T., & GUIBAS, L. J. (2022). Point2cyl: Reverse engineering 3d objects from point clouds to extrusion cylinders. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11850–11860.
- VALERO, E., ADÁN, A., & CERRADA, C. (2012). Automatic method for building indoor boundary models from dense point clouds collected by laser scanners. *Sensors*, 12(12), 16099–16115.
- WALKER, M. (2022). *Data cleaning and exploration with machine learning: Get to grips with machine learning techniques to achieve sparkling-clean data quickly*. Packt Publishing Ltd.
- WALSH, S. B., BORELLO, D. J., GULDUR, B., & HAJJAR, J. F. (2013). Data processing of point clouds for object detection for structural engineering applications. *Computer-Aided Civil and Infrastructure Engineering*, 28(7), 495–508.
- WANG, Q., TAN, Y., & MEI, Z. (2020). Computational methods of acquisition and processing of 3d point cloud data for construction applications. *Archives of computational methods in engineering*, 27, 479–499.
- XUE, R., & WU, Z. (2019). A survey of application and classification on teaching-learning-based optimization algorithm. *IEEE Access*, 8, 1062–1079.

# Declaration

I hereby affirm that I have independently written the thesis submitted by me and have not used any sources or aids other than those indicated.

\_\_\_\_\_  
Lc